Generating Customized Layouts

Xiaobo Wang and Isao Miyamoto

Information and Computer Sciences Department
University of Hawaii, Honolulu, HI 96922, USA

Abstract. A good layout tool should be able to generate customized layouts according to different requirements given by the user or applications. To achieve this goal, existing layout techniques should be enhanced and integrated to take their advantages while compensating their disadvantages. This paper presents three layout techniques based on the force-directed placement approach, including a revised force-directed placement to draw graphs with vertices of nontrivial sizes, a divide-and-conquer approach to generate structured layouts, and an integrated approach to support constraints. The combination of the three techniques significantly improves the layout ability of the force-directed placement. They can be used to generate customized layouts that reflect semantics, preference, or principles of perceptual psychology.

1 Introduction

Graphs are widely used to represent relational problems in software systems. An important task for a graph-based application is generating the layouts of the graphs. Manual layout is time-consuming and error-prone. Automating the layout task consequently has received much attention in recent years [1].

A classical requirement on an automatic layout method is that the layout generated by the method should be syntactically valid and satisfy the aesthetic criteria, e.g., the layout should have no overlapping vertices and a small number of edge-crossings. Secondly, the layout should satisfy constraints derived from preference, semantics, or principles of perceptual psychology [3, 9]. This ensures that the layout conveys the correct message to the user. The third requirement is that the response time of the layout method should be reasonable for interactive application.

In the algorithmic approach, the layout of the graph is generated by optimizing the aesthetics of the graph. The algorithmic approach is computationally efficient and is very successful in generating layouts that are aesthetically pleasant to the eye. But the algorithmic approach draws the graph according to a set of pre-defined criteria. Most algorithmic methods do not support constraints and can not generate customized layout to reflect the semantics of the graph or the preference of the user. In the declarative approach, the layout of the graph is generated by searching a solution of a set of constraints. The power of constraints makes the declarative approach well-suited to express semantics or preference in the drawing of the graph. But it is difficult to specify global criteria such as aesthetics with constraints. It is also computationally inefficient to solve a large

number of constraints. The drawbacks of the algorithmic and the declarative approaches make them inadequate to be adopted by current applications [10].

The work of Eades and Lin [10] has shown that an integrated approach can take the advantages of both algorithmic and declarative approaches while compensating their disadvantages. In this paper, we focus on the *force-directed placement* [6] and discuss how to enhance the algorithm to satisfy the requirements described above. We present a layout tool called LYCA. The tool employs the force-directed placement to improve aesthetics of undirected graphs. A constraint solver is integrated with the layout algorithm to satisfy constraints. LYCA has several features:

- The force-directed placement in [6] is modified to draw graphs that contain vertices of different sizes. The improved algorithm can distribute vertices evenly if sizes of vertices are considered. Overlaps between large vertices are eliminated.
- A divide-and-conquer approach is introduced to generate structured layouts that reflect zones, proximity, and symmetries of subgraphs.
- The constraint solver and the layout algorithm cooperate to resolve the blocking problem caused by the competition between the constraint solver and the layout algorithm. This improves the layout quality.

With those functions, LYCA can generate layouts that are nice-looking as well as functional.

The organization of this paper is the following. Section 2 briefly explains Fruchterman's algorithm and discusses how to improve the algorithm to draw graphs with vertices of different sizes. Section 3 presents the divide-and-conquer approach. Section 4 describes the integration of the constraint solver and the force-directed placement. The last section gives discussion and concluding remarks.

2 Drawing Graphs with Large Vertices

In the first half of this section, we explain the force-directed placement algorithm [6]. In the second half of the section, we discuss how to modify the algorithm to draw graphs with vertices of different sizes.

2.1 Force-directed Placement

LYCA uses the force-directed placement [6] to improve aesthetics of undirected graphs. The force-directed placement is a variation of the well-known *spring embedding* algorithm [3, 4, 8, 13]. The algorithm draws graphs by applying an analogy from a physical process. Briefly, vertices are represented as atomic particles that exert forces upon each other. All vertices repel each other with repulsive forces. Neighbor vertices attract each other with attractive forces. Given an initial layout, the vertices are moved by the forces until they reach a stable state, which is returned as the final layout.

