T-SMC/21/4//40989D

Visualization of Structural Information: Automatic Drawing of Compound Digraphs

Kozo Sugiyama
Kazuo Misue

Reprinted from
IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS
Vol. 21, No. 4, July/August 1991

Visualization of Structural Information: Automatic Drawing of Compound Digraphs

Kozo Sugiyama and Kazuo Misue

Abstract— **An automatic method for drawing compound digraphs that contain both inclusion edges and adjacency edges are presented. In the method vertices are drawn as rectangles (areas for texts, images etc.), inclusion edges by the geometric inclusion among the rectangles, and adjacency edges by arrows connecting them. Readability elements such as drawing conventions and rules are identified and a heuristic algorithm to generate "readable" diagrams is developed. Several applications are shown to demonstrate the effectiveness of the algorithm. The utilization of curves is investigated to improve the quality of diagrams. A possible set of command primitives for progressively organizing structures within our graph formalism is also discussed. The computational time for the applications shows that the algorithm achieves satisfactory performance.**

I. Introduction

THE EFFECTIVENESS of visualizing structural information, in general, or drawing graphs, specifically, is widely recognized and various diagrams are utilized in diverse fields of research and development. Since numerous variations of drawings are possible for a given structure, we need certain criteria to evaluate the quality of drawings.

In considering the criteria, we should distinguish two categories of drawings: 1) emphasizing aspects of physical implementation (e.g., resource economy, reliability) in realizing physical layouts (e.g., LSI design), and 2) emphasizing aspects of human visual cognition (e.g., readability, aesthetics) in communicating conceptual notions (e.g., database schema). Though there exist criteria common to both categories such as *line-crossings* and global length of lines, it should be noted that they have different meanings.

Many algorithms for automatically drawing graphs from the cognitive viewpoint have already been proposed so far. Extensive surveys have been conducted by Eades *et al.* [1], Tamassia *et al.* [2], Sugiyama [3], [4] and Messinger [5]. These algorithms can advance graphic facilities to enhance the human thinking processes, since they may release us from nonintrinsic operations in generating and editing diagrams, and enable us to engage in thinking itself. However, the class of the graphs treated in the algorithms have been rather restricted to ordinary directed and undirected graphs in which only adjacency among vertices is considered. The class of diagrams we utilize in research and development activities and daily work is usually much larger; i.e., diagrams representing both *Inclusive* and adjacent relations among vertices.

Manuscript received February 24, 1989; revised January 6, 1990 and October 6, 1990. This paper was partially presented at the Third International Conference on Human–Computer Interaction, held in Boston, MA, September 18–22, 1989.

The authors are with the International Institute for Advanced Study of Social Information Science, Fujitsu Ltd., 140 Miyamoto, Numazu, Shizuoka 410-03, Japan.

IEEE Log Number 9040989.

In this paper, from the cognitive viewpoint we consider an automatic method for drawing a *compound directed graph* (or *compound digraph*) with both *inclusion edges* and *adjacency edges* [6]–[8]. In the method vertices are drawn as rectangles (areas for representing texts, images, etc.), inclusion edges are expressed by *Inclusive* relations among the rectangles, and adjacency edges by arrows connecting the corresponding pairs of vertices. The whole algorithm consists of four steps: hierarchization, normalization, vertex ordering and metrical layout. We have already developed an algorithm for drawing a directed graph [9], [10] based upon Warfield [11] and some improvements of the algorithm have been carried out by Messinger [5], Rowe *et al.* [12] and Gansner *et al.* [13]. Though the algorithm for a compound digraph is developed as an extension of the algorithm for a directed graph, the former is much more complicated than the latter.

The drawings of graphs are called *maps*. Maps of compound digraphs are widely used in diverse fields as "tools" for enhancing human thinking; existential graph [14] in logics, KJ method [15] in creativity engineering, associative networks [16] and conceptual graph [17] in knowledge engineering and so on. We call such methods *diagrammatical thinking methods* [3]. Both the extension of formalism and the automatic drawing capability presented in this paper will attain an effective integration of human thinking and machine production of maps in the development of novel *computer-aided diagrammatical thinking systems*. They will also expand the use of visual systems and/or interfaces for databases, knowledge bases, expert systems, idea processors, design systems, decision support systems and so on.

In Section II, our approach to the problem is described. The four steps of the algorithm are presented in the succeeding four sections. In Section VII, several applications are presented and runtime performance of the algorithm is evaluated. The utilization of curves is investigated to improve the readability of maps and a possible set of primitives for progressively organizing structures within our graph formalism is also shown. Finally, concluding remarks are made with suggestions for future research, where the importance of investigations on how to use the automatic drawing capabilities in dynamic thinking processes under more general visual formalisms [18] is emphasized.

II. How to Approach the Problem

In general, for designing algorithms to draw graphs, it is necessary to clarify formalisms of drawn objects, analyze elements affecting the quality of maps and specify features of algorithms to be developed. In this section we describe the class of graphs to be drawn, readability elements to be attained and theoretical and heuristic aspects of algorithms to be developed.

A. Class of Graphs

When two kinds of binary relations, inclusion and adjacency relations, are defined on a finite set V of vertices, we can introduce two specific directed graphs (or digraphs) corresponding to the relations. An *inclusion digraph* is a pair $D_c = (V, E)$ where E is a finite set of *inclusion edges* whose element $(u, v) \in E$ means that u includes v. An *adjacency digraph* is a pair $D_a = (V, F)$ where F is a finite set of adjacency edges whose element $(u, v) \in F$ means that u is adjacent to v. A compound digraph is defined as a triple $D = (V, E, F)$ obtained by compounding these two digraphs.

In this paper, we require two restrictions on a compound digraph D.

Restriction 1: D_c is a tree.

Consequently, D_c is called an *inclusion tree*. In a tree, *parent, ancestors, children*, and *descendants* of vertex v are denoted by $Pa(v)$, $An(v)$, $Ch(v)$ and $De(v)$, respectively. Both ancestors $An(v)$ and descendants $De(v)$ include vertex v. The vertex without a parent is called *root* and a vertex without children a *leaf*. There exists a unique semipath [19] (or path without the property of direction) between any pair of vertices in a tree. The *depth* of any vertex v is the number of vertices on the semipath between v and *root* and is denoted by $dep(v)$; specifically, $dep(root) = 1$.

Restriction 2: In a compound digraph $D = (V, E, F)$ satisfying Restriction 1, the adjacency relation does not exist between any pair of vertices among ancestors and descendants in the inclusion tree, or

$$\{(u, v) \in F \mid v \in An(u) \cup De(u)\} = \phi. \tag{1}$$

Compound digraphs with these restrictions appear in diverse fields as stated in Section I. We can treat more general cases by replacing rectangles intersecting each other with a proxy rectangle and drawing adjacency edges violating Restriction 2 additively on a map of the restricted digraph.

In the succeeding part of this paper, a compound digraph satisfying both restrictions is called a "compound digraph" for simplicity if there is no ambiguity.

In Fig. 1, (a) shows an inclusion tree, (b) an adjacency digraph, (c) a compound digraph obtained by compounding (a) and (b), and (d) a map of the compound digraph drawn automatically and its compound levels. In (d), the adjacency edges drawn with solid lines have downward orientations and the edges drawn with broken lines upward.

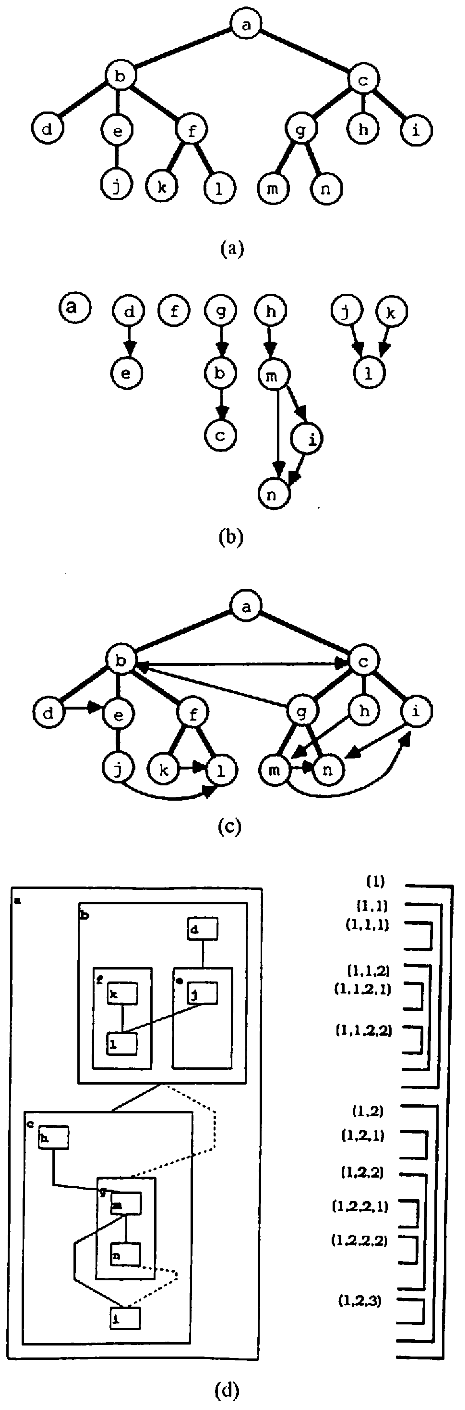


Fig. 1. An example of a compound digraph and its map. (a) Inclusion tree. (b) Adjacency digraph. (c) Compound digraph obtained from (a) and (b). (d) Map of the compound digraph drawn automatically and compound levels.

B. Readability Elements

Strictly speaking, readability of a map depends upon problems being studied and, more intrinsically, on the map's audience. Here, we consider common aspects of the readability that leave little ambiguity in identifying its elements. Readability elements are classified into *drawing conventions* and *drawing rules*: the former consists of fundamental constraints necessarily attained in maps and the latter consists of objectives satisfied as much as possible.

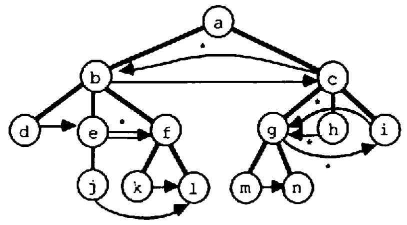


Fig. 2. A derived graph obtained from the compound digraph of Fig. 1(c). (Edges marked with asterisks are derived edges.)

1) Drawing Conventions: We adopt the following conventions for drawing a compound digraph.

C1: *Rectangle*—A vertex is drawn as a rectangle with horizontal and vertical sides.

C2: *Inclusive*—An inclusion edge (u, v) is drawn in such a way that the rectangle corresponding to u includes geometrically the rectangle corresponding to v. Rectangles without inclusion relations should be disjoint to each other.

C3: *Hierarchical*—Vertices are laid out hierarchically in terms of both *Inclusive* and adjacent relations on parallel-nested horizontal bands called *compound levels* (see Fig. 1(d)).

C4: *Down-arrow*—An adjacency edge (u, v) is drawn as a downward arrow (without or with several bending points) originating from the bottom side of the rectangle corresponding to u and terminating on the top side of the rectangle corresponding to v.

