RESEARCH REPORT ISIS-RR-94-6E

Layout Adjustment and the Mental Map

Kazuo Misue Peter Eades* Wei Lai†
Kozo Sugiyama

May, 1994

Institute for Social Information Science (*ISIS*)
at Numazu

FUJITSU LABORATORIES LTD.

140 Miyamoto, Numazu-shi, Shizuoka 410-03, Japan
Telephone: (Numazu) +81-559-24-7210 Fax: +81-559-24-6180

Layout Adjustment and the Mental Map

Kazuo Misue Peter Eades* Wei Lai† Kozo Sugiyama

Institute for Social Information Science (*ISIS*)
at Numazu

FUJITSU LABORATORIES LTD.

140 Miyamoto, Numazu-shi, Shizuoka 410-03, Japan

Email: misue@iias.flab.fujitsu.co.jp

Abstract

Many models in software and information engineering use graph representations. Automatic graph layout can release humans from the tedium of graph drawing, and is now available in several visualizati on systems. Most automatic layout facilities take a combinatorial description of a graph and produce a layout; these methods are "layout creation" methods. For interactive systems, "layout adjustment" methods must be used to adjust a layout after a change is made by the user or by the application. The use of a layout creation method for layout adjustment may totally rearrange the layout and thus destroy the user's "mental map" of the diagram; thus a set of layout adjustment methods, separate from layout creation methods, is needed. This paper discusses some layout adjustment methods and the preservation of the "mental map" of the diagram. First, several models are proposed to make the concept of "mental map" more precise. Then two kinds of layout adjustments are described. Finally, some experience with visualization systems in which the techniques have been employed is described.

Key words: mental map, layout adjustment, disjoint node images, perspective display method

*Department of Computer Science, The University of Newcastle
†Department of Computer Science, Edith Cowan University

1 Introduction

Graph representations play an important role in software engineering and information systems. Examples are data flow diagrams, state transition diagrams, flow charts, PERT charts, organization charts, Petri nets and entity-relationship diagrams. Graphs are principally used to illustrate relational structures. A node is used to represent an object and an edge represents a relation between objects. For instance, in a state transition diagram, a node is a state and an edge is a state transition.

The usefulness of these graph representations depends on the quality of the layout of the graphs. For instance, Figure 1(a) is confusing and tangled; it gives little assistance to the viewer. The same graph with a better layout in Figure 1(b) clearly shows the relationships between the entities.

Human layout of graphs is a time-consuming and detail-intensive chore. Automatic graph layout can release humans from such chores, and is now available in several visualization systems; see [1, 2, 3, 4] for example. These systems use a variety of algorithms to compute a good layout. Most graph layout algorithms are intended to draw graphs according to some "aesthetic criteria", which measure the readability of a layout. For example, a classical planarity-based layout algorithm (such as described in [5]) aims to draw graphs with as few edge crossings as possible. The state of the art in algorithms for automatic graph layout is surveyed in [6].

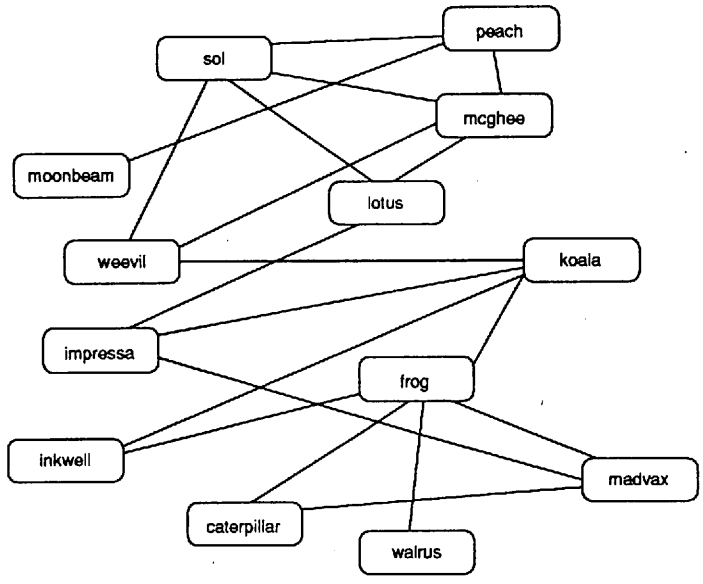
Kamada[7] proposed an architecture for graph drawing systems called a "visualization pipeline". This architecture was refined by Lin[8]. Figure 2 illustrates the basic pipeline for interactive visualization systems. The steps are:

Modeling: The *application* may be a program, a database, a data structure, a communication or security protocol, or a similar computational structure. Relational information in the application is usually modeled as a graph. At this stage, the graph is purely combinatorial and has no geometric attributes. The process of extracting the graph from the application is the *modeling* step.

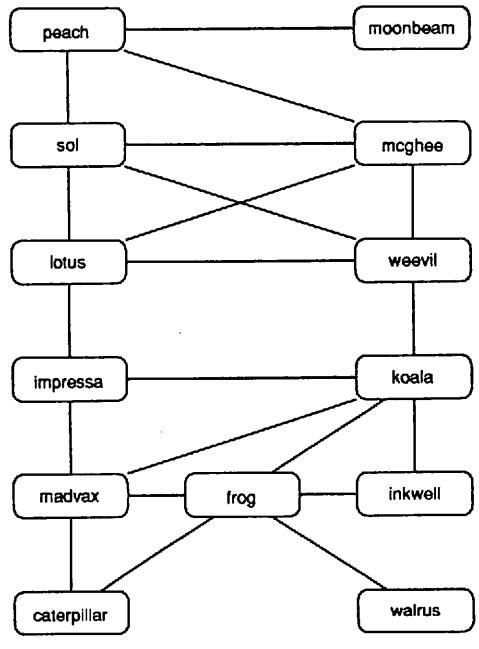
Layout Creation: The graphs created by the modeling step are conventionally drawn with nodes represented by boxes (perhaps enclosing some text) and edges represented by lines between the boxes. The process of adding geometric attributes to the graph to create a *picture* is the *layout creation* step.

Layout Adjustment: In interactive systems, changes to the graph are constantly made, sometimes by the user and sometimes by an application program. These changes may spoil the layout in some local manner; for instance, the addition of a node may overlap an existing node, or an added edge may cross existing edges. An adjustment of the picture is required. The process of modifying geometric attributes of the graph to adjust the picture is the *layout adjustment* step. This paper concerns the layout adjustment step.

Further motivation of layout adjustment techniques comes from some limitations of classical graph drawing algorithms. Many algorithms were originally designed for abstract graphs where nodes take up little or no space. While such graphs are interesting for Graph Theorists, in practical applications the images of nodes are circles, boxes, diamonds and similar



(a) A confused tangled layout



(b) The same graph with good layout

Figure 1: Examples of layout

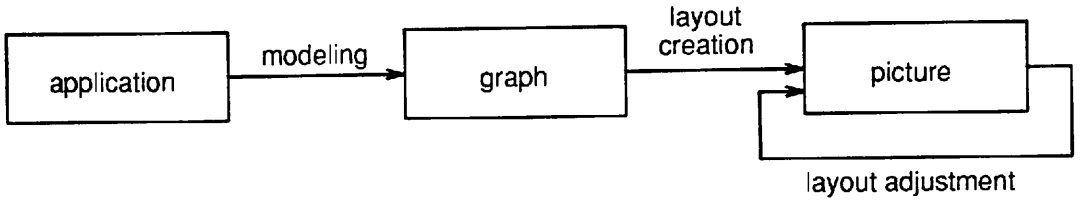


Figure 2: A basic visualization pipeline

shapes, and may contain a considerable amount of text and pictures. Thus classical algorithms typically result in overlaps between node images. An example is in Figure 3. The layout in Figure 3(a) was obtained using a classical "spring" algorithm [9]. When nontrivial sized boxes are represent nodes as in Figure 3(b), the images overlap. To use such algorithms in visualization systems, we need to adjust their output to avoid overlap.

Although the layout adjustment step is important in interactive systems, most existing layout algorithms are designed for the layout creation step. For instance, almost all papers in [6] are concerned with layout creation. Of course, a layout adjustment may be achieved by the application of a layout creation algorithm. However, such an algorithm may totally rearrange the layout to destroy the user's "mental map" of the diagram.

An example is in Figure 4. Initially, the user sees Figure 4(a), and then adds an edge connecting nodes A and C. A planarity-based static layout algorithm would avoid the edge crossing and give Figure 4(b). In this case, the rearrangement of nodes is so severe that the user's mental map of the screen is completely destroyed.

Another example is in Figure 5. The abstract subgraph node F in Figure 5(a) is expanded to form Figure 5(b). The application of a spring algorithm[9] would give Figure 5(c); again the relative positions of nodes have changed and the user's mental map is damaged.

In this paper we discuss some layout adjustment methods and the preservation of the "mental map", or "shape" of the diagram.

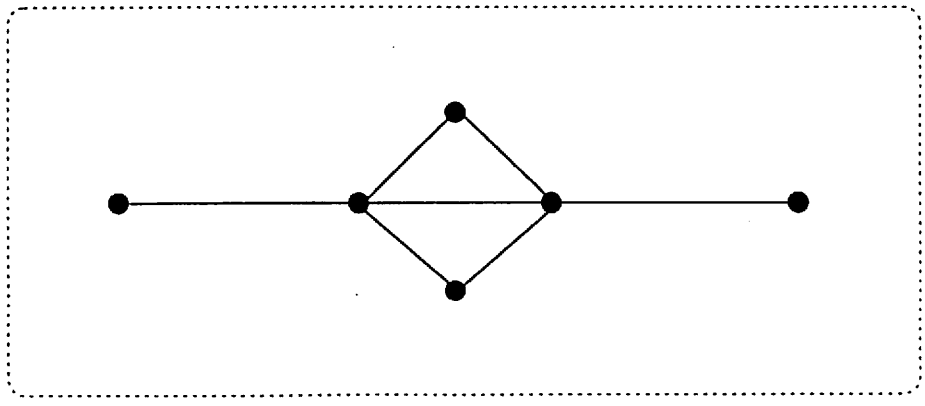
Some previous research on *dynamic graph drawing* appears in [10, 11, 12]. The main aim of dynamic graph drawing is to implement operations such as insertion and deletion of nodes and edges on planar graphs in such a way as to use logarithmic time. Here we are not concerned with planarity (because, as in Figure 4, insistence on planarity leads to destruction of the mental map); further, we believe that linear time, or even quadratic time, is adequate for graph drawing applications.

The concept "mental map" is intuitive. In Section 2 we propose several models to make it more precise.

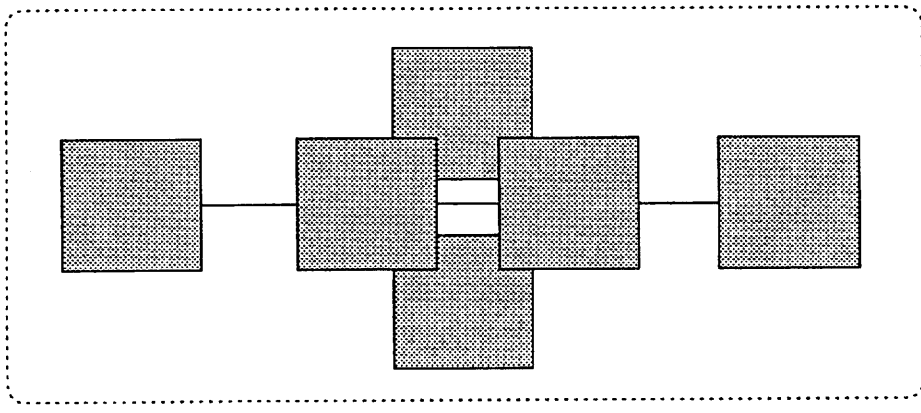
In Section 3 we describe an algorithm for layout adjustment which rearranges a diagram to avoid overlapping nodes. This algorithm can be used after a node is deleted or added to a diagram, or as a postprocess to a classical graph drawing algorithm.

Section 4 describes layout adjustment aimed at changing the focus of interest of the user without destroying the mental map.

Some experience with visualization systems in which the techniques have been employed is described in Section 5.