In [6], the strengths of forces between vertices are defined as:

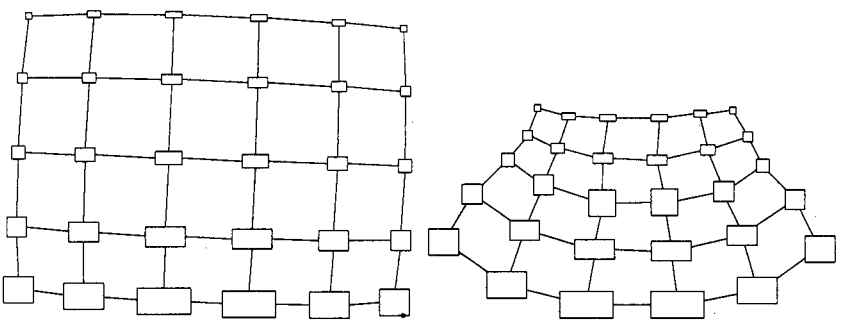
$$F_r = -k^2/d$$

$$F_a = d^2/k$$

where F_a is the attractive force, F_r is the repulsive force, d is the distance between a pair of vertices, k is the optimal distance between vertices.

2.2 The Revised Force-directed Placement

In the algorithm of [6], k represents the optimal distance between the centers of vertices. When drawing graphs which contain vertices of different sizes, k must be increased by the size of the largest vertex to avoid overlaps between vertices. This may yield layout with large area and uneven distribution of vertices if we consider sizes of vertices in the distribution, as shown in Figure 1 (a).



(a) layout of the original algorithm (b) layout of the revised algorithm

Fig. 1. Example layouts of graph with vertices of different sizes

To overcome the problem, we modify the forces between a pair of vertices v and w as follows:

$$F_a = \begin{cases} 0 & \text{if } w, v \text{ overlap} \\ \frac{d_{out}^2}{k'+d_{in}} & \text{otherwise} \end{cases}$$

$$F_r = \begin{cases} C\frac{k'^2}{d} & \text{if } w, v \text{ overlap} \\ \frac{k'^2}{d} & \text{otherwise} \end{cases}$$

where F_a is the attractive force between v and w, F_r is the repulsive force between v and w, d is the distance between the centers of v and w, d_{out} is the

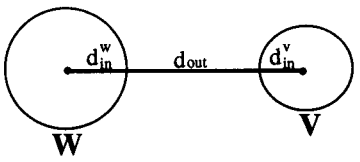


Fig. 2. Distance between vertices with areas

distance between the boundaries of v and w (see Figure 2), $d_{in} = d - d_{out}$, k' is the optimal distance between vertices, and C is a constant.

In the revised algorithm, the repulsive and attractive forces between a pair of vertices cancel each other when d_{out} between the vertices is equal to k'. Therefore, k' represents the optimal distance between the boundaries of vertices. Because the revised algorithm adjusts distances between vertices according to the actual sizes of the vertices, it distributes vertices evenly into a compact area. Overlaps between vertices are effectively eliminated since the attractive force between a pair of vertices becomes zero while the repulsive force is increased when the vertices overlap. An example layout created by the revised force-directed placement is shown in Figure 1 (b).

3 Generating Structured Layouts

Kosak el at. [9] pointed out that an important property of a layout is its perceptual organization. It was found that layouts that are organized according to the principles of perceptual psychology are easy to understand, while layouts that violate those principles are likely misleading [9].

We use a divide-and-conquer approach to generate visually structured layouts and display certain *Visual Organization Features* defined in [9], including zones, proximity, and symmetries of subgraphs.

3.1 Basic Notations

We first explain necessary concepts before presenting the divide-and-conquer algorithm.