Our method is characterized as the *hierarchical* drawing of compound digraphs though other conventions such as positioning vertices on *grids* or *2-D free surfaces* can be possible. The reasons why we adopt hierarchical convention are as follows.

1) It is difficult to readily grasp structures by human eyes without some *regularities*; vertices are laid out and edges are drawn in a certain regular or unified form. Hierarchical layouts attain effective regularities.
2) The drawing algorithm can be made simpler than others.

It should be noted here that we can not always obtain a hierarchical map for any given compound digraph due to the cyclical nature of the digraph and hierarchical convention. In such a case, certain adjacency edges are reversed and in the final drawing step the original orientation of the reversed edge is revived. For example, in Fig. 1(d), reversed edge (g, b) is due to the cyclical nature of the compound digraph (or "compound cycle"), while reversed edge (i, n) is due to the *Hierarchical* convention. (However, these both cases may result in the cyclical nature of the *derived graph* as seen in Fig. 2.)

2) Drawing Rules and Their Priority: Drawing rules are classified into *semantic* and *structural*: the former comes from the meanings of vertices and edges (e.g., relative importances, user's specifications), and the latter comes from structural information itself [2]. In this paper we consider rules only for the latter. We adopt the following structural rules for drawing a compound digraph:

R1: *Closeness*—Vertices connected to each other are laid out as closely as possible. There exist two types of connections:
  a: connections between the inside and outside vertices of some rectangle;
  b: connections among the inside vertices of some rectangle.

R2: *Line-crossing*—The number of crossings among adjacency edges is reduced as much as possible.

R3: *Line-rect-crossing*—The number of crossings between adjacency edges and rectangles is reduced as much as possible.

R4: *Line-straightness*—One-span adjacency edges (or edges between adjacent levels) are drawn as straight lines, and long span adjacency edges are drawn as polygonal lines where the number of bends is reduced and the length of vertical part of the lines is increased as much as possible.

R5: *Balancing*—Edges terminating on and originating from a vertex are laid out in a balanced form.

The priority among these rules is specified as

$$R1 > R2 > R3 > R4 > R5 \tag{2}$$

where $Ri > Rj$ means that Ri has a higher priority than Rj. This priority is empirically justified as follows. The first three $R1$, $R2$, and $R3$ relate mainly to topological features of maps while $R4$ and $R5$ are metrical features. The former rules should be attained before the latter [2]. Among the former rules, the length of the connections between inside and outside vertices ($R1a$) is usually the longest and the length of the connections between same levels ($R3$) usually the least. $R1b$ is realized simultaneously in attaining $R2$ and $R3$. Among the latter rules, we consider that the traceability of edges that is attained by their straightness ($R4$) is more important than the *Balancing* ($R5$).

C. Features of Algorithm

The whole algorithm for drawing a compound digraph consists of four steps similar to those of the algorithm for drawing a digraph that have been developed by one of the authors and his colleagues [9], [10].

Step I: *Hierarchization*—When a compound digraph is given, we begin by checking the possibility of hierarchization. If this not possible, we find *reversed adjacency edges* to assign a compound level to each vertex and obtain an *assigned compound digraph*. The problem of finding the minimum feedback edge set is NP-complete [20]. Consequently, we introduce heuristics for determining reversed edges.

Step II: *Normalization*—An *assigned compound digraph* is converted to a *proper* compound digraph by replacing every nonproper adjacency edge with an appropriate "linear" compound digraph. The time complexity of this step is proportional to the total number of inclusion and adjacency edges of the "linear" compound digraphs produced.

Step III: *Vertex Ordering*—In a proper compound digraph, reordering procedures are applied to *local hierarchies* defined for each vertex from root to leaves. In each local hierarchy, horizontal orders of vertices are determined by permuting orders of vertices in each level of the local hierarchy so as to attain *Closeness*, *line-crossing* and *line-rect-crossing* rules as much as possible. The problem of minimizing *line-crossing* is NP-complete even if the number of levels of the local hierarchy is two [21]. The problem of minimizing *line-rect-crossing* is equivalent to the linear arrangement problem [20], as shown in Section V–B–1), which is NP-complete. Therefore, we develop heuristic methods.

Step IV: *Metrical Layout*—Horizontal positions of vertices are determined by attaining *Closeness, line-straightness*, and *Balancing* rules as much as possible. The orders of vertices determined in Step III are given as constraints to preserve the reduced number of crossings. The problem to attain these rules can be formalized as a quadratic programming problem. We also develop a heuristic method called priority layout method. A map is automatically drawn where the original orientation of reversed edges is revived, and the dummy vertices and edges are deleted and the corresponding edges are regenerated.

III. Hierarchization (Step I)

A. Compound Level Assignment

The drawing conventions C1–C4 require that rectangles corresponding to vertices of a compound digraph are laid out on parallel-nested horizontal bands so that *Inclusive* and *Down-arrow* conventions are satisfied. As seen in Fig. 1(d), the parallel-nested horizontal bands can be expressed by sequences of positive integers. Extending the *level assignment* [19] of a digraph, we consider the possibility of assigning a sequence of positive integers called a *compound level* to each vertex of a compound digraph $D = (V, E, F)$.

Let $\Sigma = \{1, 2, 3, \ldots\}, \Sigma^i = \{$ sequences of i elements of $\Sigma\}$ and $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$. And suppose that the lexicographical order is introduced for any pair of elements of Σ^+; for example, $(1,1,2) < (1,2) < (1,2,1) < (1,2,2)$. Then, our problem is to find a mapping clev: $V \to \Sigma^+$ satisfying *Inclusive* and *Down-arrow* conventions.

Inclusive convention can be easily expressed as the following conditions:

1) For any vertex $v \in V, clev(v) \in \Sigma^{dep(v)}$; (3)
2) For any inclusion edge $(v, w) \in E$,
$clev(w) = append(clev(v), s), s \in \Sigma$. (4)

where *append* is a function that append a component to a sequence. Since the formalization of the *Down-arrow* convention is rather complicated, we need some explanation. For any adjacency edge $e = (v, w) \in F$, we can uniquely determine a semipath from v to w in the inclusion tree D_c as

$$p_m(= v), p_{m-1}, \cdots, p_1, t, q_1, \cdots, q_{n-1}, q_n(= w) \quad (5)$$

where t is the top vertex (or the vertex of minimal depth). This means that the adjacency edge e originates from the rectangle of v , goes out across $p_{m-1}, \ldots, p_1$, passes t , goes in across $q_1, \ldots, q_{n-1}$ and terminates on w. *Down-arrow* convention is formulated by specifying an order among each pair (p_i, q_i) of vertices of same depth for any adjacency edge $(v, w) \in F$ as follows:

1) if $dep(v) > dep(w)$ (or $m > n$),
$clev(p_i) \leq clev(q_i), i = 1, \ldots, n-1,$ (6)
$clev(p_n) < clev(w);$ (7)
2) if $dep(v) \leq dep(w)$ (or $m \leq n$),
$clev(p_i) clev(q_i), i = 1, ..., m-1$ (8)
$clev(v) < clev(qm).$ (9)

For example, in Fig. 1, the semipath corresponding to adjacency edge (j, l) is j, e, b, f, l where b is the top vertex. Since $dep(j) = dep(l)$, we have $clev(e) \leq clev(f)$ and $clev(j) < clev(l)$ from (8) and (9) respectively.

A compound digraph D has a *compound level assignment* if and only if there exists a mapping $clev : V \to \Sigma^+$ satisfying (3), (4), and (6)–(9).

B. Hierarchization Algorithm

A given compound digraph can not always have a compound level assignment due to the existence of cycles and the definition of compound levels as described in Section II–B–1) Consequently, we develop an algorithm to attain a hierarchical layout of the compound digraph even when the compound level assignment is not possible. We first derive a new-typed compound digraph called *derived graph,* find *feedback adjacency edges* and then reverse the orientation of the edges.

1) Replacement of Adjacency Edges: Given a compound digraph $D = (V, E, F)$, in order to derive a new graph representing requirements (6)–(9) of the *Down-arrow* convention, every adjacency edge of D is replaced with two types of adjacency edges, $\rightarrow$ and $\Rightarrow$, which respectively express relations $<$ and $\leq$ in (6)–(9); i.e., for each adjacency edge $e = (v, w) \in F$, if $m > n$ in (5), e is replaced with $p_n \rightarrow w$ and $p_i \Rightarrow q_i, i = 1, \ldots, n-1$, and otherwise, $v \rightarrow q_m$ and $p_i \Rightarrow q_i, i = 1, \ldots, m-1$. (In the replacement, if edges between the same pair of vertices are duplicated, then reducing rules(&) such as $\rightarrow = \rightarrow \& \rightarrow, \rightarrow = \rightarrow \& \Rightarrow$ and $\Rightarrow = \Rightarrow \& \Rightarrow$ are applied to determine a type of a resulting edge.) We call the graph derived through this replacement the *derived graph* of D. Notice here that every adjacency edge in the derived graph links two vertices whose depth is identical. An adjacency edge e in the derived graph is called an *original edge* if $e \in F$, and a *derived edge* if $e \notin F$. For example, the derived graph of the compound digraph presented in Fig. 1(c) is shown in Fig. 2 where edges marked with asterisk($*$) are derived edges.

2) Assignment of Compound Levels to Vertices: The derived graph of D is used for assigning compound levels to all the vertices of V. We denote the derived graph by $DD = (V, E, FD, type)$ where FD is a set of adjacency edges derived from F and *type*: $FD \to \{\rightarrow, \Rightarrow\}$. If DD has cycles, we need procedure ResolveCycle below for resolving the cycles to obtain a cycle-free graph $DF = (VF, E, FF, type)$

```
procedure ResolveCycle($DD, DF$);
begin
  find all the strongly connected components
  [22] $C = C_i$ in $DD_a = (V, FD)$; {$DD_a$ is the
  adjacency digraph of $DD$} if ($c \neq \emptyset$ ) then
  for (each $C_i$ ) do
    if (all the edges of $C_i$ consist of type $\Rightarrow$)
      then
        replace $C_i$ into a single proxy vertex
      else
        eliminate appropriate edges called feed-
        back edges(*) according to the following
        rules: (1) edges of type $\Rightarrow$ are eliminated
        before edges of type $\rightarrow$ and (2) derived
        edges are eliminated before original ones
        among edges of type$\rightarrow$;
  denote new sets of vertices and adjacency edges
  as $VF$ and $FF$ respectively
end;[1]
```

For example, in Fig. 2, there are two strongly connected components C_1 and C_2 with vertex sets $\{b, c\}$ and $\{g, i\}$ respectively. In C_1 edge (c, b) is eliminated according to the rule (2) and in C_2 edge (i, g) is eliminated by chance.