(a)



(b)

Figure 3: Output of a classical graph drawing algorithm

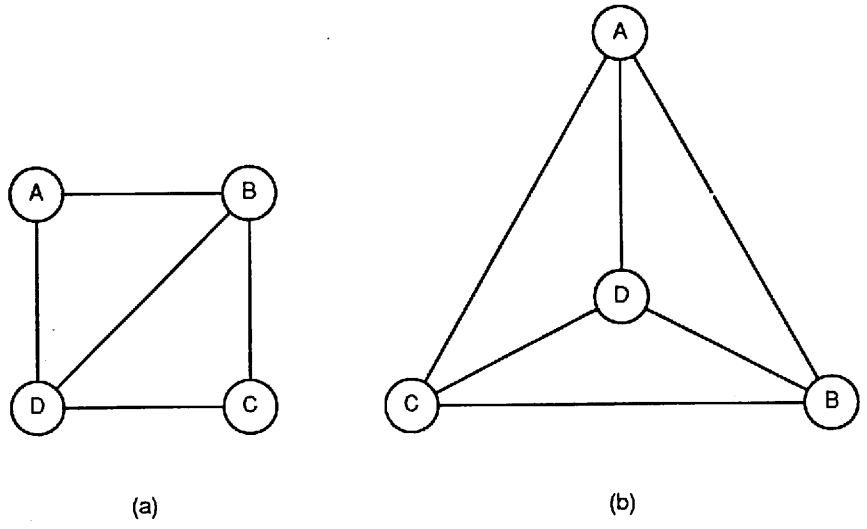


Figure 4: Adding an edge destroys the mental map

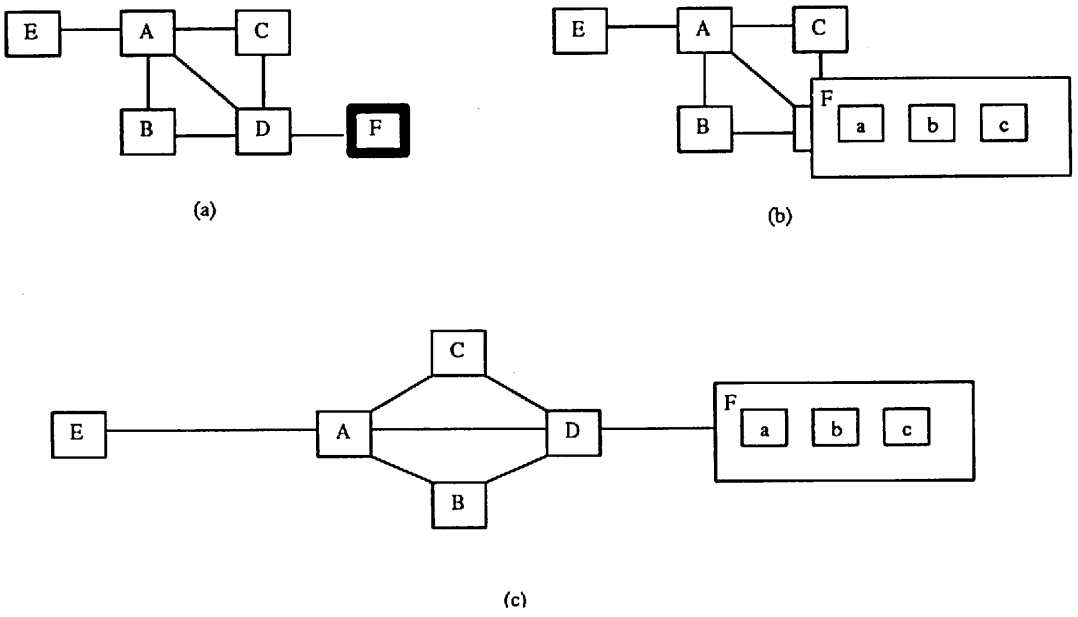


Figure 5: Expanding a subgraph destroys the mental map

2 Models

In this section we will introduce several feasible models for the user's mental map of a diagram.

2.1 Orthogonal Ordering

The most basic mental map is, in Sesame Street terms, preserving up, down, left and right. This concept is discussed below.

A diagram consists of a set V of objects, and each object v appears on the screen as a *visual representation* which is a function which maps V into a space of graphical attributes. The graphical attributes include the position, color, size and shape of the representation of the object; here we are concerned mostly with the position of the object, which we will denote by (x_v, y_v). In practice, (x_v, y_v) gives the pixel coordinates of a reference point (say, the center) of the object.

Suppose that the set V of objects has two visual representations D and D'. The position of v is (x_v, y_v) under D and (x'_v, y'_v) under D'. We say that two representations of V have the same *orthogonal ordering* if for each $u, v \in V$,

- $x_u < x_v$ if and only if $x'_u < x'_v$, and
- $y_u < y_v$ if and only if $y'_u < y'_v$, and
- $x_u = x_v$ if and only if $x'_u = x'_v$, and
- $y_u = y_v$ if and only if $y'_u = y'_v$.

If D has the same orthogonal ordering as D', then we say that the transformation which maps D to D' *preserves orthogonal ordering.*

Another formulation gives further intuition for its significance. Suppose that M is an $|V| \times |V|$ matrix whose ijth entry is one of $I, N, NE, E, SE, S, SW, W, NW$, depending on whether, in the representation D, the ith object is identical to the jth object, or the direction of the ith object from the jth object is North, North-East, East, South-East, South, South-West, West, or North-West. Similarly we can define a matrix M' for the representation D'. Then D and D' have the same orthogonal ordering if and only if $M = M'$. Preservation of these directional relations is clearly a very desirable property for transformations.

Layout adjustment techniques which preserve orthogonal ordering are described in Sections 3 and 4.

2.2 Proximity Relations

Layout adjustment should preserve "proximity relations": items which are close together should stay close together. There are many ways to capture this intuition mathematically.

Several graph-theoretic notions of the "shape" of a set of points are discussed in [13]. The general idea is to define a graph G whose nodes are the points of S, and whose edges are defined by some kind of proximity relation between the points. The graph G is called a *proximity graph*; there are many kinds of proximity graph, and we mention only a sample:

- Simple proximity graphs include the *convex hull* and the *minimum spanning tree* of the set.
- The *delaunay triangulation* is a very strong notion of proximity relationships because several other proximity graphs are subgraphs of the delaunay triangulation.
- A more sophisticated concept is the *sphere of influence graph*, which has an edge between p and q if and only if

$$d(p,q) \leq \min_{p \neq r \in S} d(p,r) + \min_{q \neq s \in S} d(q,s).$$

  Toussaint [13] argues that the sphere of influence graph accurately captures the "low level perceptual structure of visual scenes of dot patterns".

Detailed discussions of the mathematical properties of these proximity graphs can be found in texts on Computational Geometry, such as [14].

Different kinds proximity graphs can be used to formalize concepts of two visual preserving proximity relations. For example, we can use the delaunay triangulation: if a transformation takes a visual representation D and produces a visual representation D' with the same delaunay triangulation, then we say that the transformation *preserves delaunay triangulations.* Since the delaunay triangulation is a very strong notion of proximity relation, a transformation which preserves delaunay triangulations can be regarded intuitively as strongly preserving proximity relations.

No nontrivial transformations which preserve delaunay triangulations (or any other major class of proximity graph) are known, in spite the clear desirability of such transformations. Some research in this direction is described in [15].

2.3 Topology

A set C of curves in 2 dimensions (such as a drawing of a graph) divides the plane into a number of regions. Intuitively, the "dual graph" of C is the graph whose nodes are these regions, with an edge between two nodes which share a boundary.

The dual graph in a sense captures the topological structure of a set of curves. Intuitively, two visual representations have the "same topology" if they have the same dual graph. However, further notation is required so that this notion can be applied to diagrams.

A visual representation of a graph consists of regions (usually circles or boxes) which represent nodes and curves representing edges. Note that the curves representing the edges may cross each other. These curves divide the region of the plane not occupied by node images into subregions, which we call *faces.* A face α can be identified by the circular list $\lambda(\alpha)$ of the edges and nodes encountered in a clockwise traversal of the boundary of the face. The *dual graph* of a visual representation D of a graph is a graph whose vertices are the node images and the faces of D, with an edge between two vertices for which the corresponding regions of the plane share a common nontrivial boundary. The dual graph of the diagram in Figure 6(a) is Figure 6(b), In the dual graph the node labeled 1 represents the face AbBcCgFdeDfEfDa (that is, the outside face), the node labeled 2 represents the face AdeDa, the node labeled 3 represents the face CgFde, and the node labeled 4 represents the face AbBcCed.

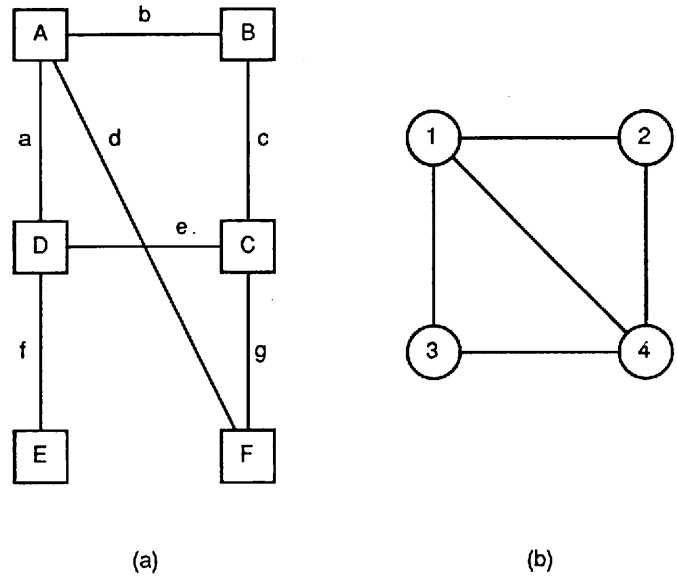


Figure 6: dual graph

Two visual representations have the *same topology* if they have the same dual graph. A layout adjustment *preserves topology* if, for every diagram, the visual representation of the diagram has the same topology as the transformed visual representation. It is clearly desirable that a transformation, should preserve topology, and some described in section 3 and 4 do.

3 Disjoint Node Images

In this section, we consider layout adjustment for ensuring disjointness of node images.

The adjustment should satisfy three requirements:

(1) The new drawing should be compact. Essentially, we must ensure that the new drawing fits the page.
(2) The node images in the new drawing should be disjoint.
(3) The mental map should be preserved.

In Section 3.2 we introduce a new technique which attempts to adjust a layout such that the requirements (1), (2), and (3) above are satisfied. Firstly, however, in Section 3.1 we examine some very simple approaches.

3.1 Simple Techniques

One simple approach is to use uniform scaling. If s, a, b are real numbers with $s > 0$, then scaling moves each point (x, y) to (x', y') defined by

$$(x', y') = (a + s(x - a), b + s(y - b)).$$

The resulting drawing has precisely the same geometrical shape as the original – the transformation just "expands" the diagram by a factor of s about the point (a, b). Thus the

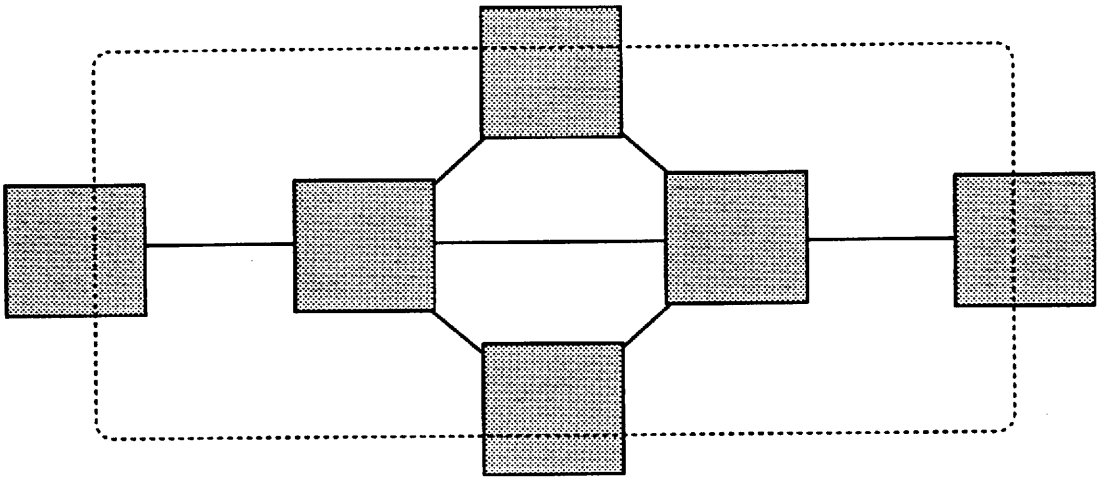


Figure 7: Output of a scaling transformation

transformation preserves the mental map under each of the models in the previous section. Further, by choosing s large enough, we can ensure that the node images are disjoint.