In DF we can put $clev(root) = (1)$ without losing generality. Then, we can assign compound levels to all the vertices of VF by assigning them to children of vertices whose levels are already assigned from root to leaves recursively in DF. A *digraph* for a subset $W \subset V$ in DF is denoted by $S = (W, H, type)$ where $H = \{(v, w) \in FF | v, w \in W\}$. Then, invoking procedure CompLevAssign(DF, $\{root\}$, $clev$) can assign $clev(v)$ for each $v \in VF$ where function *tail* returns the last component of a sequence of integers.

```
procedure CompLevAssign($DF, W, clev$);
begin
  M:= LevAssign($DF, W, clev$);
  for i:=1 to M do
    begin
      Z:= $Ch(\{z \in W | \mathrm{tail}(clev(z)) = i\})$;
      if ($Z \neq \emptyset$)
        then CompLevAssign($DF, Z, clev$)
    end
end

function LevAssign ($DF, Z, clev$): integer;
begin
  constitute $S = (W, H, type)$; partition $W$ into its
  blocks called levels [11,p. 407] $L_i, i = 1, \ldots, n$
  in $S$;
  for (each $y \in L_1$) do
    begin
      $lev(y)$:=1;
      $clev(y)$:= $append(clev(Pa(y)), lev(y))$
    end;
  for $i$:=2 to $n$ do for (each vertex $y \in L_i$ ) do
    begin
      $lev(y)$:= max$\{l_v | (v, y) \in H, v \in L_1 \cup \ldots \cup L_{i-1}\}$ where
      $l_v = lev(v) + 1$ if $type(v, y)$ =$\rightarrow$or $l_v = lev(v)$
      if$type(v, y)$ =$\Rightarrow$; $clev(y)$:= $append(clev(Pa(y)). lev(y))$
    end;
  LevAssign:= max $\{lev(y) | y \in W\}$
end;
```

[1] Since the problem of finding the minimum feedback edge set is NP-complete [20], [23], a heuristic method is used as follows. An edge is eliminated in each component according to rules 1) and 2), and then strongly connected components are found. If there is no component, this procedure is terminated, otherwise, repeated.

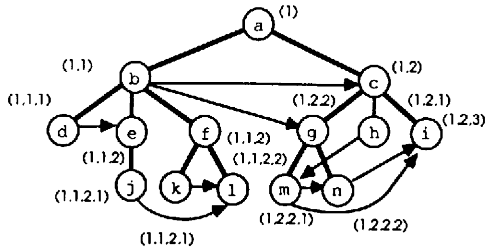


Fig. 3. An *assigned compound digraph* obtained from the compound digraph of Fig. 1(c).

3) Reversing the Orientation of Adjacency Edges: If VF includes proxy vertices, compound levels of all the vertices of the component in DD corresponding to each proxy vertex are set identical. Each adjacency edge (v, w) of the compound digraph $D = (V, E, F)$ is checked whether $clev(v) < clev(w)$ holds or not. If it does not hold, the orientation of the edge is reversed. As the result, we have an *assigned compound digraph* $DA = (V, E, FA, clev)$. In the final drawing step the original orientation of the edge is revived and drawn as an upward arrow.

For example, in the compound digraph presented in Fig. 1(c), the orientation of adjacency edges (g, b) and (i, n) are reversed into (b, g) and (n, i) , because $clev(g) = (1, 2, 2) > clev(b) = (1, 1)$ and $clev(i) = (1, 2, 3) > clev(n) = (1, 2, 2, 2)$ respectively. Fig. 3 shows an *assigned compound digraph* obtained.

IV. Normalization (Step II)

In an *assigned compound digraph* $DA = (V, E, FA, clev)$, an adjacency edge $(v, w) \in FA$ is said to be *proper* if and only if $clev(Pa(v)) = clev(Pa(w))$ and $\mathrm{tail}(clev(v)) = \mathrm{tail}(clev(w)) - 1$. An *assigned compound digraph* DA is called a *proper compound digraph* if and only if every adjacency edge in DA is proper.

Our normalization algorithm takes an *assigned compound digraph* DA and converts it into a proper compound digraph $DP = (VP, EP, FP, clev)$. In the algorithm, every nonproper adjacency edge is replaced with appropriate dummy vertices, dummy inclusion edges and dummy proper adjacency edges. Let $(v, w) \in FA$ be a nonproper adjacency edge, then in general we can put $clev(v) = (\alpha, s_1, \ldots, s_m)$ and $clev(w) = (\alpha, t_1, \ldots, t_n)$ where α is a subsequence common in both sequences, s_i's and t_j's are positive integers, and $s_1 < t_1$. Notice that since the conversion is limited within such a vertex of which compound level is α, we can neglect α. Then, every nonproper edge (v, w) in DA is replaced with the following "linear" compound digraph:

$$\mathbf{V} \rightarrow (s_1 + 1) \rightarrow \cdots \rightarrow (t_1 - 1) \rightarrow \mathbf{W} \qquad (10)$$

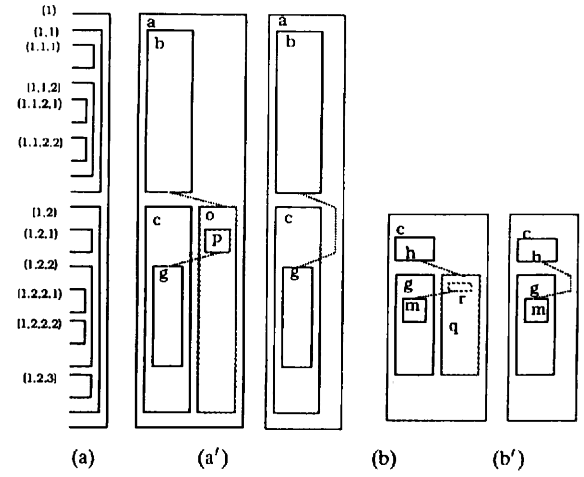


Fig. 4. Conversion of nonproper adjacency edges.

and the equations at the bottom of the page. where (β) is a dummy vertex whose compound levels is (β); $\rightarrow, \downarrow$: dummy adjacency edges; $\subset, \supset$: dummy inclusion edges; M_k: the maximum level or $\max\{\text{tail}(clev(v))|clev(Pa(v)) = (s_1, \dots, s_{m-k}), v \in V\}$. If $s_m = M_1$ (or $t_n = 1$), a *virtual* bottom(top) level and a virtual dummy vertex on the level are added (see vertex r in Fig. 4 (b)).

In the *assigned compound digraph* presented in Fig. 3, adjacency edges $(b, g), (h, m), (m, i)$ and (n, i) are not proper. For example, (b, g) and (h, m) is replaced with the linear compound digraphs shown Fig. 4 (a) and (b). The linear compound digraphs are drawn actually as shown in Fig. 4 (a′) and (b′) respectively.

V. Vertex Ordering (Step III)

A. Preliminaries

Let $D = (V, E, F, clev)$ be a proper compound digraph and $W = V - \{\text{leaves}\} = \{v_1, \dots, v_N\}$. And suppose that children $Ch(v)$ for each $v \in W$ can be partitioned into $n(v)$ subsets according to their compound levels, i.e.,

$$Ch(v) = V_1(v) \cup \cdots \cup V_i(v) \cup \cdots \cup V_{n(v)}(v) \qquad (11)$$

where $V_i(v) = \{u \in Ch(v)|\text{tail}(clev(u)) = i\}$ and$V_i(v)$ is called the ith *level*. Then, we define an *ordered compound digraph* by

$$D(\sigma) = (V, E, F, \text{clev}, \sigma) \qquad (12)$$

where $\sigma = (\sigma(v_1), \dots, \sigma(v_N)), \sigma(v_j) = (\sigma_1(v_j), \dots, \sigma_{n(v_j)}(v_j))$ and $\sigma_i(v_j)$ is an order among all the vertices of each $V_i(v_j)$. Fig. 5(a) shows a map of an *ordered compound digraph* where $\sigma = (((u_1, v, w_1, w_2)), ((u_2), (u_3)), ((v_1, v_2), (v_3, v_4, v_5)), ((w_3), (w_5), (w_7)), ((w_4), (w_6), (w_8)), ((u_4)), ((v_6), (v_9)), ((v_{10})), ((v_7, v_8), (v_{11})))$.

In an *ordered compound digraph* $D(\sigma)$, let $A(v)$ be a set of vertices (except v) whose compound level is equal to $clev(v)$. Then, we can partition $A(v)$ into $A^L(v)$ and $A^R(v)$ where every vertex in $A^L(v)$ or $A^R(v)$ is leftward or rightward to v in σ respectively. For example, $A^L(v) = \{u_1\}$ and $A^R(v) = \{w_1, w_2\}$. A local hierarchy for vertex $v \in V - \{\text{leaves}\}$ is defined by

$$H(\sigma(v)) = (Ch(v), F(v), n(v), \sigma(v), \lambda, \rho, \omega) \qquad (13)$$

where

1) $Ch(v)$ is partitioned according to (11);
2) $F(v)$ consists of the following two sets:
   a) F^d is a set of edges between different levels or $F^d = \{(u, w) \in F|u, w \in Ch(v)\}$,
   b) F^s is a set of edges between vertices of same level or $F^s = \{(u, w)|(x, y) \in F, x \in De(u) - \{u\}, y \in De(w) - \{w\}, u, w \in Ch(v)\}$;
3) $\lambda(w), \rho(w)$: the numbers of edges between descendants of $Ch(v)$ and descendants of $A^L(w)$ or $A^R(w)$ for $w \in Ch(v)$ respectively;
4) $\omega(u, w)$: the multiple degree of edge $(u, w) \in F^s$.

Fig. 5(b) shows the local hierarchy for vertex v in the *ordered compound digraph* shown in Fig. 5(a) where $F^d = \{(v_1, v_3), (v_1, v_4), (v_2, v_1), (v_2, v_5)\}$; $F^s = \{(v_3, v_5), (v_5, v_4)\}$; $\omega((v_3, v_5)) = 1$ and $\omega((v_5, v_4)) = 2$.

$$\mathsf{V} = \begin{cases} v & \text{if } m = 1 \\ \begin{array}{c} v \\ \downarrow \\ \{\{\cdots\{(s_1, \cdots, s_m + 1) \rightarrow \cdots \rightarrow (s_1, \cdots, M_1)\} \subset (s_1, \cdots, s_{m-1}) \rightarrow (s_1, \cdots, s_{m-1} + 1) \rightarrow \cdots \rightarrow (s_1, \cdots, M_2)\} \subset \cdots\} \\ \subset (s_1) \end{array} & \text{if } m > 1 \end{cases}$$

$$\mathsf{W} = \begin{cases} w & \text{if } n = 1 \\ \begin{array}{c} (t_1) \supset (t_1, 1) \rightarrow (t_1, 2) \rightarrow \cdots \rightarrow (t_1, t_2) \supset \cdots \supset \{(t_1, \cdots, t_{n-1}, 1) \rightarrow \cdots \rightarrow (t_1, \cdots, t_{n-1}, t_n - 1)\} \cdots\} \\ \downarrow \\ w \end{array} & \text{if } n > 1 \end{cases}$$

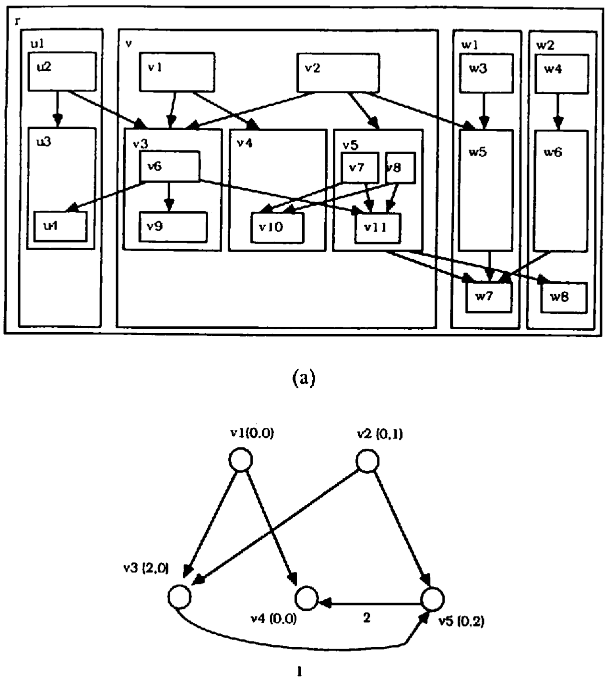


Fig. 5. (a) An *ordered compound digraph* and (b) a local hierarchy for vertex v where $v_i(m,n)$ means $\lambda(v_i) = m$ and $\rho(v_i) = n$.

B. Vertex Ordering Algorithm

We consider reordering of vertices in $D(\sigma)$ to attain the *Closeness* , *Line-crossing* and *Line-rect-crossing* rules. Let $S = (S(v_1),\ldots,S(v_N)), S(v_j) = S_1(v_j) \times \ldots \times S_{n(v_j)}(v_j)$ and $S_i(v_j)$ be a set of all possible orders of $\sigma_i(v_j)$. Then, the problem of attaining the rules is stated as

$$\mathbb{P}: \ \min\{c_1C(D(\sigma)) + c_2K(D(\sigma)) + c_3Q(D(\sigma)) \mid \sigma \in S\} \tag{14}$$

where

1) $C(D(\sigma)), K(D(\sigma)), Q(D(\sigma))$: quantitative measures for *Closeness*, *Line-crossing* and *Line-rect-crossing* respectively;
2) c_1, c_2, c_3: weighting constants satisfying $c_1+c_2+c_3 = 1$.

We decompose $\mathbb{P}$ into problems for local hierarchies of $D(\sigma)$ such as

$$\mathbb{P}(v_j): \ \min\{c_1C(H(\sigma(v_j))) + c_2K(H(\sigma(v_j))) + c_3Q(H(\sigma(v_j))) \mid \sigma(v_j) \in S(v_j)\}, \quad j = 1,\cdots,N. \tag{15}$$

In order to solve $\mathbb{P}$, problems $\mathbb{P}(v_j)$'s are solved one by one from root to lieves recursively by invoking VOrderGlobal($D(\sigma)$, *root*), where the maximum depth of $D(\sigma)$ is supposed to be more than one. Procedure VOrderLocal is described in Section V-B-2).

```
procedure VOrderGlobal ($D(\sigma), v$);
begin
if $Ch(v) \neq \emptyset$) then
  begin
    VOrderLocal($H(\sigma(v))$);
    {Problem $\mathbb{P}(v)$ is solved here.}
    for (each $w \in Ch(v)$) do
      VOrderGlobal($D(\sigma), w$)
  end
end;
```

Since problem $\mathbb{P}(v)$ is based upon a problem for a two-level local hierarchies, we consider a procedure for a two-level local hierarchy first, and then this procedure is extended to cases for n-level local hierarchies.

1) Vertex Ordering in a Two-Level Local Hierarchy: Let $H(\sigma) = (V, F, 2, \sigma, \lambda, \rho, \omega)$ be a two-level local hierarchy where $V = V_1 \cup V_2$, $F = F^d \cup F^s$ and $\sigma = (\sigma_1, \sigma_2)$. When we put $\sigma_1 = (u_1,\ldots,u_k,\ldots,u_p)$ and $\sigma_2 = (w_1,\ldots,w_\alpha,\ldots,w_q)$ where $p = |V_1|$ and $q = |V_2|$, then the problem to reorder vertices in $H(\sigma)$ to attain the *Closeness*, *Line-crossing* and *Line-rect-crossing* rules is stated as

$$\mathbb{Q}: \ \min\{c_1C(H(\sigma)) + c_2K(H(\sigma)) + c_3Q(H(\sigma)) \mid \sigma_1 \in S_1, \sigma_2 \in S_2\} \tag{16}$$

where quantitative measures for the drawing rules are expressed by

1) *Closeness* is the sum of differences between positions of vertices with nonzero values of λ and ρ and both ends in each level; i.e.,

$$C(H(\sigma)) = \sum_{1\le k\le p} (k\lambda(u_k) + (p-k+1)\rho(u_k)) + \sum_{1\le\alpha\le q} (\alpha\lambda(w_\alpha) + (q-\alpha+1)\rho(w_\alpha)). \tag{17}$$

2) *Line-crossing* is the number of crossings among edges of F^d; i.e.,

$$K(H(\sigma)) = \left| \{\{(u_k,w_\alpha),(u_l,w_\beta)\} \subset F^d \mid 1 \le k < l \le p, 1 \le \beta < \alpha \le q\} \right|. \tag{18}$$

3) *Line-rect-crossing* is the total sum of lengths to which each edge of F^s stretches; i.e.,

$$Q(H(\sigma)) = \sum_{(u_k,u_l)\in F^s} \omega((u_k,u_l))|k-l| + \sum_{(w_\alpha,w_\beta)\in F^s} \omega((w_\alpha,w_\beta))|\alpha-\beta|. \tag{19}$$

Before describing how to solve the problem $\mathbb{Q}$, we discuss theoretical and heuristic aspects of three elementary problems such as minimizations of $C(H(\sigma)), K(H(\sigma))$ and $Q(H(\sigma))$, and then show a heuristic method for solving $\mathbb{Q}$.

1) *Elementary Problems*:

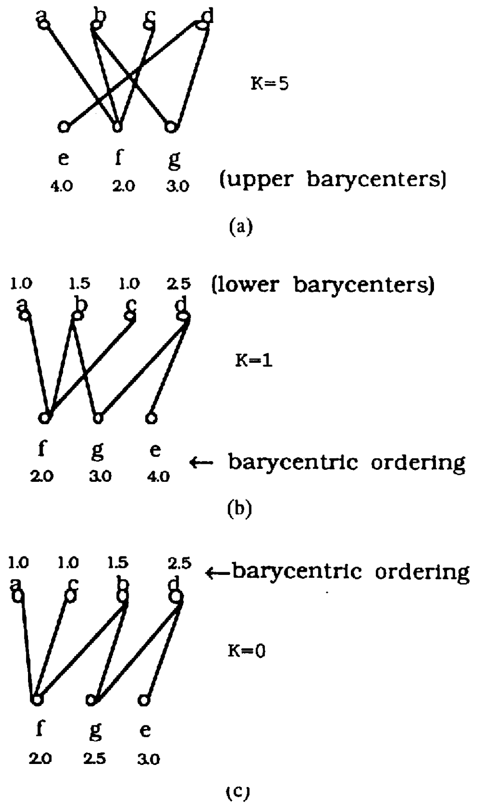
a) minimizing $C(H(\sigma))$: This optimization can be easily attained. For each vertex $v \in V$, let $t = \lambda(v) - \rho(v)$. Then if $t > 0$ or $t < 0$, the vertex is positioned at the left end or right end, respectively, in σ_1 and σ_2. If there exist more than one such vertices, then the larger $|t|$ is, the nearer to the ends their positions are set. This method is called *splitting method*.

b) minimizing $K(H(\sigma))$: This optimization problem is equivalent [9] to the minimum *feedback edge set* *problem* whose complexity is NP-complete [21]. Consequently, we developed a heuristic method called *two-level barycentric(BC) method* [9, p. 115]. This method repeats *barycentric ordering* of vertices in the first and second levels in turn.

In Fig. 6(a), the upper barycenter of vertex f, for example, is calculated as $2.0 = (1+2+3)/3$ because it connects to the first, second and third vertices (a, b, c) of the first level. Since the upper barycenters of vertices of the second level in a are not in an increasing order, the barycentric ordering is applied to them and we have b. Similarly, the barycentric ordering is applied to vertices of the first level in b and we have c. As the result, the number of crossings, K, is reduced from 5 to 0.

c) minimizing $Q(H(\sigma))$ For simplicity, the multiple degree ω of edges is ignored in (19). This problem concerns only with one-level hierarchies $H(\sigma_1)$ and $H(\sigma_2)$. The optimization on $H(\sigma_1)$ or $H(\sigma_2)$ is equivalent to the *linear arrangement problem* [20] whose complexity is NP-complete since for each edge in $H(\sigma_i)$ the number of crossings between the edge and rectangles is equal to the *length* [20] of the edge minus one. Therefore, a heuristic method called *insertion BC method* is developed. Fig. 7 explains the insertion BC method. One-level hierarchy $H(\tau), \tau = (g, h, i, j, k)$ is given in (a). In (b), vertices w, x, y and z are inserted and ordered according to their barycenters into (w, x, y, z). Then, the barycentric ordering is applied to the vertices of the second level and we have $\tau = (i, g, j, k, h)$ in (c); (d) shows an optimum solution. The performance of the insertion BC method has been evaluated by choosing various types of one-level hierarchies different in the number of vertices and the density of edges. For each type of hierarchy, the mean values of an initial value, the minimum solution, the worst solution and a heuristic solution of the total length of edges are obtained for each given density of edges, i.e., 0.2, 0.4, 0.6 or 0.8. A combinatorial (exhaustive) search is carried out for each type of hierarchies whose numbers of vertices are 4, 5 and 6, whereas samples of one thousand initial orders of one hundred hierarchies generated randomly are tested for hierarchies with 7 and 8 vertices. The mean values obtained for the case of six vertices are shown in Table I. We have found that the mean of heuristic solutions is less than 1.2 times of the mean of the minimum soluions for all the types of hierarchies we have tested.

2) *Heuristic Algorithm for Solving Problem* Q: Our heuristic algorithm is characterized by two features: 1) we adopt empirical priority $R1 > R2 > R3$ as presented in (1), and 2) the barycentric method is applied to solve simultaneously both crossing reduction problem and



a b c d
K=5
e f g
4.0 2.0 3.0 (upper barycenters)
(a)
1.0 1.5 1.0 2.5 (lower barycenters)
a b c d
K=1
f g e
2.0 3.0 4.0 ← barycentric ordering
(b)
1.0 1.0 1.5 2.5 ←barycentric ordering
a c b d
K=0
f g e
2.0 2.5 3.0
(c)

Fig. 6. An application of barycentric method to a two-level hierarchy.