The problem with this simple approach is that in practice the whole diagram tends to grow too large, that is, criterion (1) is often violated. A typical case is illustrated in Figure 7. Here the diagram of Figure 3(b) has been scaled about the center of the page, but the scaling pushes some nodes outside the page boundary.

Another simple approach is to use horizontal sorting, that is, to sort in the x direction, and place node v in a column whose w_v is the width of the node v, in order of x coordinate. A similar approach using sorting in both directions is discussed in [15]. (Note, however, that the methods of [15] are aimed at classical graphs and were not intended to deal with graphs with nontrivial sized nodes). The sorting approach may be used to achieve disjointedness, but does not give a compact drawing. For example, horizontal sorting transforms Figure 8(a) to Figure 8(b); however, Figure 8(a) is already disjoint.

3.2 The Force-Scan Algorithm

The *force-scan* algorithm uses a paradigm similar to the "spring algorithms" (see [16, 9, 7]) to move nodes in both the horizontal and vertical directions to avoid overlaps. The main idea is to choose a "force" f_{uv} between two pairs u, v of nodes so that if u and v overlap then f_{uv} pushes v away from u. The forces are applied in two scans: one from left to right, one from top to bottom.

Although the shape of the image can vary considerably, for the sake of node disjointness we need only consider its bounding box. Throughout this section we will denote the center of the bounding box of the image of node v by $p_v = (x_v, y_v)$.

3.2.1 Desirable Distance and Force

The choice of the force function f_{uv} is critical; experience has shown that the function defined as follows is suitable.

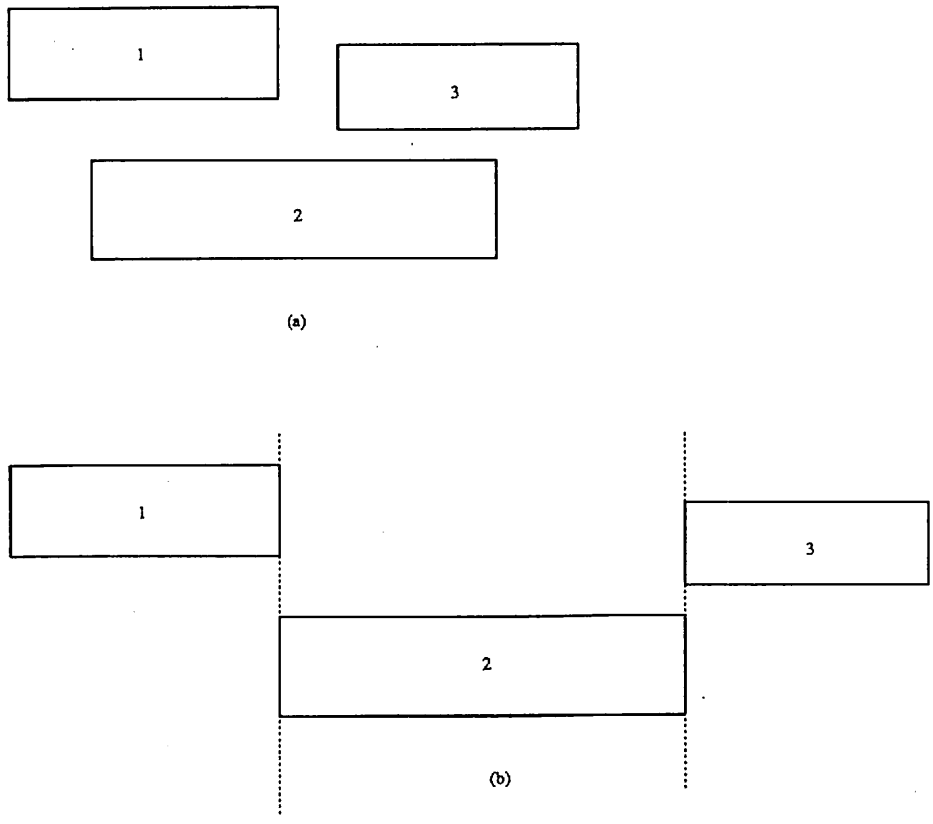


Figure 8: An example of horizontal sorting

We choose the direction of f_{uv} to be along the line from p_u to p_v. The intuition here is that a force in this direction will preserve the mental map. The magnitude of f_{uv} is the difference between the "actual distance" d_{uv} and the "desirable distance" k_{uv} between the node images for u and v. In this way, the force is analogous to a spring which obeys Hooke's law.

More precisely, the *actual distance* d_{uv} is the usual Euclidean distance between p_u and p_v (see Figure 9(a)). The "desirable distance" k_{uv} is defined as follows. Let $r = (x, y)$ be the first point along the line from p_u to p_v for which either $|x - x_u| \geq \frac{w_u + w_v}{2}$ or $|y - y_u| \geq \frac{h_u + h_v}{2}$. In other words, r is the first point along the line from p_u to p_v for which a node image of u centered at p_u is disjoint from a node image of v centered on r. Then the *desirable distance* k_{uv} is the Euclidean distance between p_u and r. This is illustrated in Figure 9(b). The intention of the choice of k_{uv} is not only to make u and v disjoint but to keep the diagram as compact as possible.

Denote the projections of $p - r$ in the x and y directions by k_{uv}^x and k_{uv}^y respectively. Note that either

$$k_{uv}^x = \frac{w_u + w_v}{2} \quad or \quad k_{uv}^y = \frac{h_u + h_v}{2} \tag{1}$$

If the node images of u and v overlap, then the magnitude of the force f_{uv} is $k_{uv} - d_{uv}$. Thus f_{uv} is

$$f_{uv} = \max(0, k_{uv} - d_{uv})\iota_{uv} \tag{2}$$

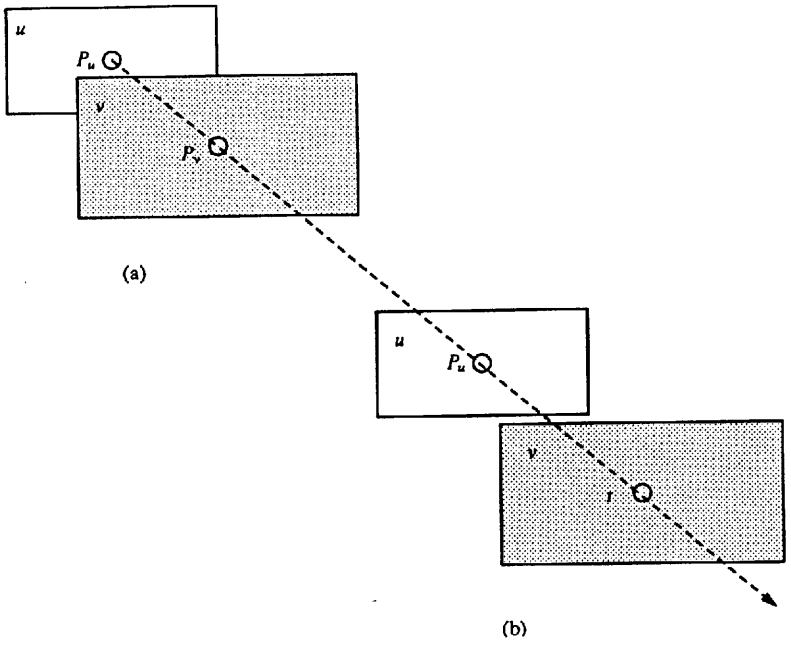


Figure 9: Actual and desirable distances

where ι_{uv} is a unit vector in the direction from p_u to p_v.

In practice a global value g can be added to k_{uv} to force a gap of size g between nodes.

3.2.2 The Scan

The forces are applied in two scans. The first is in the horizontal direction and preserves the horizontal order of the nodes, the second is in the vertical direction and preserves the vertical order of the nodes.

The horizontal scan is illustrated below. We will assume that $x_{v_1} \leq x_{v_2} \leq \cdots \leq x_{v_n}$; we denote the projection of f_{uv} in the x direction as f^x_{uv}.

Algorithm **Horizontal Scan**

$i \leftarrow 1$;
while $i < |V|$ *do*
    Suppose that $x_i = x_{i+1} = \cdots = x_k$;
    $\delta \leftarrow max_{i \leq m \leq k < j \leq |V|} f^x_{v_m v_j}$;
    for $j \leftarrow k+1$ *to* $|V|$ *do* $x_{v_j} \leftarrow x_{v_j} + \delta$;
    $i \leftarrow k+1$;

Algorithm **force-scan** consists of the Horizontal Scan above followed by a similar "Vertical Scan".

3.2.3 Push and Push-pull Force Scan Algorithms

A slight variation of the algorithm described above may be used. We can allow both positive and negative forces; that is, we choose

$$f_{uv} = (k_{uv} - d_{uv})\iota_{uv}. \tag{3}$$

The difference between this and the previous force is that this allows "pulling" forces as well as "pushing" forces. This has the effect of further compaction of diagrams which are too sparsely spread out while preserving the general shape of the diagram.

The algorithm with forces chosen by formula (2) is called the **push force-scan algorithm**. If the forces are chosen by formula (3) in the force-scan algorithm, then it is called the **push-pull force-scan algorithm**.

The two force-scan algorithms give similar results. The output of either force-scan algorithm applied to to Figure 3(b) is in Figure 10. The main difference between the two variants is that the push algorithm will not pull nodes together, it only aims for disjointness. See Figure 11 for examples: the push-pull force-scan algorithm gives Figure 11(c) for both inputs 11(a) and (b), while the push algorithm gives (c) from (b), but leaves (a) with no change.

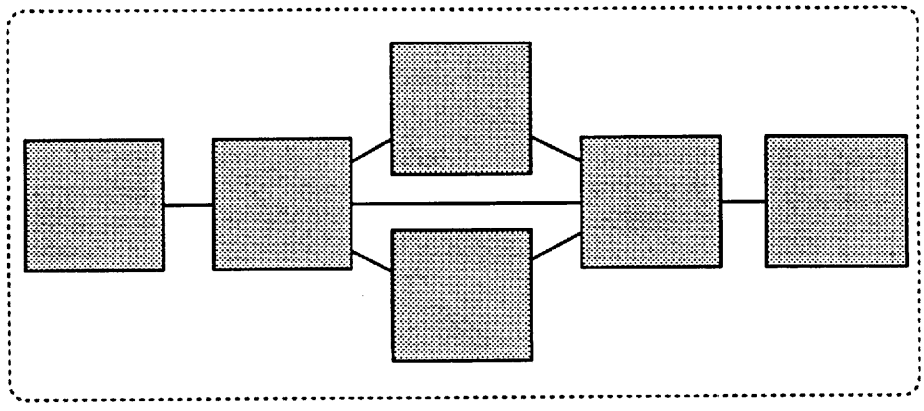


Figure 10: Output of the force-scan algorithm

3.2.4 Mathematical Properties

In this section we discuss some mathematical properties of the force-scan algorithm. We show that both versions preserve orthogonal ordering, and that the *push* version ensures node disjointness.

Theorem 1 *Both force-scan algorithms preserve orthogonal ordering.*

Proof. Before the horizontal scan, we assume that $x_{v_1} \leq x_{v_2} \leq \cdots \leq x_{v_n}$. We define, after the horizontal scan, $x'_{v_1}, x'_{v_2}, \cdots, x'_{v_n}$ as the new x coordinates for $x_{v_1}, x_{v_2}, \cdots, x_{v_n}$ respectively. We need to show that $x_{v'_1} \leq x_{v'_2} \leq \cdots \leq x_{v'_n}$.