TABLE I
RESULTS OF PERFORMANCE TESTS OF THE INSERTION BC METHOD

Density	Minumum	Worst	Initial	Heuristic
0.2	3.00	9.00	7.00	3.00
0.4	8.05	19.97	14.00	9.53
0.6	14.94	26.89	21.00	17.65
0.8	23.20	31.73	28.00	26.62

linear arrangement problem. In procedure VOrder2Lev, BOU(i) and BOL(i) mean the barycentric ordering according to upper and lower barycenters respectively in the ith level; IDU(i) and IDL(i) the insertion of dummy vertices in the upper and lower level respectively for edges among V_i; and I the limit of iterations. As a preprocess to attain rule $R1$, in $H(\sigma)$, vertices in each level are reordered according to the splitting method and the positions of vertices such that either $\lambda(v)$ or $\rho(v)$ is nonzero are fixed. Therefore, for simplicity, we ignore such vertices and edges incident to the vertices in VOrder2Lev.

```
procedure VOrder2Lev ($H(\sigma(v))$);
begin
  for i:= 1 to I do
    begin
      IDU(1); BOU(1); BOU(2); IDU(2);
      BOU(2); {down step}
      IDL(2); BOL(2); BOL(1); IDL(1);
      BOL(1) {up step}
  end
end;
```

Fig. 8 shows an application of vertex ordering to a two-level

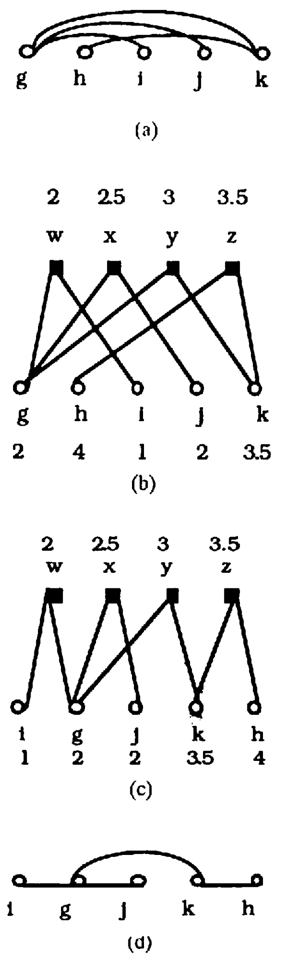


Fig. 7. An application of the insertion BC method.

local hierarchy where (a) and (b) show the preprocess.

2) Vertex Ordering in an n-Level Local Hierarchy: The algorithm for an n-level local hierarchy is an extension of the algorithm for a two-level local hierarchy. In procedure VOrderLocal$I$ means a limit of iterations. As a preprocess, in $H(\sigma_1, \ldots, \sigma_n)$ vertices in each level are reordered according to the splitting method and the positions of vertices such that either $\lambda(v)$ or $\rho(v)$ is nonzero are fixed. Therefore, for simplicity, we ignore such vertices and edges incident to the vertices in VOrderLocal.

```
procedure VOrderLocal($H(\sigma_1, \ldots, \sigma_n)$);
begin
  for $i$:=1 to I do
    begin
      IDU(1); BOU(1); {start of down step} for
      $j := 2$ to $n$ do
        begin
          BOU($j$); IDU($j$); BOU($j$)
        end;
      IDL($n$); BOL($n$); {start of up step}
      for $j := n - 1$ downto 1 do
        begin
          BOL($j$); IDL($j$); BOL($j$)
        end
    end
end;
```

VI. Metrical Layout (Step IV)

Here we consider how to determine a metrical layout of a given *ordered compound digraph* and present its heuristic algorithm. Our algorithm has been developed along the following basic ideas.

1) A metrical layout consists of *vertex positioning* (i.e., determining horizontal and vertical positions, widths and heights of rectangles), and *edge routing* (i.e., determining metrical layouts of adjacency edges). In the algorithm our effort is made mainly for the vertex positioning, since we can determine the edge routing readily from the vertex positioning as follows:
   a) for any proper adjacency edge in an original compound digraph, we can draw it as a straight line without bending points, and
   b) for any nonproper adjacency edge in the digraph, we can obtain the edge routing directly from positions of rectangles by setting widths of rectangles corresponding dummy or virtual vertices to be zero (see Fig. 4(a′) and (b′)).
2) The problem to determine a metrical layout of rectangles is separable into two independent problems: *horizontal problem* to calculate horizontal positions and widths of rectangles, and *vertical problem* to calculate vertical positions and heights of rectangles. The latter is an easy task since it has no relation to the drawing rules while the former is rather complicated since it relates to the rules. In this section, we consider only the horizontal problem.

A. Metrical Layout Algorithm

1) Preliminaries: When an *ordered compound digraph* $D(s) = (V, E, F, clev, s)$ is given, a *metrical local hierarchy* for vertex $v \in V - \{\text{leaves}\}$ is defined by

$$HL(v) = (Ch(v), F^d, n(v), \sigma(v)) \tag{20}$$

where notations are shown in (13). In a metrical local hierarchy $HL(v)$, *upper metrical barycenter* $Bu(w)$ and *lower metrical barycenter* $Bl(w)$ for vertex $w \in Ch(v)$ are defined by

$$Bu(w) = \begin{cases} \left\{ \sum_{(u,w) \in Ft(w)} \chi(u) \right\} / |Ft(w)| & \text{if } |Ft(w)| > 0 \\ \chi(w) & \text{otherwise} \end{cases} \tag{21}$$

$$Bu(w) = \begin{cases} \left\{ \sum_{(w,u) \in Fo(w)} \chi(u) \right\} / |Fo(w)| & \text{if } |Fo(w)| > 0 \\ \chi(w) & \text{otherwise} \end{cases} \tag{22}$$

where $Ft(w) \subset F^d$ is a set of edges terminating on w and $Fo(w) \subset F^d$ a set of edges originating from w. $|Ft(w)|$and$|Fo(w)|$ are called *upper* and *lower connectivities* respectively.

2) A Heuristic Metrical Layout Algorithm: To obtain a readable map of $D(\sigma)$, procedure MetricalLayout have been developed, where the maximum depth of $D(\sigma)$ is supposed to be more than one. The procedure receives $D(\sigma)$ and widths

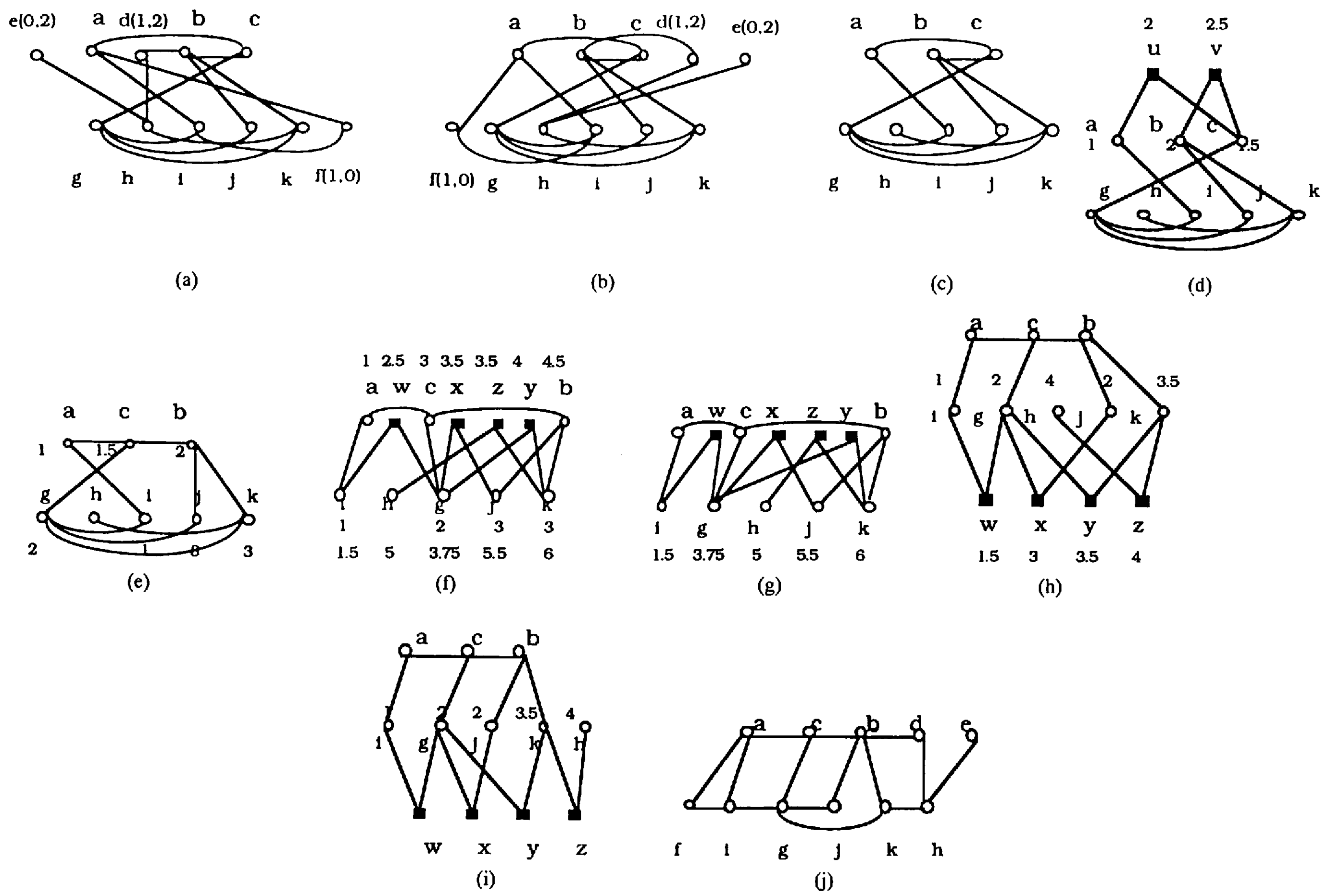

e(0,2) a d(1,2) b c
g h i j k f(1,0)
(a)
a b c d(1,2) e(0,2)
f(1,0) g h i j k
(b)
a b c
g h i j k
(c)
2 2.5 u v a 1 b c 2 2.5 g h i j k
(d)
a c b 1 1.5 2 g h i j k 2 3
(e)
1 2.5 3 3.5 3.5 4 4.5 a w c x z y b
1 2 3 3 1.5 5 3.75 5.5 6
(f)
a w c x z y b
i g h j k 1.5 3.75 5 5.5 6
(g)
a c b 1 2 4 2 3.5 i g h j k w x y z 1.5 3 3.5 4
(h)
a c b 1 2 3.5 4 i g j k h w x y z
(i)
a c b d e f i g j k h
(j)

Fig. 8. An application of vertex ordering to a two-level local hierarchy.

of leaves ($\delta(v)$ for each v $\in$\{leaves\}; the width should be a positive even integer), and calculates global horizontal positions ($X(v)$ for each $v \in V$) and widths of nonleaf vertices ($\delta(v)$ for each $v \in V-$\{leaves\} where d_1 and d_2 means metrical parameters. (See Fig. 9.) LayoutLocal calculates local positions and widths of all the vertices of $D(\sigma)$ and LayoutGlobal their global positions. PRMethod is explained in Section XI-B.

```
procedure MetricalLayout ($D(b), X, \delta$);
begin
  LayoutLocal($D(\sigma), root, \chi, \delta$ );
  $X(root)$:= $\delta(root)/2$; LayoutGlobal($D(\sigma), root, \chi, X, \delta$)
end;

procedure LayoutLocal ($D(\sigma), v, \chi, \delta$);
begin
  if ($Ch(v)$ includes nonleaf vertices) then
    for (each nonleaf vertex $w \in Ch(v)$) do
      LayoutLocal ($D(\sigma), w, X, \delta$ );
  PRMethod($HL(v), \chi, \delta$)
end;
procedure LayoutGlobal ($D(\sigma), v, \chi, X, \delta$ );
begin
    if ($Ch(v) \neq \emptyset$) then for (each $w \in Ch(v)$) do
      begin
      $\chi(w)$:= $(C(v) - \delta(v)/2) + \chi(w)$;
      LayoutGlobal ($D(\sigma), w, \chi, X, \delta$)
    end
end;
```

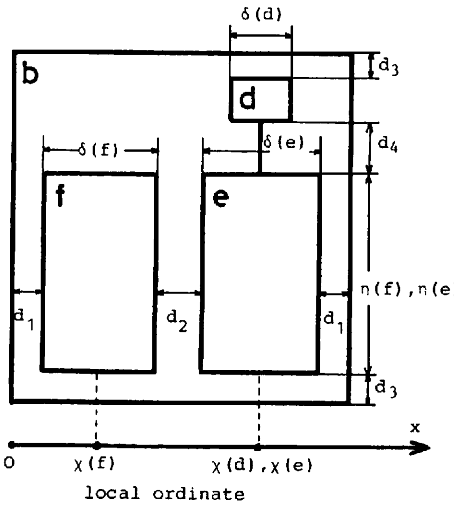


Fig. 9. A map of metrical local hierarchy $HL(b)$, which is a part of the map presented in Fig. 1(d).