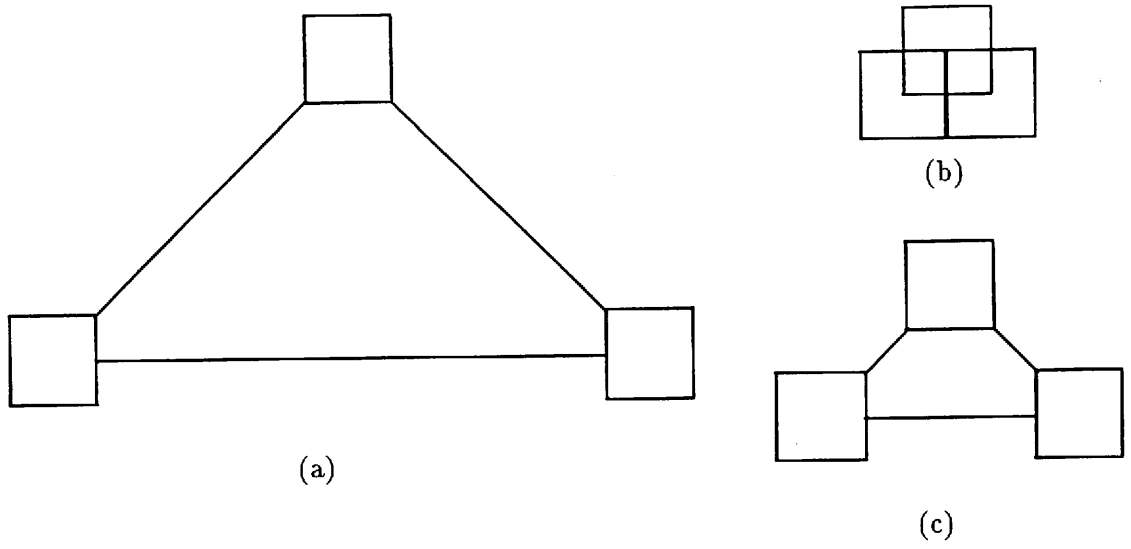


Figure 11: Output of force-scan algorithms

We define a value δ_i for $0 \leq i < n$, as follows. If $i = 0$ or $x_{v_i} = x_{v_{i+1}}$, then $\delta_i = 0$; if $x_{v_i} < x_{v_{i+1}}$, then δ_i is the amount by which $x_{v_{i+1}}, x_{v_{i+2}}, \cdots, x_{v_n}$ are moved at the second last statement of algorithm **Horizontal Scan**. Thus

$$\delta_i = \begin{cases} 0 & \text{if } i = 0 \text{ or } x_{v_i} = x_{v_{i+1}} \\ max_{k \leq m \leq i < j \leq |V|} f^x_{v_i v_j} & \text{if } (x_{v_i} < x_{v_{i+1}}) \text{ and } (x_{v_{k-1}} < x_{v_k} = x_{v_{k+1}} \cdots = x_{v_i}) \end{cases}$$

Also

$$x'_{v_j} = x_{v_j} + \delta_0 + \delta_1 + \cdots + \delta_{j-1}. \tag{4}$$

From (4) we have

$$x'_{v_{i+1}} - x'_{v_i} = x_{v_{i+1}} - x_{v_i} + \delta_i. \tag{5}$$

For the push force-scan algorithm, it is clear that $\delta_i \geq 0$ for $1 \leq i < n$, and thus $x'_{v_{i+1}} \geq x'_{v_i}$; in other words, orthogonal ordering is preserved.

Now we consider the push-pull force-scan algorithm. If $x_{v_{i+1}} > x_{v_i}$, then

$$\begin{aligned} \delta_i &= max_{k \leq m \leq i < j \leq |V|} f^x_{v_i v_j} \\ &\geq f^x_{v_i v_{i+1}} \\ &= \frac{k_{v_i v_{i+1}} - d_{v_i v_{i+1}}}{d_{v_i v_{i+1}}} (x_{v_{i+1}} - x_{v_i}) \end{aligned} \tag{6}$$

Note that in the last expression the force (that is, $f_{v_i v_{i+1}} = k_{v_i v_{i+1}} - d_{v_i v_{i+1}}$) could have a non-positive value. However, combining (5) and (6) we have

$$x'_{v_{i+1}} - x'_{v_i} = x_{v_{i+1}} - x_{v_i} + \delta_i$$

$$\begin{aligned} &\geq x_{v_{i+1}} - x_{v_i} + \frac{k_{v_iv_{i+1}} - d_{v_iv_{i+1}}}{d_{v_iv_{i+1}}}(x_{v_{i+1}} - x_{v_i}) \\ &= \frac{k_{v_iv_{i+1}}}{d_{v_iv_{i+1}}}(x_{v_{i+1}} - x_{v_i}) \end{aligned} \tag{7}$$

However, $x_{v_{i+1}} > x_{v_i}$, and both $k_{v_iv_{i+1}}$ and $d_{v_iv_{i+1}}$ are positive values. Hence $x'_{v_{i+1}} > x'_{v_i}$. Similarly, we can prove that a vertical scan preserves vertical ordering. □

Theorem 2 *The push force-scan algorithm ensures that its output has disjoint node images.*

Proof. For $x_{v_j} \geq x_{v_i}$, by (4) we have

$$x'_{v_j} - x'_{v_i} = x_{v_j} - x_{v_i} + \delta_i + \cdots + \delta_{j-1} \tag{8}$$

Now $\delta_i \geq 0$ for all $1 \leq i < n$ in the push force-scan algorithm. Thus

$$\begin{aligned} x'_{v_j} - x'_{v_i} &\geq x_{v_j} - x_{v_i} + \delta_i \\ &= x_{v_j} - x_{v_i} + \frac{k_{v_iv_j} - d_{v_iv_j}}{d_{v_iv_j}}(x_{v_j} - x_{v_i}) \\ &= \frac{k_{v_iv_j}}{d_{v_iv_j}}(x_{v_j} - x_{v_i}) \\ &= k^x_{v_iv_j} \end{aligned} \tag{9}$$

Similarly, for $y_{v_j} \geq y_{v_i}$, thus

$$\begin{aligned} y'_{v_j} - y'_{v_i} &\geq \frac{k_{v_iv_j}}{d_{v_iv_j}}(y_{v_j} - y_{v_i}) \\ &= k^y_{v_iv_j} \end{aligned} \tag{10}$$

From (1), either

$$k^x_{v_iv_j} = \tfrac{w_{v_i}+w_{v_j}}{2} \text{ or } k^y_{v_iv_j} = \tfrac{h_{v_i}+h_{v_j}}{2}.$$

If $k^x_{v_iv_j} = \frac{w_{v_i}+w_{v_j}}{2}$, then from (9) we deduce

$$x'_{v_j} - x'_{v_i} \geq \frac{w_{v_i} + w_{v_j}}{2},$$

and thus the rectangles are separated in the x direction. Otherwise $k^y_{v_iv_j} = \frac{h_{v_i}+h_{v_j}}{2}$ and from (10) we deduce

$$y'_{v_j} - y'_{v_i} \geq \frac{h_{v_i} + h_{v_j}}{2},$$

thus the rectangles are separated in the y direction. □

3.2.5 Remarks

We have experimented with algorithm **force-scan** over a wide class of sets of overlapping nodes and found that it is quite effective. It tends to give both a more compact drawing as well as preserving the mental map of the original diagram. One striking aspect is that, like many "spring" algorithms, it seems to preserve the symmetries of the input diagram – this is very important for the user's perception.

The push-pull force-scan algorithm does not always ensure that its output has disjoint node images. However, a variation can be formed by repeating the push-pull force-scan algorithm until node images are disjoint. In practice we have observed that if it is repeated a small number of times, the value δ in the horizontal and vertical scans becomes non-negative and the output has disjoint node images.

The force scan algorithm can be applied recursively for compound graphs. Figure 12 shows an example, in which the output of either force-scan algorithm recursively on Figure 12(a) is Figure 12(b). The recursive version preserves the topology of the box-in-box relations in compound graphs. Further, a variation which moves lines representing edges as well as nodes preserves topology [17].

In our experience, all variations tend to preserve some proximity graphs. This is important for the email visualizer in Section 5.1. However, we have been unable to formalize this experience with a mathematical proof.

The time complexity of computing the forces f_{uv} for all u and v is $O(n^2)$, where $n = |V|$. Further, the force between pairs of nodes is tested once in the algorithm **Horizontal Scan**. Thus the time complexity of the push force-scan algorithm is $O(n^2)$. Each iteration of the push-pull algorithm has time complexity $O(n^2)$. For visualization, the number of nodes is relatively small (because total screen size is limited) and both versions have instantaneous response time.

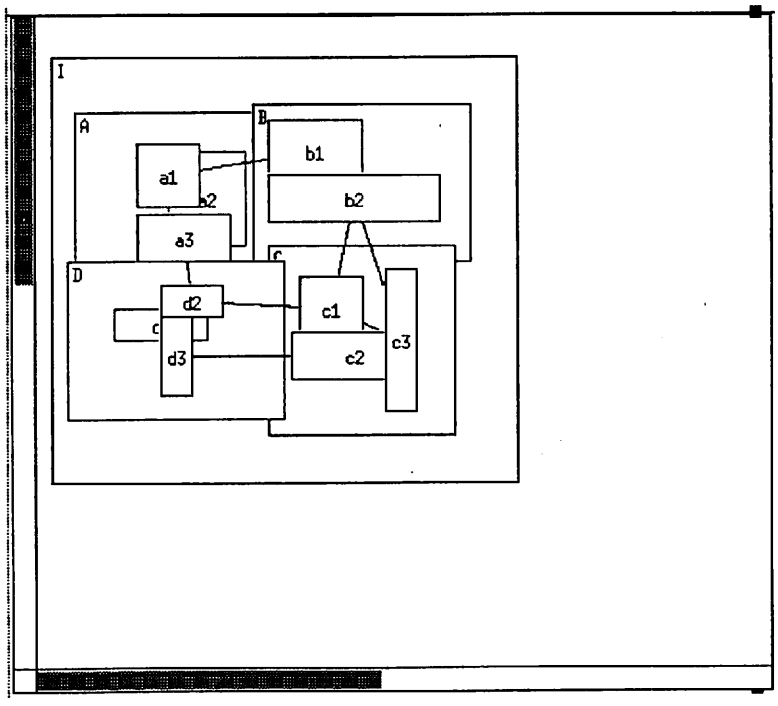
4 Prespective Display Method

4.1 The Whole and Detail View Problem

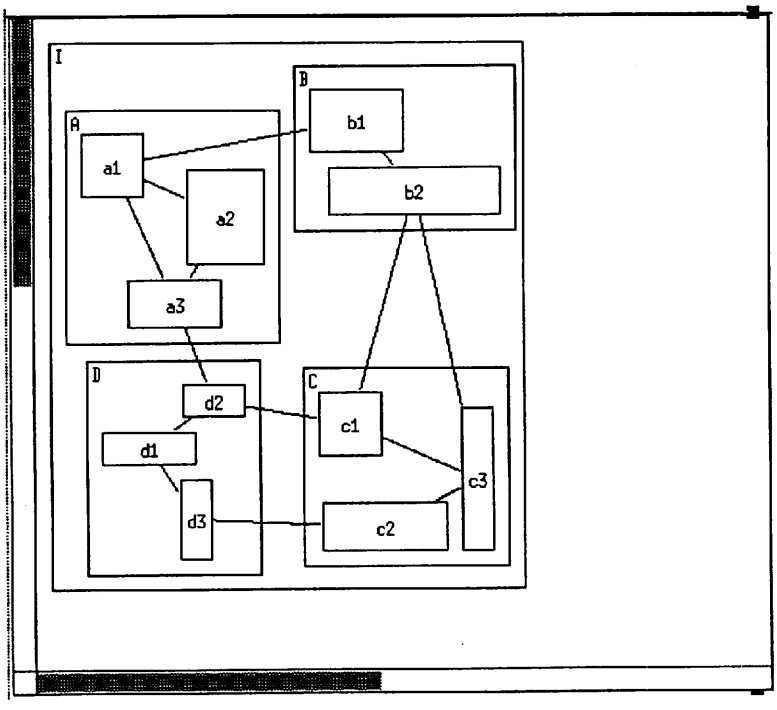
While the processing power of computers has increased rapidly in recent years, their display power, in terms of size and resolution of screens, has not kept pace. Figure 13 shows an example of diagram too large for the screen. Only part of the diagram can been seen at a time. In Figure 14, the same diagram is reduced to show the whole part. We can see the whole diagram but cannot see it in detail. In most visualization systems, therefore, we find that it is difficult to obtain whole and detail views of large display objects, such as diagrams, maps and text, on an ordinary screen. We will call the problem of resolving the conflict between whole views for global information and detailed views for detail information in limited areas "*whole and detail view problem (WDVP)*"[18].

Several discussions of WDVP have appeared. Furnas proposed a generalized fisheye view to show large amount of information, balancing local detail and global context[19]. Leung and Apperley present a taxonomy of graphical data presentation by using such fisheye views[20].