B. Metrical Layout Algorithm in a Metrical Local Hierarchy

In order to solve the problem to determine horizontal positions and widths of rectangles corresponding to all the vertices, we solve problems formulated on metrical local hierarchies one by one recursively with procedure LayoutLocal. We have developed two types of methods to solve the problems; *quadratic programming* (QP) *method* [8] and *priority layout*

(PR) *method.* It is time-consuming to solve a QP method when its size is large. Consequently, we have developed a heuristic method (PR method) to reduce the computing cost. Here we describe only the PR method.

The fundamental idea for this method is similar to that for procedure VOrderLocal, i.e., "sequential" application of level operations called improvements of horizontal positions. In the case of VOrderLocal, barycenters of the vertices are used for reordering vertices, while in this method the improvement of positions is carried out according to the metrical barycenters defined by (21) and (22) and "priority numbers" given to the vertices. This method is an extension of the early method [6, p. 121], [22].

Let $\sigma_i(v) = (w_1^i, \cdots, w_k^i, \cdots, w_{|V_i(v)|}^i)$, $i = 1, \cdots, n(v)$ and we denote $x_k^i = \chi(w_k^i)$, $\delta_k^i = \delta(w_k^i)$ and $n = n(v)$ for simplicity. Procedure PRMethod receives $HL(v)$ and calculates local coordinates and widths of the vertices of $HL(v)$ (i.e., $\chi(w)$ and $\delta(w)$ for $w \in Ch(v)$). The procedure is outlined in the following.

1) *Initialization* : Initial values of horizontal positions of rectangles in each level are given by

$$x_k^i = \delta_1^i/2 + (d_2 + \delta_2^i) + \cdots + (d_2 + \delta_{k-1}^i) + (d_2 + \delta_k^i/2).$$

2) *Order of improved levels*: Positions of rectangles in each level are improved in the order of levels $2, 3, \ldots, n, n-1, \ldots, 2, 1, t, t+1, \ldots, n$ where t is a given integer $(2 \le t \le n-1)$. The improvements of the positions of rectangles in levels $2, \ldots, n$ and $t, \ldots, n$ are called DOWN procedures, while those for levels $n-1, \ldots, 1$ are called UP procedures.
3) *Priority*: Positions of rectangles in each level are determined one by one according to their priority numbers. The highest priority number, an integer more than the maximum of connectivities of all the vertices, is given to dummy vertices to improve *Line-straightness*. Priority numbers given to the other vertices are the connectivities of the vertices to improve *Balancing* through the operations described in 4). Details of a sophisticated method on how to assign the priority numbers are shown in [3, Part II, pp. 93–105].
4) *Improvement of Positions*: The principle to improve the position of a rectangle is to minimize the difference between the present position of the rectangle and the upper (or lower) metrical barycenter given by (21) (or (22)), of the corresponding vertex in DOWN (or UP) procedure under the following conditions.
   a) The ordinate of the rectangle should be integer and rectangles can not overlap with the other rectangles in the same level.
   b) The order of vertices of each level should be preserved to attain *Less-crossings*.
   c) Positions of rectangles whose priorities are less than the priority of the improved vertex can be displaced, where the distance displaced should be as small as possible to attain *Closeness*.

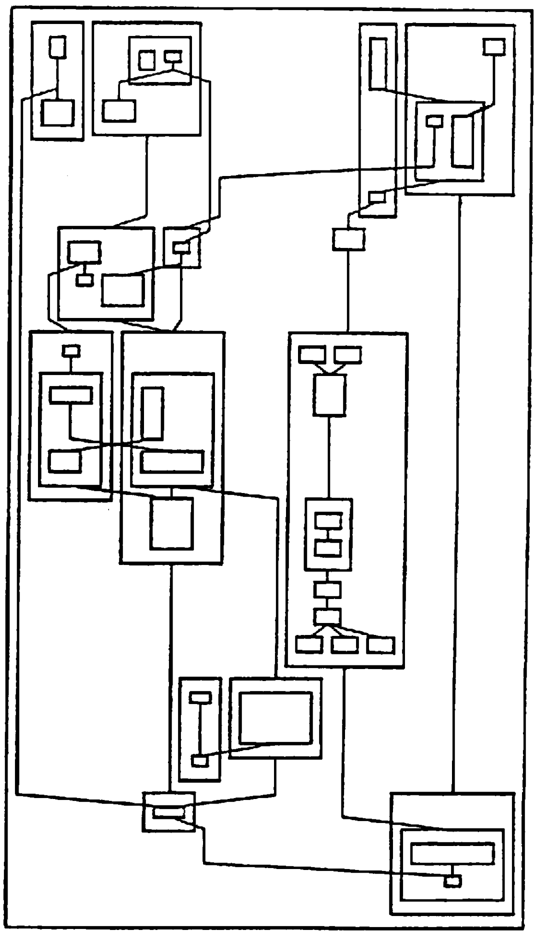


Fig. 10. A map of a compound digraph of 57 vertices, 39 adjacency edges and 56 inclusion edges.

5) After the UP/DOWN iterations have been terminated, positions $\chi(w)$'s for each $w \in Ch(v)$ are rescaled by $\chi(w) := \chi(w) - (\chi_0 - d_1)$ where $\chi_0 = \min\{\chi(u) - \delta(u)/2 | u \in Ch(v)\}$ and width $\delta(v)$ for v is obtained by $\delta(v) := \max\{\chi(w) + \delta(w)/2 | w \in Ch(v)\} - \min\{\chi(w) - \delta(w)/2 | w \in Ch(v)\} + 2d_1$.

Details of the algorithm are described in [8].

VII. Applications

We have already developed an algorithm for drawing digraphs in a hierarchical form, implemented it in a drawing tool called SKETCH and applied it to many different problems in a variety of fields, which is summarized in [25]. In Sections III–VI, we have described an algorithm to draw a more general class of graphs; i.e., compound digraphs. The algorithm has been implemented in SKETCH-II, written in *C*, on a Sun workstation. The tool can provide automatic drawing facilities including "straight line/curved line" options in drawing vertices and adjacency edges. In order to demonstrate the effectiveness of the heuristic drawing algorithm for compound digraphs, we show its several applications.

Fig. 10 shows a map of a rather large compound digraph of 57 vertices, 39 adjacency edges and 56 inclusion edges

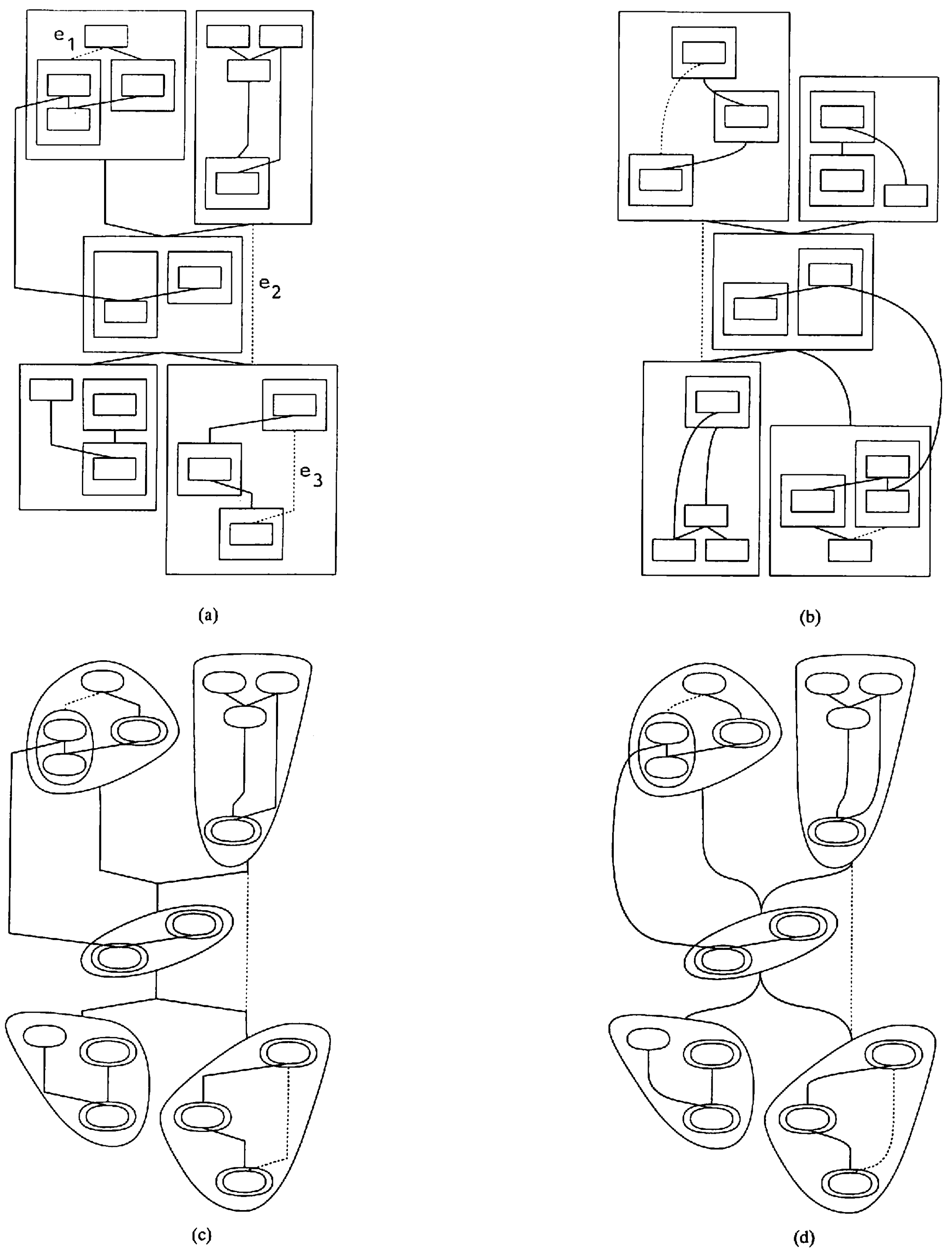


Fig. 11. Variations of a map of a compound digraph. (a) Option0; (b) Option1; (c) Option2. (d) Option3. (Dotted lines mean feadback edges.)

without cycles. In this application, the heights and widths of rectangles corresponding to leaves take various sizes that are given according to metrical drawing convention *Leaf-size*. Although the outermost rectangle corresponding to the root vertex is drawn in the figure, we usually do not draw it.

A map of a compound digraph of 32 vertices, 20 adjacency edges and 31 inclusion edges with cycles is shown in Fig. 11(a), which is drawn according to the drawing conventions described in Section II (Option0: rectangles for vertices and bending lines for adjacency edges). In the map, downward edges are drawn as solid lines and upward edges (i.e., reversed edges) broken lines. Of the upward edges, the reversion of the orientation of edge e_1 is carried out due to our definition of hierarchization and edges e_2 and e_3 due to the existence of cycles. Fig. 11 (b)–(d) show variations of the map by options: (b) shows Option1 (vertices: rectangles; adjacency edges: curves); (c) shows Option2 (vertices: closed curves; adjacency edges: bending lines); (d) shows Option3 (vertices: closed curves; adjacency edges: curves).

In producing curves, a simple interpolation is used; details

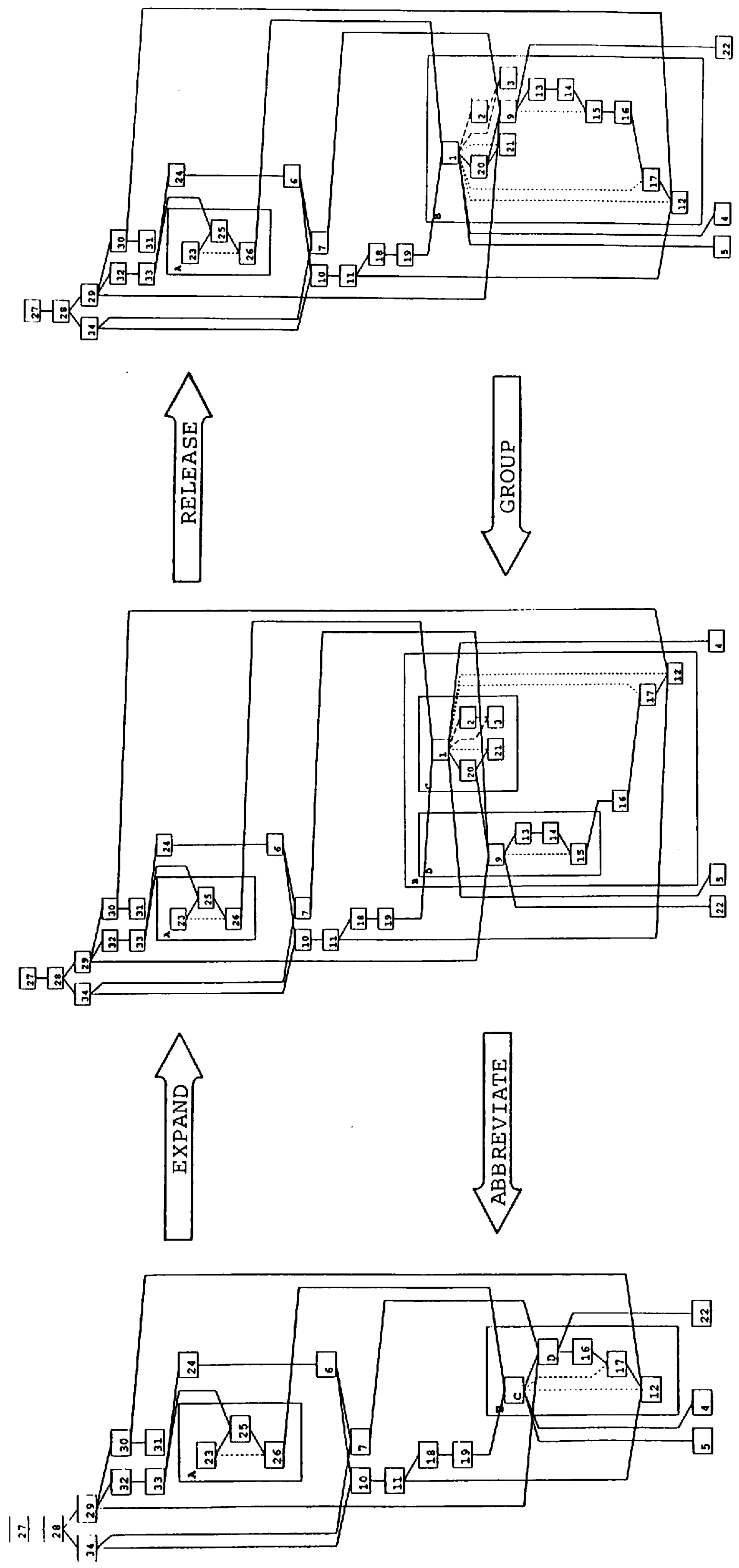


Fig. 12. Maps and command primitives (Option0). Dotted lines mean feadback edges and dashed lines bidirectional edges.

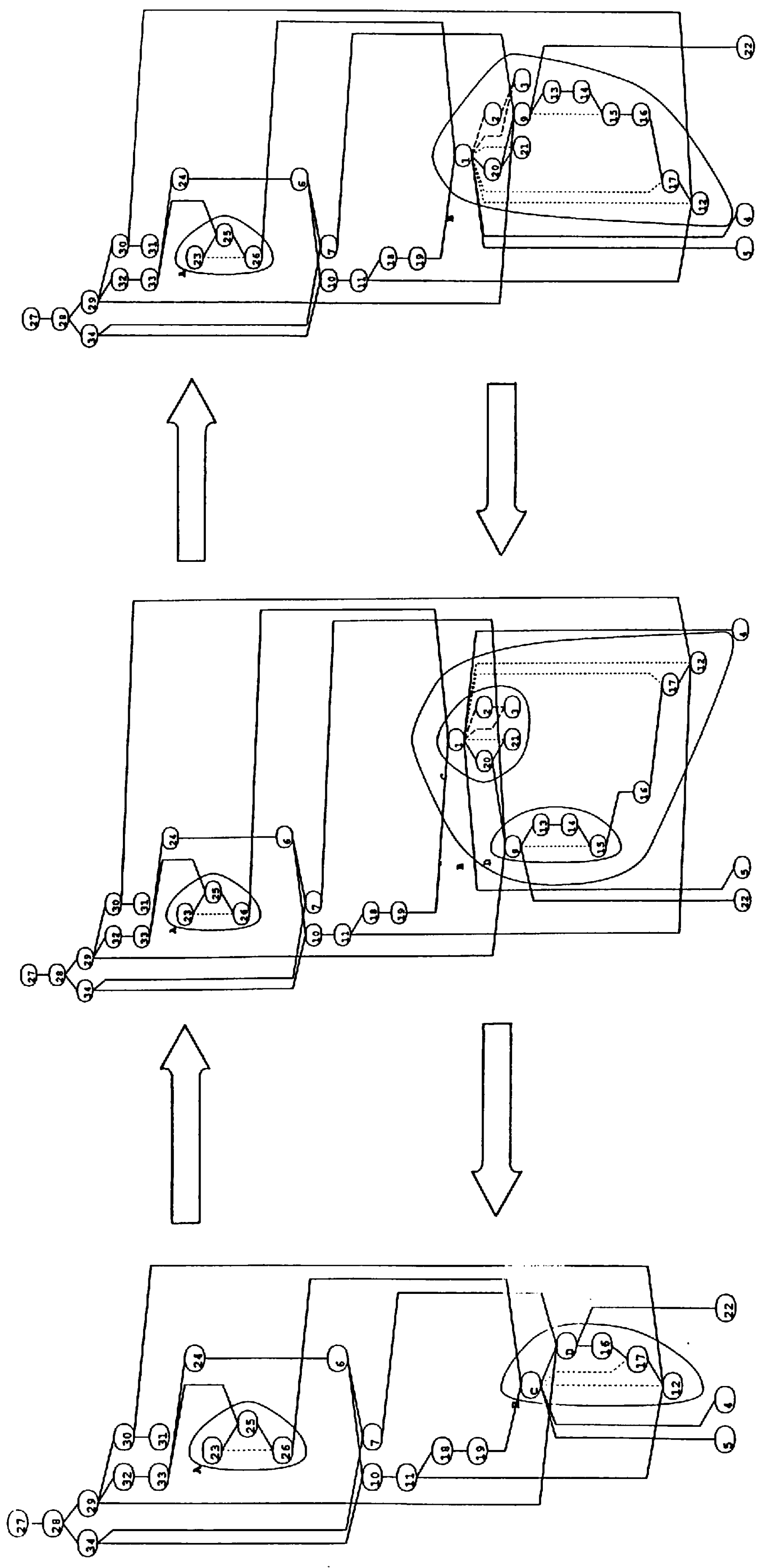


Fig. 13. Maps and command primitives (Option2). Dotted lines mean feadback edges and dashed lines bidirectional edges.

TABLE II
FEATURES OF MAPS AND COMPUTATION TIMES

	Fig. 10	Fig. 11(a)	Fig. 12(a)	Fig. 12(b)	Fig. 12(c)
I. Features of Maps					
a) The Number of Vertices	75	40	77	91	96
b) The Number of Feedback Edges	0	3	3	8	8
c) The Number of Nonproper Edges	12	8	15	19	16
d) $k = \sum\sum \|V_i(v)\|^2$	255	69	394	427	471
II. Computation Times (sec)					
1) Hierarchization	0.45	0.34	1.20	2.29	2.82
(finding feedback edges)	(0.00)	(0.25)	(0.47)	(1.23)	(1.72)
2) Normalization	0.02	0.01	0.02	0.03	0.03
3) Vertex-Ordering	0.83	0.35	1.03	1.33	1.40
4) Metrical Layout	0.27	0.13	0.32	0.39	0.40
Total	1.57	0.83	2.57	4.04	4.65

of how to draw curves are seen in [7]. By observing the variations of the map, we might say the followings.

1) In the cases of (b) and (c), mixing of straight and curved lines is effective to distinguish vertices and edges.
2) In the cases of (c) and (d), the shapes of closed curves expressing vertices reflect those of subgraphs included by the closed curves. This might well affect the readability of maps.
3) Maps (a) and (d) cause very different psychology; probably many people feel that the former is neat and rigid while the latter is friendly and flexible. It might be adequate that the former is utilized for representing formal, static or final notions while the latter informal, dynamic or temporary ones. We can select a more appropriate type of map depending on applications.

Our extension of formalism into a compound digraph can provide us more prominent and powerful facilities for representing and manipulating structural information in SKETCH-II than SKETCH. One of the most remarkable facilities, for example, is a "grouping" of vertices or "labeling" of a subgraph that ordinarily means generalization, abstraction, aggregation or integration of concepts assigned to the vertices or elements of the subgraph. Another is the facility for the refinement of concepts assigned to vertices, which can be realized by "expanding" the vertices into more refined structures.

The maps in Fig. 12 are simplified structures of C syntax [26, pp. 214–219], where

1) vertices labeled by numbers mean nonterminals in C syntax;
2) vertices labeled by alphabets mean cycles (or recursive parts in the syntax);
3) an adjacency edge $27 \rightarrow 28$, for example, means that nonterminal 27 (this means "program") is defined using nonterminal 28 (this means "external-definition").