In this section, we discuss techniques to solve WDVP with a *layout adjustment*. The next section introduces the requirements for WDVP, and the following section describes



(a)



(b)

Figure 12: An example of applying force-scan recursively

and evaluates three *perspective* mappings, which are display methods related to the Fisheye approach.

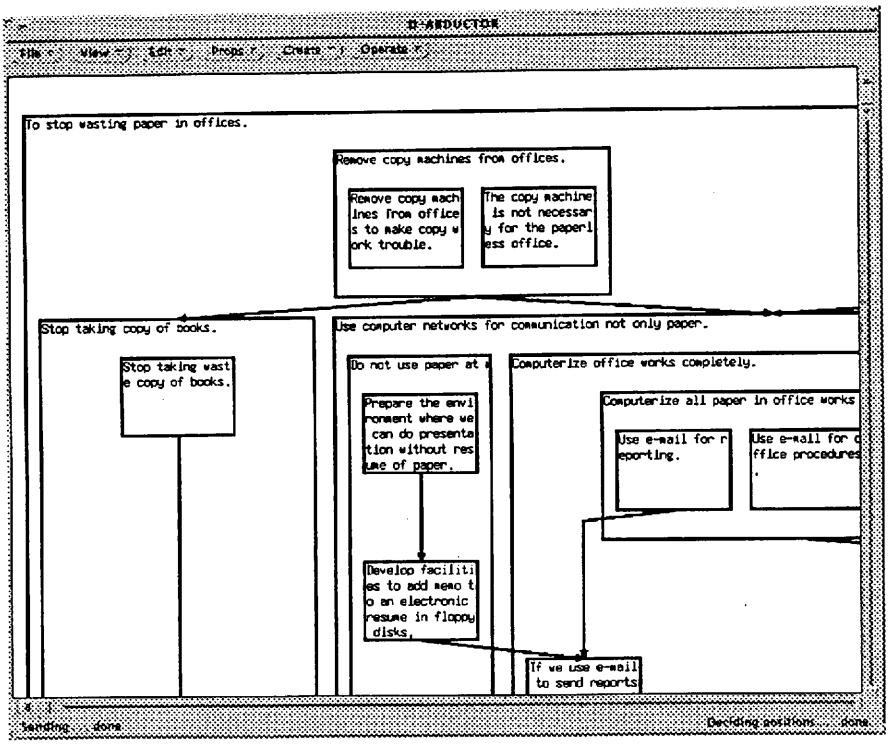


Figure 13: A diagram which is too large for the screen

4.2 Requirements for Display Methods

With respect to WDVP, visualization systems have following requirements for layout adjustment:

(1) Detailedness
Display necessary details of node images. The user needs to read characters and symbols, and to identify line styles and font styles.

(2) Wholeness
Display the whole graph image. The user needs to understand the whole structure, especially for navigational purposes.

(3) Simultaneity
Display all the information simultaneously. The user dislikes switching among images frequently, and remembering hidden information.

(4) Image-uniqueness
Display only one image of a single object at a time. It is difficult for the users to identify parts of one image with parts of another image.

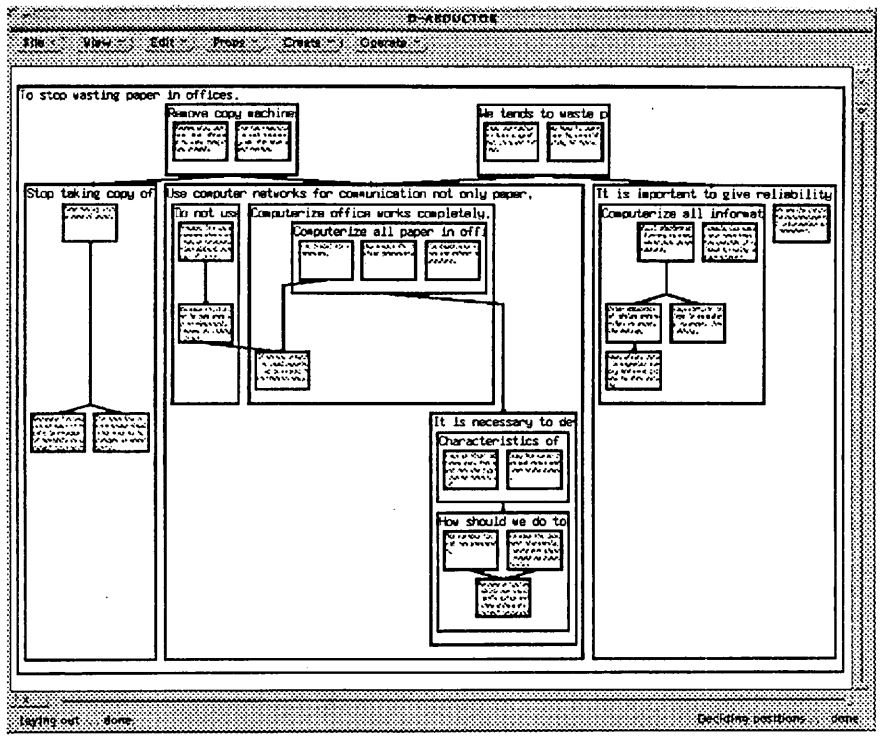


Figure 14: A diagram represented totally but not in detail

As well as the four general requirements above, there are usually application specific requirements; for example, for route maps of subways, preservation of topology with respect to the actual route.

All requirements above are desirable for layout adjustment for WDVP. In this paper we concentrate on the seemingly incompatible requirements detailedness and wholeness.

We can classify display methods employed by most current visual systems into three types: (a) *detailed-image method* (DIM), which displays detailed images at all times; (b) *switching-image method* (SIM), which provides several images with different magnification ratios, and facilities for the user to switch among them; and (c) *multi-image method* (MIM), which displays several images on screens or windows at a time.

Table 1 classifies these types by simultaneity and image-uniqueness assuming that they satisfy both detailedness and wholeness. DIM is omitted from the table because it does not satisfy wholeness. SIM and MIM do not satisfy both simultaneity and image-uniqueness. Our proposed *perspective display method* (PDM) is designed to satisfy all four requirements: (1) – (4). Our technique has been successfully used in the idea-organizer D-ABDUCTOR[18].

4.3 Perspective Mappings

The aim of PDM is to show a whole outline and local details of a figure transformed by mappings, which magnify parts near a viewpoint to display them in detail and demagnify parts distant from a viewpoint to display them rather roughly.

Table 1: Classification of display methods.

Display Method		simultaneity	image-uniqueness
SIM *1		no	yes
MIM	superimposition type *2 tiling type *3 overlapping type *4	yes	no
PDM	fisheye mapping orthogonal fisheye mapping biform mapping	yes	yes

*1 for example, vi (text editor)
*2 for example, KJ-editor[21]
*3 for example, Aldus SuperPaint (software of Macintosh)
*4 for example, xdvi (DVI previewer for the X window system)

4.3.1 Fisheye Mapping

A early idea to solve WDVP was to display large objects distorted as if through an optical fisheye lens. A fisheye lens is very wide-angle (e.g., 180 degrees); this greatly enlarges the area around the viewpoint. To simulate a fisheye lens, the *fisheye mapping* (FEM) is constructed by using a reverse *tan* function so as to map an infinite domain into a finite area. FEM has the flavor of a optical fisheye lens but can have two or more viewpoints. FEM moves the point (x, y) to (x', y'), where x' and y' are defined as follows:

$$x' = \frac{1}{n}\sum_{i=0}^{n-1}\phi_i \tag{11}$$

$$y' = \frac{1}{n}\sum_{i=0}^{n-1}\psi_i \tag{12}$$

$$\phi_i = \begin{cases} 0 & \text{if } x = p_i \text{ and } y = q_i \\ l_i \cdot \cos\theta_i & \text{otherwise} \end{cases} \tag{13}$$

$$\psi_i = \begin{cases} 0 & \text{if } x = p_i \text{ and } y = q_i \\ l_i \cdot \sin\theta_i & \text{otherwise} \end{cases} \tag{14}$$

$$\theta_i = \tan^{-1}\frac{y - q_i}{x - p_i} \tag{15}$$

$$l_i = \frac{2r}{\pi} \cdot \tan^{-1}\frac{\sqrt{(x - p_i)^2 + (y - q_i)^2}}{s_i} \tag{16}$$

$$(i = 0, 1, \ldots, n-1).$$

Viewpoints are specified as the points (p_i, q_i). Expression (15) finds polar angle θ_i for a viewpoint (p_i, q_i). Expression (16) finds the new distance l_i of the point (x, y) from a viewpoint (p_i, q_i). The constant s_i controls the magnification ratio at the viewpoint (p_i, q_i). The

coefficient $2r/\pi$ ensures that the object is mapped to within a circle of radius r. Expressions (13) and (14) are the mappings for each viewpoint. They preserve polar angles of original points for the viewpoint, and map an object at (x, y) to a position at a distance of l_i form the viewpoint. The mapping for several viewpoints is the arithmetic mean of mappings for each viewpoint.

Figure 15 shows a representation of a diagram by FEM with a viewpoint which is on node 3. Figure 16 also shows another representation the same diagram by FEM, but with two viewpoints that are on node 3 and node 25. It is evident from these figures that FEM does not preserve orthogonal ordering.

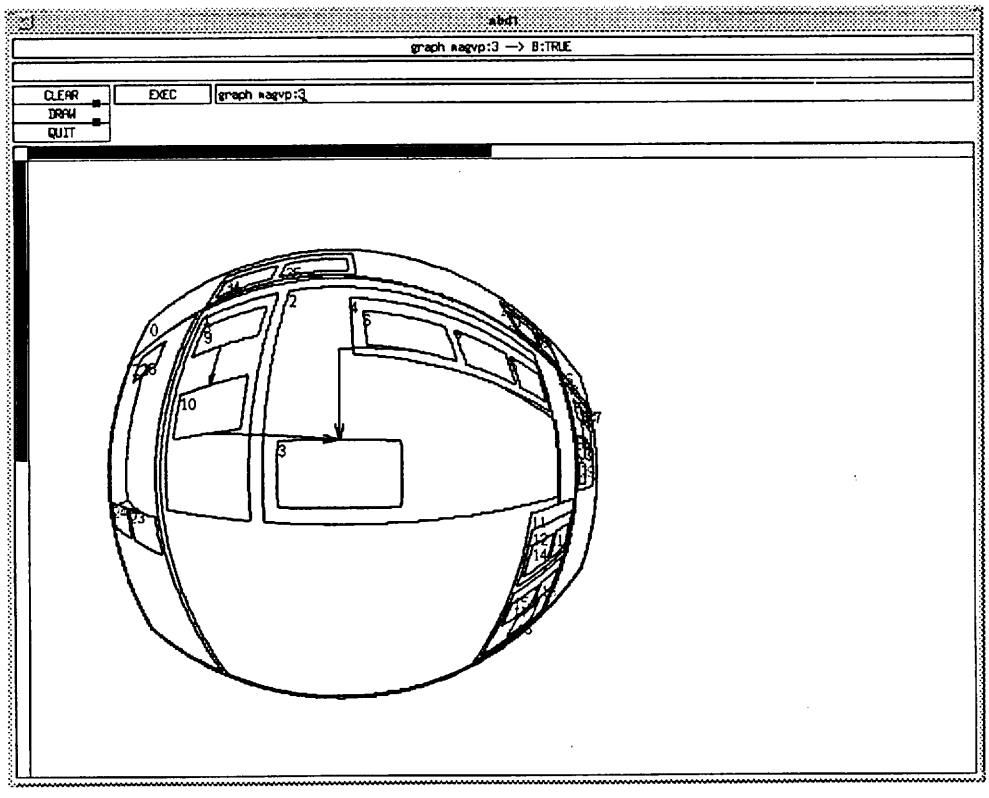


Figure 15: A representation in the fisheye mapping

4.3.2 Orthogonal Fisheye Mapping

The *orthogonal fisheye mapping* (OFM) is similar to FEM but the mappings for the two axes are independent of each other, and OFM preserves orthogonal ordering. OFM also preserves straightness of straight lines parallel to x- or y-axis, and orthogonality of pairs of lines parallel to x- or y-axis. OFM moves the point (x, y) to (x', y'), where x' and y' are defined as follows:

$$x' = \frac{1}{n} \sum_{i=0}^{n-1} \phi_i \tag{17}$$

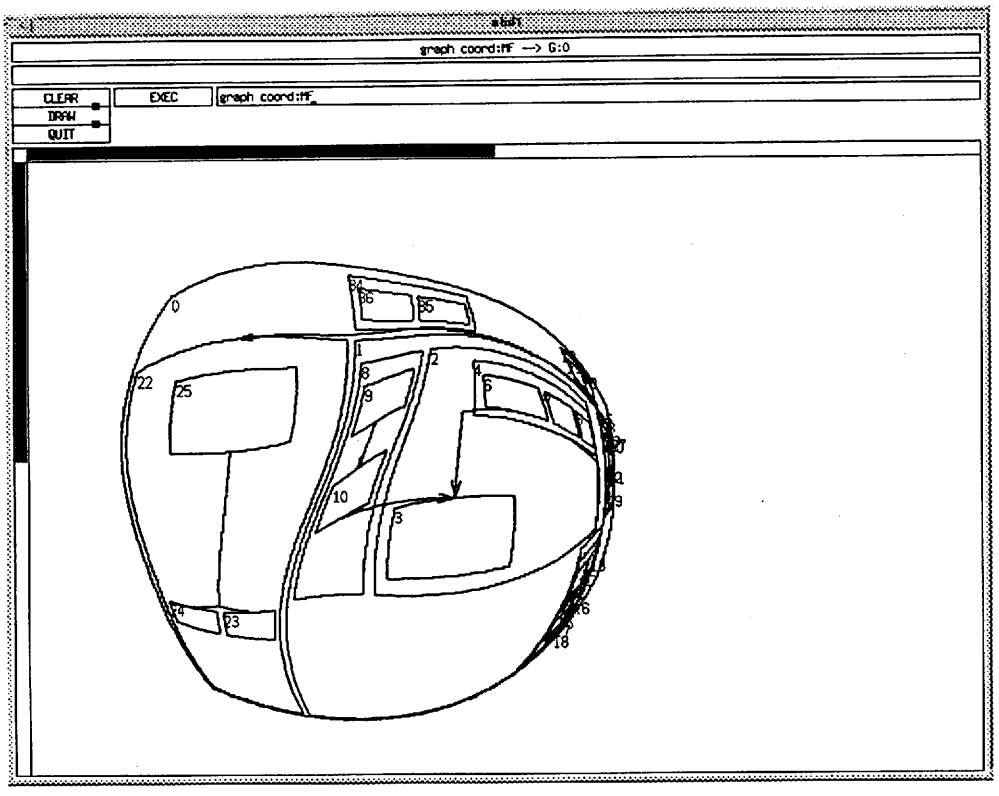


Figure 16: A representation in the fisheye mapping with two viewpoints

$$y' = \frac{1}{n}\sum_{i=0}^{n-1}\psi_i \tag{18}$$

$$\phi_i = \frac{r}{\pi}\cdot\tan^{-1}\frac{x-p_i}{s_i} \tag{19}$$

$$\psi_i = \frac{r}{\pi}\cdot\tan^{-1}\frac{y-q_i}{s_i} \tag{20}$$

$$(i = 0, 1, \ldots, n-1).$$

Viewpoints are specified as the points (p_i, q_i). Expression (19) and (20) define the mappings for a viewpoint (p_i, q_i), and find a new distance of a point from the viewpoint by using a constant s_i and the reverse *tan* function for each axis. The constant s_i has the same role as s_i of FEM. The coefficient r/π ensures that the object stays within a square of side r. The mappings for several viewpoints, which are represented by expression (17) and (18) are the arithmetic mean of mappings for each viewpoint.

Figure 17 shows representations of a diagram by OFM. It is obvious that OFM preserves orthogonal ordering since the mappings (17) and (18) are monotone increasing.

Although FEM and OFM can theoretically display an infinite domain on a finite area, in practice they practically tend to crush surrounding areas infinitely, and make such areas invisible.

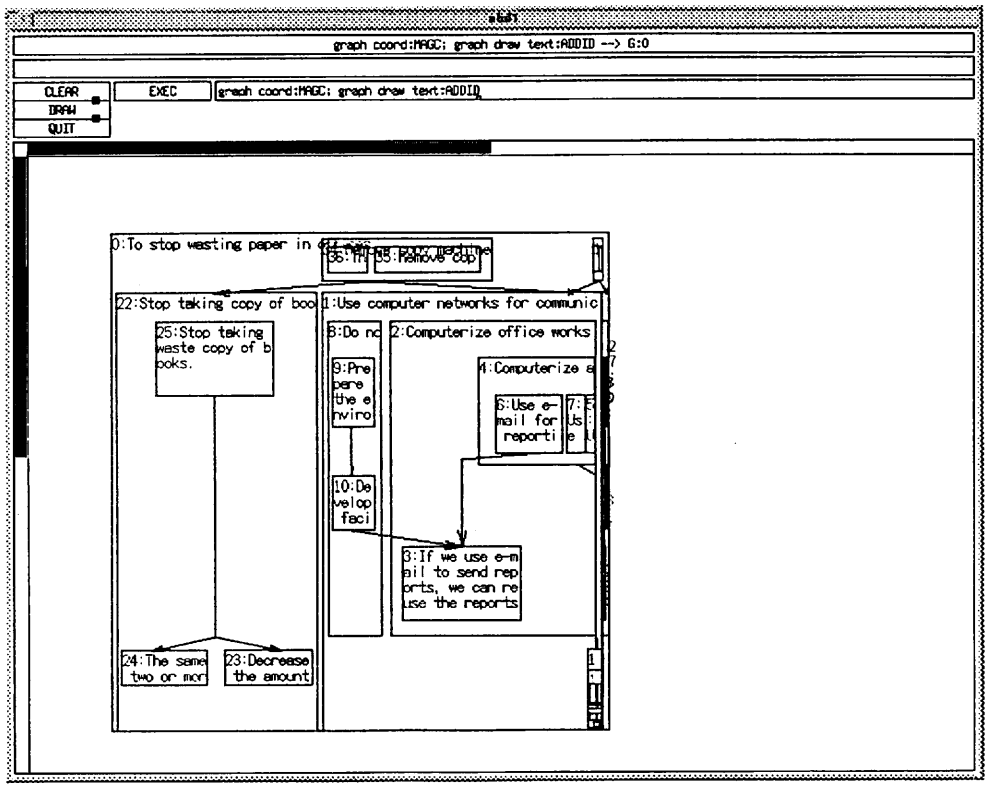


Figure 17: A representation in the orthogonal fisheye mapping

4.3.3 Biform Mapping

The *biform mapping* (BFM) is constructed to overcomes the shortcomings of FEM and OFM. The method uses *view areas*, which are rectangles on the screen, instead of viewpoints. BFM preserves the aspect ratio of the rectangular frames of display objects. The display objects are magnified uniformly in each view area, and demagnified uniformly outside the view areas (thus we use the name "biform"($\approx$ two uniforms)). The magnification ratios for each axis are equal to magnify view areas similarly.

The magnified areas are defined by a set

$$I_x = \{[a_1, b_1], [a_2, b_2], \ldots, [a_m, b_m]\} \tag{21}$$

of disjoint intervals in the x direction, and a set

$$I_y = \{[c_1, d_1], [c_2, d_2], \ldots, [c_n, d_n]\} \tag{22}$$

in y direction. In practice, the intervals are derived from the view areas. Figure 18 illustrates relationships between view areas and magnified areas. Assuming that the largest rectangle corresponds to the outer frame of an object and four rectangles with thick lines are view areas; the areas with //// are magnified in the direction of x-axis and the areas with \\\\ are magnified in the direction of y-axis. Note that the number of view area may differ from the number of magnified areas, as in Figure 18.

BFM moves the point (x, y) to (x', y'), where x' and y' are defined as follows:

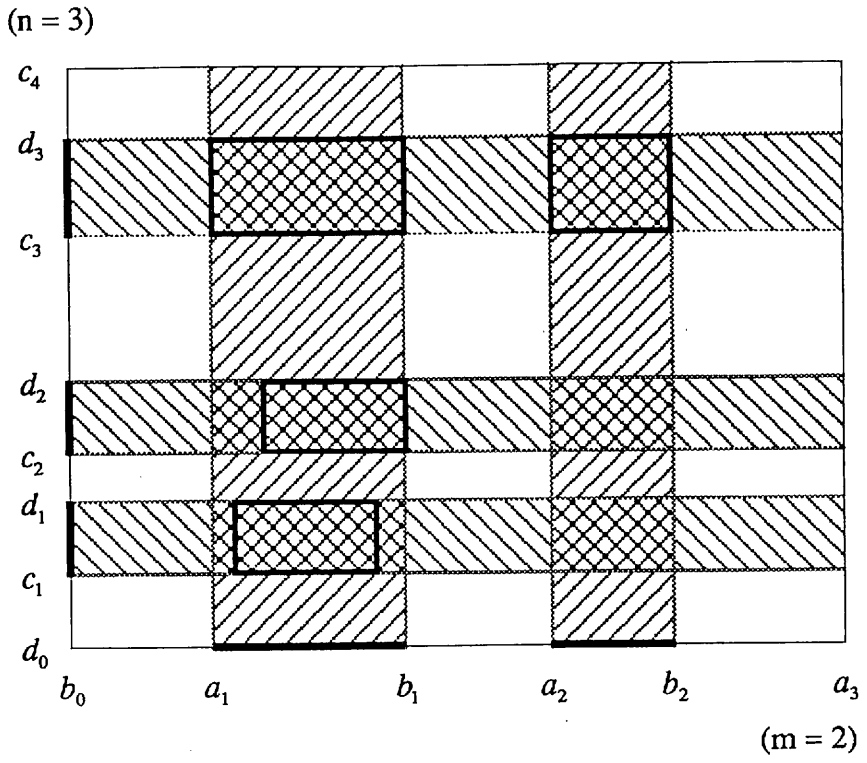


Figure 18: Magnified areas in the biform mapping

$$x' = \begin{cases} b_0 + \lambda(x - b_i) + \sigma\sum_{k=1}^{i}(b_k - a_k) + \lambda\sum_{k=1}^{i}(a_k - b_{k-1}) \\ \qquad \text{if } b_i < x \le a_{i+1} \qquad (i = 0, 1, \ldots, m) \\ b_0 + \sigma(x - a_i) + \sigma\sum_{k=1}^{i-1}(b_k - a_k) + \lambda\sum_{k=1}^{i}(a_k - b_{k-1}) \\ \qquad \text{if } a_i < x \le b_i \qquad (i = 1, 2, \ldots, m). \end{cases} \tag{23}$$

$$y' = \begin{cases} d_0 + \mu(y - d_j) + \sigma\sum_{l=1}^{j}(d_l - c_l) + \mu\sum_{l=1}^{j}(c_l - d_{l-1}) \\ \qquad \text{if } d_j < y \le c_{j+1} \qquad (j = 0, 1, \ldots, n) \\ d_0 + \sigma(y - c_j) + \sigma\sum_{l=1}^{j-1}(d_l - c_l) + \mu\sum_{l=1}^{j}(c_l - d_{l-1}) \\ \qquad \text{if } c_j < y \le d_j \qquad (j = 1, 2, \ldots, n). \end{cases} \tag{24}$$

Here, m and n are the number of magnified areas in the directions of x- and y-axis, respectively. The magnified areas are $[a_i, b_i]$ and $[c_j, d_j]$ for $1 \le i \le m$, $1 \le j \le n$, and the remainder demagnified. The points (b_0, d_0) and (b_{m+1}, d_{n+1}) are respectively the bottom-left vertex and the top-right vertex of the outer rectangle. A coefficient σ is the magnification ratio of the magnified areas. Coefficients λ and μ are the magnified ratios of the remainder in the directions x- and y-axis respectively. They are found as follows by using σ:

$$\lambda = \frac{(a_{m+1} - b_0) - \sigma\sum_{k=1}^{m}(b_i - a_i)}{\sum_{k=1}^{m+1}(a_k - b_{k-1})} \tag{25}$$

$$\mu = \frac{(c_{n+1} - d_0) - \sigma\sum_{l=1}^{n}(d_i - c_i)}{\sum_{l=1}^{n+1}(c_l - d_{l-1})}. \tag{26}$$

Figure 19 shows representations of a diagram by BFM. BFM also preserves the mental map of the orthogonal ordering model since the mappings (23) and (24) are monotone increasing.

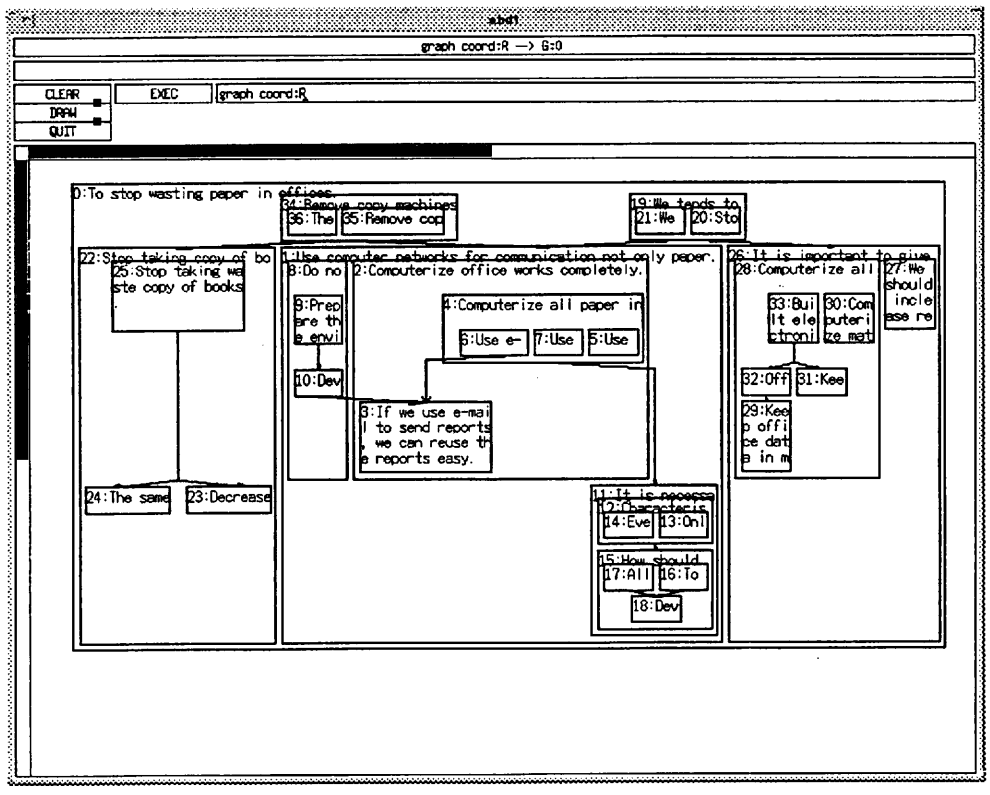


Figure 19: A representation in the biform mapping

5 Examples

5.1 A Visualizer of e-mail Traffic

In this section we illustrate the use of algorithms for node disjointness in an application: email traffic analysis. A textual file records information about frequency of e-mail messages. If the communication between two persons is very frequent, then their relationship is very close. It is difficult to infer these relationships from the e-mail traffic text file, but it is easy using a picture. This picture represents persons on the screen as boxes enclosing names and the closeness of two users is represented by the Euclidean distance between the boxes: for example, the Euclidean distance could be inversely proportional to the number of email messages exchanged.

The user interface for the visualizer is shown in Figure 20 and 21 (This picture depicts a real network, but the user names have been changed for confidentiality.).

For this application, the *modeling* step of the visualization pipeline consists of reading the text file and producing a weighted complete graph, where the weight w_{uv} of an edge represents the desirable Euclidean distance between two nodes u and v.

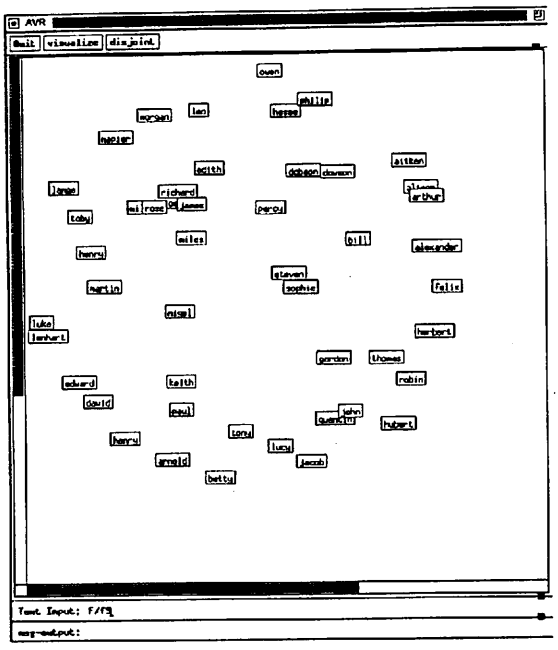


Figure 20: A result of visualize operation

Of course it is impossible in general to compute a layout in which the distance is precisely w_{uv}. However, a *spring algorithm* [9] can be used to give an reasonable approximation. This technique simulates a dynamic system in which every two nodes are connected by a "spring" of desirable length. The optimal layout is generated when the total spring energy of the system is minimal. There are several variations [22, 9, 7]. For our case, the weight of an edge is the desirable length.

Note that we only draw the nodes of diagram in this application; the edges are not represented visually.

The spring algorithm gives a good approximation to the desired distances, but the node images for closely related users overlap each other. This is shown in Figure 20.

The force scan algorithm of Section 3 can be applied. This adjusts the layout to remove overlaps, but preserves the general shape of the diagram, including the approximate distances between nodes. In practice we have found that the technique preserves several proximity graphs, but we have been unable to prove this in general.

5.2 Dynamic Layout for Compound Graphs

In interactive systems, a subgraph is often changed. Examples are expanding or collapsing a subgraph, and inserting or deleting a node in a subgraph. A variation of the force-scan algorithm can be formed for adjusting the layout of a graph after such a change.

Figure 22 shows an example of inserting the node $a4$ into the subgraph A in a diagram. In this case, only the subgraph A is changed. We need to rearrange the nodes in A, and this may resize A. Thus a rearrangement of the ancestors of A may be needed. After applying a variation of the force-scan algorithm to the node A and its ancestors, the result is shown in Figure 23.

Note that the nodes B, C and D (subgraphs) do not need to be internally rearranged,

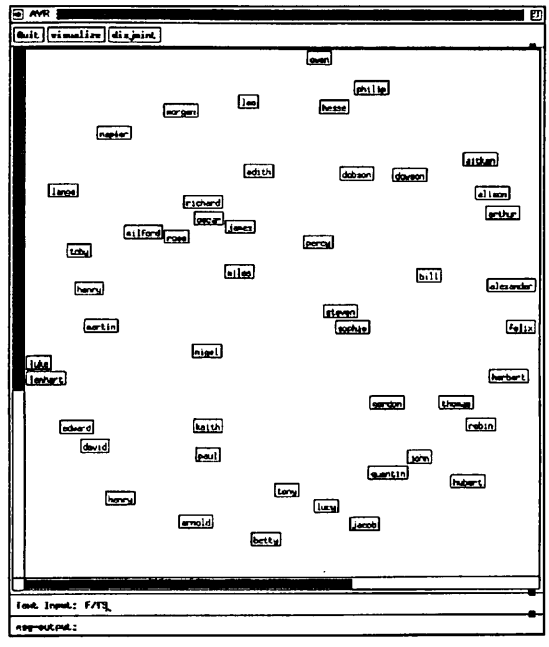


Figure 21: A result of disjoint operation

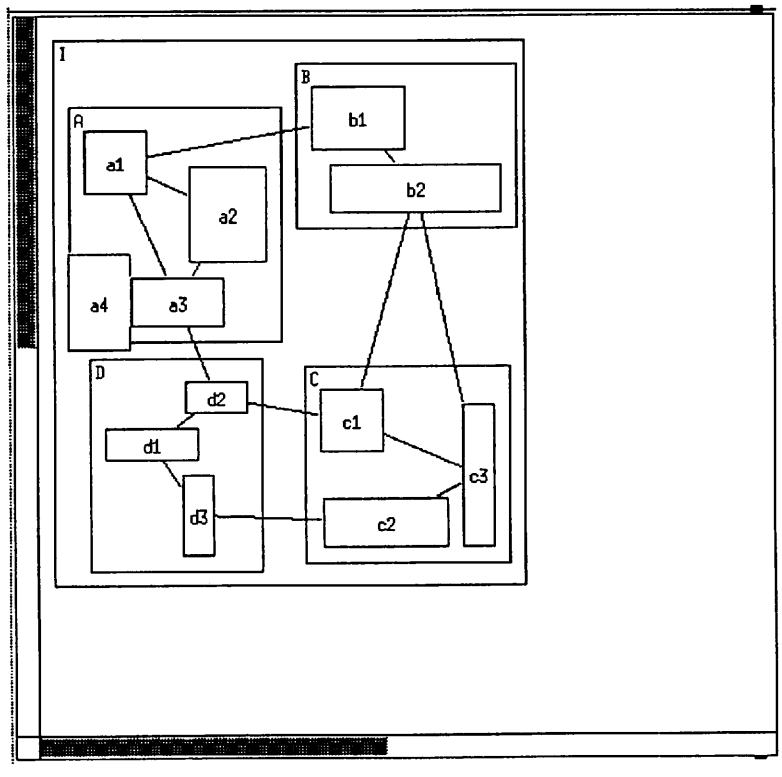


Figure 22: Inserting a node into a subgraph

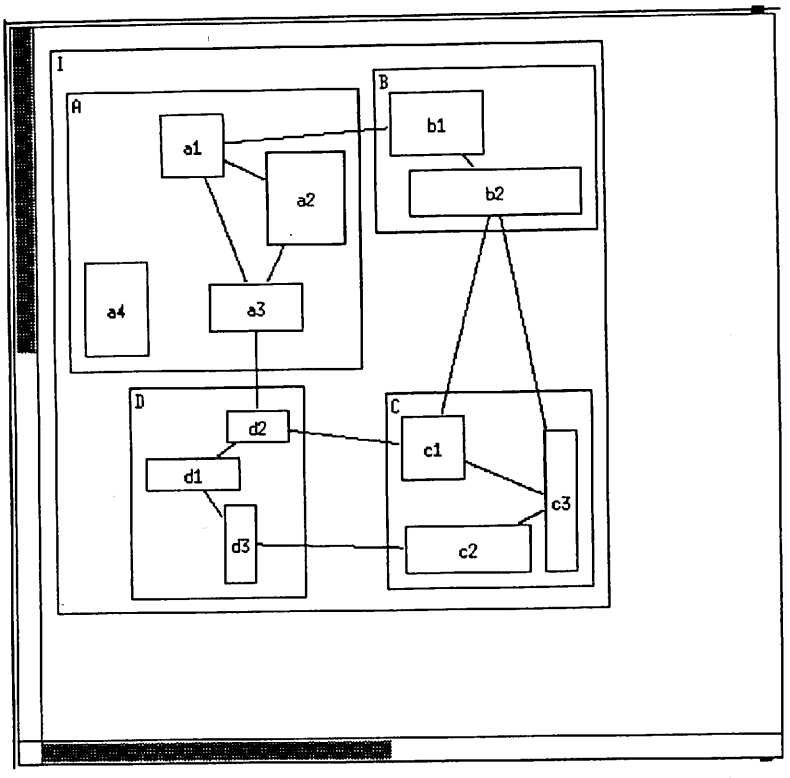


Figure 23: An example of dynamic changing a subgraph layout

since they are not changed at all. When the subgraph A is rearranged, its parent I may be changed so that it needs to be rearranged. In this way, dynamic graph drawings can be supported.

5.3 D-ABDUCTOR

D-ABDUCTOR is a diagram based idea organizer[23, 24, 1]. This system provides an automatic layout facility of compound graphs and perspective mappings. Figure 24 shows a call-graph of a program drawn by D-ABDUCTOR. It is so large that all nodes cannot be drawn in detail. When we are working with D-ABDUCTOR, if a perspective mapping is active, we can specify viewpoints (or view areas) interactively. Figure 25 shows a representation of the same diagram shown in Figure 24 by BFM with a view area (i.e., node). Additionally, Figure 26 shows the same diagram with two view areas. These figures illustrate that BFM preserves the mental map of the orthogonal ordering model while showing viewpoint areas in detail.

6 Conclusions

The literature on layout creation is quite large compared to that on layout adjustments, despite a relatively high demand for layout adjustment techniques in interactive visualization systems. This paper describes some initial results of research which aims to identify and investigate issues involved with layout adjustment.