These structures are chosen merely to explain the possibility of a set of command primitives for reorganizing structures within the extent of our graph formalism. Meanings of primitives in Fig. 12 are as follows.

1) *Expand:* Construct a more refined structure within a vertex (Vertices C and D are expanded in map (a), and map (b) is obtained.);
2) *Abbreviate:* Eliminate all the structures that a vertex includes (Subgraphs included by vertices C and D are eliminated in map (b), and map (a.1) is obtained.);
3) *Release:* Eliminate a vertex and release the structure that the vertex includes (Vertices C and D are eliminated and the structures they include are released in map (b), and map (c) is obtained.) ;
4) *Group:* Create a vertex including the structure that is constituted by some other vertices (Vertices 1, 2, 3, 20, and 21 are grouped into a subgraph included by vertex C and vertices 9, 13, 14, and 15 by vertex B in map (c) and map(b) is obtained.).

The maps in Fig. 12 are drawn with Option0 whereas the maps in Fig. 13 are drawn with Option2. In this case it seems that maps with Options2 are more readable than those with Option0 since we can distinguish vertices and edges more easily in the former than the latter.

Features of maps presented in this section and CPU time (s) needed for their computations on a Sun3/260 workstation are shown in Table II, where PR method is used for metrical layout. In Table II, (a) means the number of both original and dummy vertices, and (d) an index to evaluate computation times for vertex-ordering and metrical layout (see Fig. 14), which is defined by

$$k = \sum_{v \in W} \sum_{i \in \{1,2,\cdots,n(v)\}} |V_i(v)|^2 \tag{23}$$

where notations are defined in (13). The times for hierarchization include those for finding feedback edges and therefore they depend upon the number of feedback edges. If there is no cycle, the computation time is rather small (see the case of Fig. 10). The times for normalization are almost negligible. The times for both vertex ordering and metrical layout are nearly linear to the index k as seen in Fig. 14. The total times for all the cases show that the algorithm achieves satisfactory runtime performance.

VIII. CONCLUSION

We have introduced a compound digraph as a diagramming object by extending the class of digraphs, where both *Inclusive* and adjacent relations defined on a set of vertices are considered. Then, we have identified readability elements

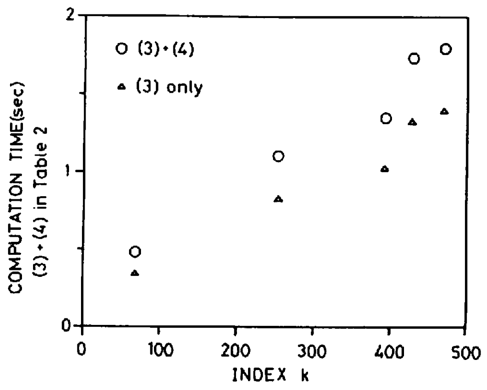


Fig. 14. Computations time for vertex-ordering (3) and metrical layout (4).

to specify drawing conventions, rules and priority among the rules for maps of the compound digraph, and have developed algorithms to generate "readable" maps in a hierarchical form. The whole algorithm consists of four steps; hierarchization, normalization, vertex ordering and metrical layout.

1) In the step of hierarchization, we have introduced the concept of compound levels and have developed an algorithm to assign the compound levels to a compound digraph even if the digraph includes compound cycles.
2) In the step of normalization, we have developed an algorithm to normalize a given *assigned compound digraph* into a proper one.
3) In the step of vertex ordering, we have given a heuristic algorithm to attain drawing rules *Closeness*, *Line-crossing* and *Line-rect-crossing* in a local hierarchy, which is used as a subalgorithm in an algorithm to obtain the improved orders of all the vertices in a given proper *assigned compound digraph*.
4) In the step of metrical layout, we have developed theoretical and heuristic algorithms to attain drawing rules *Closeness*, *Line-straightness* and *Balancing* in a metrical local hierarchy. Either one of these algorithms is used as a subalgorithm in an algorithm to obtain the improved horizontal positions of all the vertices in a given *ordered compound digraph*.

Three applications have been shown to demonstrate the effectiveness of the algorithms developed, where in comparison with straight lines, utilization of curves has been investigated to improve the readability of maps. A possible set of primitives for progressively organizing structures within our graph formalism has also been shown. The computation time to calculate map layout for all the cases show the algorithm achieves satisfactory runtime performance.

Both the extension of formalism and automatic drawing capabilities will attain an effective integration of human thinking and machine production of maps. And in many of our intellectual activities, they will be used as visual systems and/or interfaces for databases, knowledge bases, expert systems, idea processors, design systems, decision support systems and so on. Consequently, it is desired to investigate how to use the automatic drawing capabilities in dynamic thinking processes under more general visual formalisms [18]. In this direction the following studies are envisaged for future research [4]:

1) studies of drawing algorithms for compound graphs with both directed and undirected adjacency edges,
2) analyses of dynamic readability elements for diagrammatical thinking processes and developments of algorithms attaining the elements, and
3) studies on how to display and edit large maps on limited spaces of bit-mapped displays.

Acknowledgment

The authors would like to acknowledge useful comments of Dr. Mitsuhiko Toda of their institute. They are also grateful to the referees for suggestions to improve the notation and presentation of the paper.

References

[1] P. Eades and R. Tamassia, "Algorithms for automatic graph drawing: An annotated bibliography," Dept. Comput. Sci., Brown Univ., Providence, RI, Tech. Rep. no. CS-89-09, 1989.
[2] R. Tamassia, G. Di Battista, and C. Batini, "Automatic graph drawing and readability of diagrams," *IEEE Trans. Syst., Man, Cybern.*, vol. 18, pp. 61–79, 1988.
[3] K. Sugiyama, "A study on supporting diagrammatical thinking process: Toward developing informatics for thinking enhancement," Int. Institute for Advanced Study of Social Inform. Sci., Fujitsu Ltd., Res. Summary Rep. no. 24 (Part I), pp. 1–64, no. 25 (Part II), pp. 1–115, 1988, (in Japanese).
[4] K. Sugiyama and K. Misue, " 'Good' graphic interfaces for 'good' idea organizers," in *Proc. IFIP TC13*, Third Int. Conf. Human–Computer Interaction, Cambridge, UK, Aug. 27–31, 1990, pp. 521–526.
[5] E. B. Messinger, "Automatic layout of large directed graphs," Dept. Comput. Science, Univ. Washington, Seattle, Tech. Rep. no. 87-07-08, pp. 1–198, 1988.
[6] K. Misue and K. Sugiyama, "Compound graphs as abstraction of card systems and their hierarchical drawing," *Inform. Processing Soc.*, Japan, Research Report 88-GC-32-2, 1988, (in Japanese).
[7] K. Misue and K. Sugiyama, "Freehand-like drawing of compound graphs: A fundamental technique for computer aided abduction," in *Proc. 37th Conf. Inf. Proc. Soc. Jpn.*, 1988, pp. 1304–1305, (in Japanese).
[8] K. Sugiyama and K. Misue, "Visualizing structural information: Hierarchical drawing of a compound digraph," Int. Institute for Advanced Study Social Inform. Sci., Fujitsu Ltd., Res. Rep. no. 86, pp. 1–49,1989.
[9] K. Sugiyama, S. Tagawa, and M. Toda, "Method for visual understanding of hierarchical system structures," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-11, no. 2, pp. 109–125, 1981.
[10] K. Sugiyama, "A cognitive approach for graph drawing," *Cybern. Syst.*, vol. 18, no. 6, pp. 447–488, 1987.
[11] J. N. Warfield, "Crossing theory and hierarchy mapping," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-7, no. 7, pp. 505–523, 1977.
[12] L. A. Rowe, M. Davis, E. Messinger, C. Meyer, C. Spirakis, and A. Tuan, "A browser for directed graphs," *Software—Practice and Experience*, vol. 17, no. 1, pp. 61–76, 1987.
[13] E. R. Gansner, S. C. North, and K. P. Vo, "A program that draws directed graphs," *Software-Practice and Experience*, vol. 18, no. 11, 1047–1062, 1988.
[14] C. Hartshorne and P. Weiss, Ed., collected papers of C. S. Peirce, vol. II. Cambridge, MA: Harvard Univ. Press, 1978.
[15] J. Kawakita, "KJ method," Chuokoron-sha, 1986 (in Japanese).
[16] N. V. Findler, Ed., *Associative Networks*. New York: Academic, 1979.
[17] J. F. Sowa, *Conceptual Structures*. New York: Addison-Wesley, 1984.
[18] D. Harel, "On visual formalisms," *Commun. ACM*, vol. 31, no. 5, pp. 514–530, 1988.
[19] F. Harary, *Graph Theory*. Reading, MA: Addison-Wesley, 1972.
[20] S. Even, *Graph Algorithms*. Rockville, MD: Computer Science Press, 1979.
[21] D. S. Johnson, "The NP-completeness column: An ongoing guide," *J. Algorithms*, vol. 3, pp. 89–99, 1982.
[22] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.
[23] A. Lempel and I. Cederbaum, "Minimum feedback arc and vertex sets of a directed graph," *IEEE Trans. Circuit Theory*, vol. CT-13, no. 4, pp. 339–403, 1966.

[24] K. Sugiyama, S. Tagawa, and M. Toda, "Effective representations of hierarchical structures," Int. Institute for Advanced Study of Social Inform. Sci., Fujitsu Ltd., Res. Rep., no. 8, pp. 1-29, 1979.

[25] K. Sugiyama and M. Toda, "Structuring information for understanding complex systems: A basis for decision making," *Fujitsu Sci. Tech. J.*, vol. 14, no. 2, pp. 144-164, 1985.

[26] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*. New York: Prentice-Hall, 1978.

Kozo Sugiyama (M'78-M'82) was born in Gifu, Japan, on September 17, 1945. he received the B.S., M.S., and Ph.D. degrees in geophysics in 1969, 1971, and 1974, respectively, from the Nagoya University, Nagaya, Japan.

Since April 1974, he has been with the International Institute for Advanced Study of Social Information Science, Fujitsu Ltd., Shizuoka, Japan, where he has been doing research on systems analysis of societal systems, structural modeling, decision support, automatic graph drawing, graphic language, and human-machine interface. He is presently the manager of the First Research Laboratory there, From 1982 to 1983, he was research scholar at the International Institute for Applied Systems Analysis, Laxenburg, Austria, joining the Management and Technology Area. His current research interests are in graphic interfaces, personal/group idea organizers, and conputer-aided creativity.

Dr. Sugiyama is a member of the Society of Instrument and Control Engineers of Japan and the Information Processing Society of Japan.

Kazuo Misue was born in Yamaguchi, Japan, on January 17, 1962. He received the B.S. and M.S. degrees in information science in 1984 and 1986, respectively, from the Science University of Tokyo.

He has been with the International Institute for Advanced Study of Social Information Science, Fujitsu, Ltd., Shizuoka, Japan, since 1986. He has been doing research on diagrammatical abduction support systems. His research interest include graphical user interface, programming languages, and formal semantics.

Mr. Misue is a member of the Information Processing Society of Japan, and the Japan Society for Software Science and Technology.