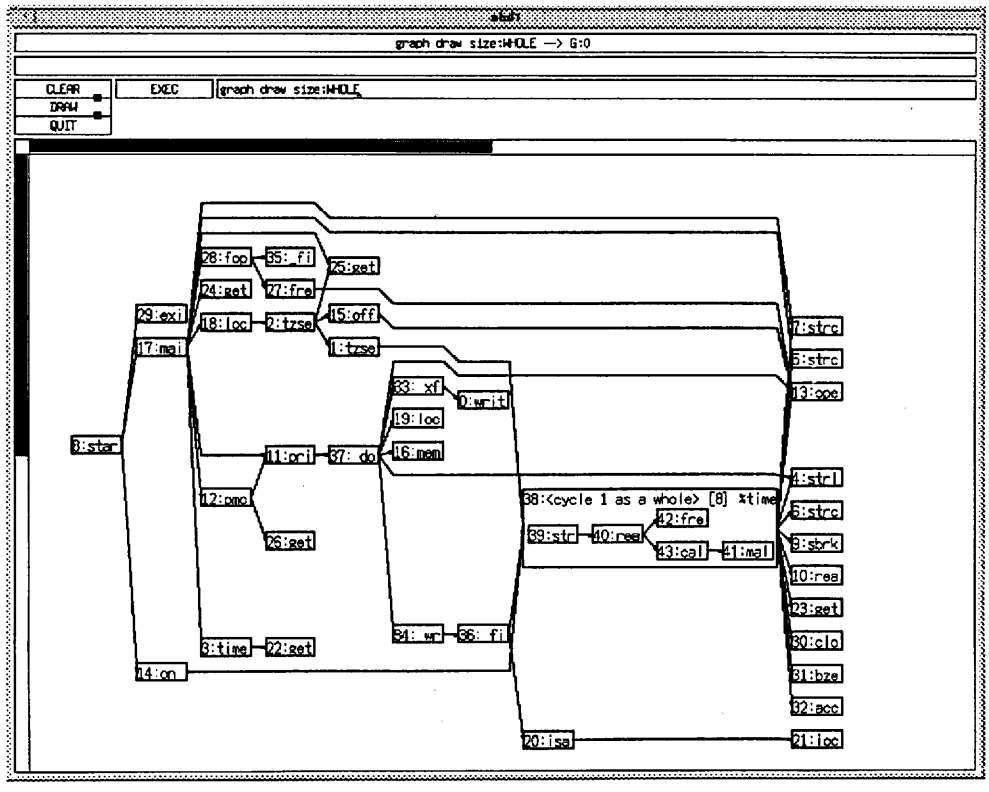


Figure 24: A call-graph of a program

The most critical issue is that of preserving the mental map of a diagram. We have proposed three models of the mental map: orthogonal ordering, proximity graphs, and topology.

Our investigation has covered two layout adjustment problems: that of ensuring node disjointness, and that of providing whole and detailed views. In each case, we are able to present methods which preserve orthogonal ordering and (to some extent) topology. However, we feel that the preservation of topology and proximity graphs needs significant further work. In particular, it would be interesting to find techniques for the node disjointness problem which provably preserve some classes of proximity graphs.

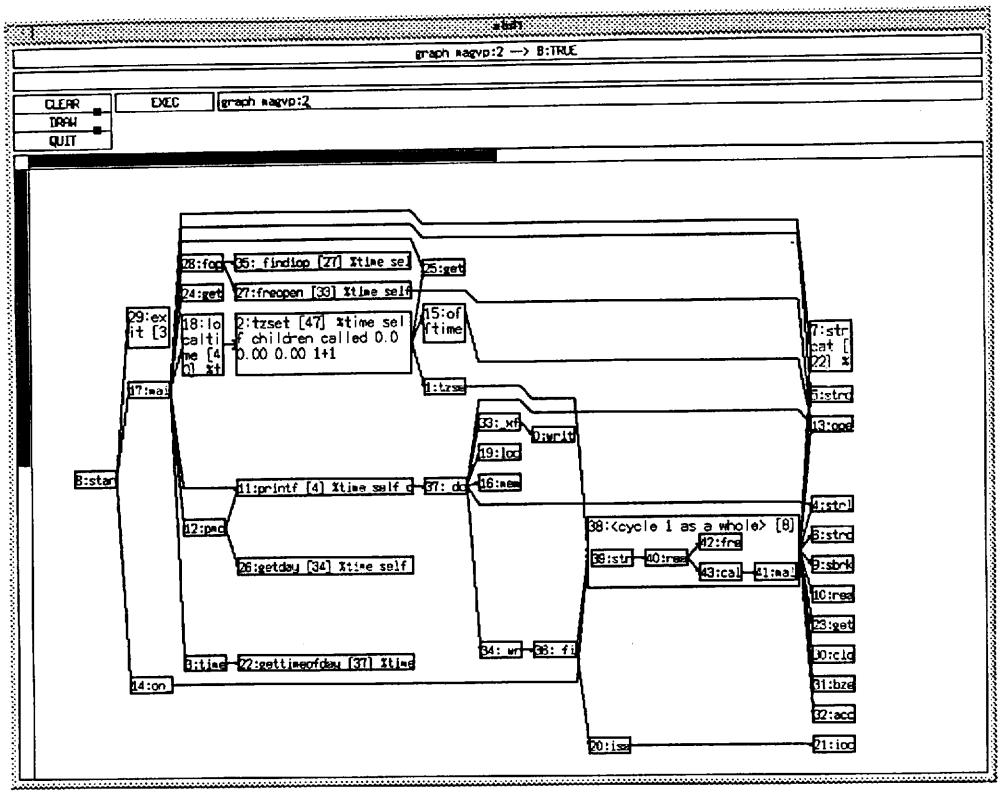


Figure 25: A call-graph represented in the biform mapping with a view area (The node labeled "2" is the view area.)

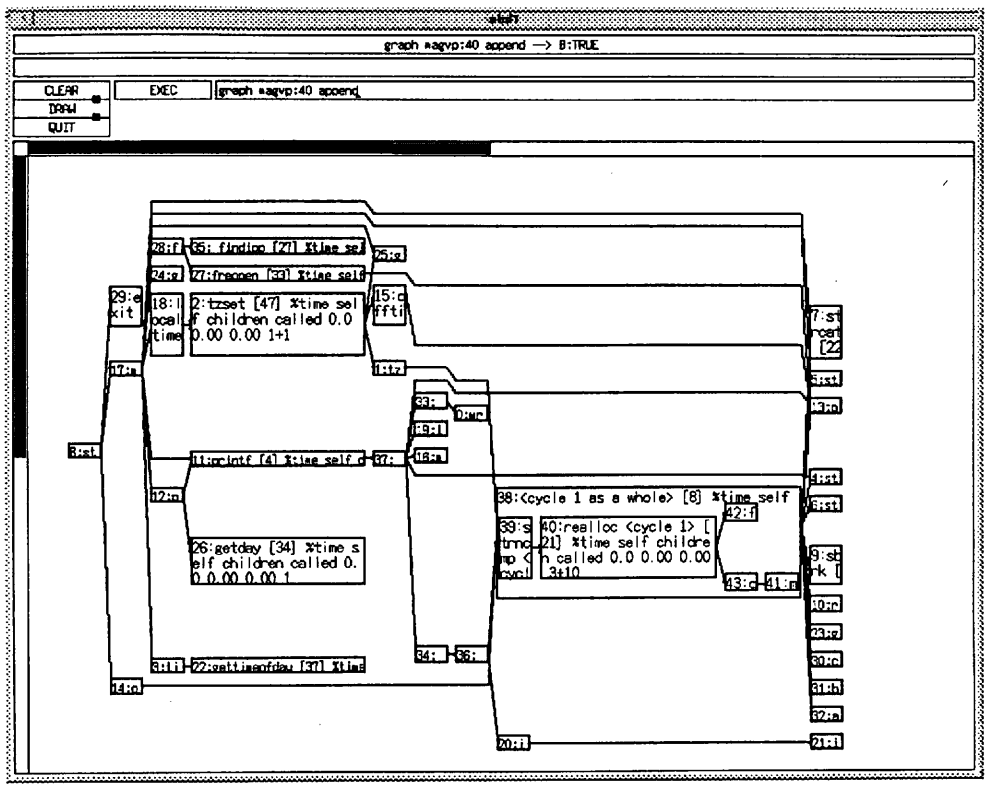


Figure 26: A call-graph represented in the biform mapping with two view areas (The nodes labeled "2" and "40" are the view areas.)

References

[1] Kazuo Misue and Kozo Sugiyama. Support of human thinking processes with D-ABDUCTOR. Research Report ISIS-RR-94-1E, Instutute for Social Information Science, Fujitsu Laboratories Ltd., 1994.

[2] M. Himsolt. Graphed: An interactive graph editor. In *Proc. STACS 89*, volume 349 of *Lecture Notes in Computer Science*, pages 532–533, Berlin, 1989. Springer-Verlag.

[3] G. Di Battista, G. Liotta, M. Strani, and F. Vargiu. Diagram server. In *Advanced Visual Interfaces (Proceedings of AVI 92)*, volume 36 of *World Scientific Series in Computer Science*, pages 415–417, 1992.

[4] G. Di Battista, A. Giammarco, G. Santucci, and R. Tamassia. The architecture of diagram server. In *Proc. IEEE Workshop on Visual Languages (VL'90)*, pages 60–65, 1990.

[5] R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-18(1):61–79, 1988.

[6] G. Di Battista, P. Eades, and R. Tamassia. Algorithms for automatic graph drawing: An annotated bibliography. Technical report, Department of Computer Science, Brown University, 1993.

[7] T. Kamada. *On Visualization of Abstract Objects and Relations.* PhD thesis, Department of Information Science, University of Tokyo, 1988.

[8] T. Lin. *A General Schema for Diagrammatic User Interfaces.* PhD thesis, University of Newcastle, 1993.

[9] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[10] R.F. Cohen, G. Di Battista, R. Tamassia, I.G. Tollis, and P. Bertolazzi. A framework for dynamic graph drawing. In *Proc. ACM Symp. on Computational Geometry*, pages 261–270, 1992.

[11] G. Di Battista and R. Tamassia. Incremental planarity testing. In *Proc. 30th IEEE Symp. on Foundations of Computer Science*, pages 436–441, 1989.

[12] S. Moen. Drawing dynamic trees. *IEEE Software*, 7:21–8, 1990.

[13] G. Toussaint. A graph-theoretical primal sketch. In G. Toussaint, editor, *Computational Morphology*, pages 229 – 260. Elsevier, 1988.

[14] F.P. Preparata and M.I. Shamos. *Computational Geometry.* Springer-Verlag, Berlin, 1985.

[15] K. Lyons. Cluster busting in anchored graph drawing. In *Proceedings of the 1992 CAS Conference*, pages 7–16, 1992.

[16] R. Davidson and D. Harel. Drawing graphs nicely using simulated aneealing. Technical report, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, 1989.

[17] Wei Lai. *Building Interactive Diagram Applications.* PhD thesis, University of Newcsastle, 1993.

[18] Kazuo Misue and Kozo Sugiyama. Multi-viewpoint perspective display methods: Formulation and application to compound graphs. In *Proc. of the Fourth International Conference on Human-Computer Interaction, (HCI International '91), volume 1*, pages 834–838, 1991.

[19] George W. Furnas. Generalized fisheye views. In *CHI'86 Proceedings*, pages 16–23, 1986.

[20] Y. K. Leung and M. D. Apperley. A taxonomy of distortion-oriented techniques for graphical data presentation. In *Proc. of the Fifth International Conference on Human-Computer Interaction, (HCI International '93), vol. 2*, pages 104–109, 1993.

[21] Hajime Ohiwa, Kazuhisa Kawai, and Masanobu Koyama. Idea processor and the KJ method. *Journal of Information Processing*, 31(1):44–48, 1990.

[22] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Software – Practice and Experience*, 21(11):1129–1164, 1991. Also as Technical Report UIUCDCS-R-90-1609, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, 1990.

[23] Kazuo Misue and Kozo Sugiyama. An overview of diagram based idea organizer: D-ABDUCTOR. Research Report IIAS-RR-93-3E, International Instutute for Advanced Study of Social Information Science, Fujitsu Laboratories Ltd., 1993.

[24] Kazuo Misue. D-ABDUCTOR 2.0 user manual. Research Report IIAS-RR-93-9E, International Instutute for Advanced Study of Social Information Science, Fujitsu Laboratories Ltd., 1993.