Given a graph $G = (V, E)$, a *partition* P splits G into disjoint subgraphs, $P = \{G_1, \ldots, G_n\}$, such that

$$\begin{aligned} &G_i = (V_i, E_i) \\ &\textstyle\bigcup_{i=1}^{n} V_i = V \\ &V_i \cap V_j = \emptyset \text{ for } i \neq j \\ &E_i = \{(v, w) \in E | v, w \in V_i\} \end{aligned}$$

P also divides the edges of G into *intra-edges* and *inter-edges*. Intra-edges are edges between vertices in the same subgraph:

$$E_{intra} = \bigcup_{i=1}^{n} E_i$$

Inter-edges are edges between vertices in different subgraphs:

$$E_{inter} = E - E_{intra}$$

An undirected graph G_{meta} called a *meta-graph* is constructed by collapsing subgraphs of G into *meta-vertices* and transforming inter-edges of G into *meta-edges*. A layout of G_{meta} is called a *meta-layout* of G. A meta-layout can be obtained by the force-directed placement where the dimensions and center of each meta-vertex are set to the dimensions and center of the underlying subgraph, respectively.

Forces in the force-directed placement [6] are also divide into two categories: a force between a pair of vertices in the same subgraph is called an *intra-force*, a force between a pair of vertices in different subgraphs is called an *inter-force*. For a meta layout constructed from a partition P and a layout obtained by the original force-directed placement [6], the improved force-directed placement described in the previous section is used to calculate forces between meta-vertices in the meta-layout. The net force on a meta-vertex is defined as the *meta-force* on all vertices contained by the subgraph that is represented by the meta-vertex.

In Figure 3, a layout and a partition are given on the left side, the corresponding meta-layout is shown on the right side. The intra-force on vertex C is the sum of the forces between C, A and C, B, the inter-force on C is the sum of forces between C and vertices in subgraphs S_2 and S_3, the meta-force on C is the net force on the meta vertex $S1$ in the meta-layout on the right side.

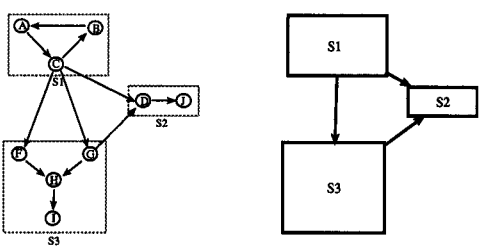


Fig. 3. Meta-graph and meta-layout

3.2 The Divide-and-conquer Approach

A divide-and-conquer approach draws a graph in three steps: 1) partition a graph into subgraphs, 2) draw subgraphs, 3) compose subgraph layouts together to form the resulting layout. A difficulty in divide-and-conquer layout is how to position inter-edges. If inter-edges are ignored in subgraph layouts, the resulting layout may have long-edges and edge-crossings. On the other hand, considering inter-edges in subgraph layouts leads to a circular dependency problem: inter-edges depend on subgraph layouts which in turn depend on inter-edges. In LYCA, this problem is solved as follows.

The divide-and-conquer approach uses a composite force to position a vertex:

$$F_{comp} = F_{intra} + S(t)F_{inter} + (1 - S(t))F_{meta}$$

where F_{comp} is the net force on a vertex, F_{intra} is the intra-force on a vertex, F_{inter} is the inter-force on a vertex, and F_{meta} is the meta force on a vertex. $S(t) \in [0, 1]$ is a function of layout time t such that $S(t)$ decreases as t increases after a threshold t' and reaches 0 at another threshold t'' $(> t')$.

The layout process consists of three phases. Between time 0 and time t', $S(t) = 1$, $F_{comp} = F_{inter} + F_{intra}$. The force-directed placement [6] is used to generate a layout with uniform edges and a small number of edge-crossings, as shown in Figure 4 (a). In the phase between time t' and t'', $S(t)$ decreases. This reduces the strengths of inter-forces. Meanwhile, the strengths of meta-forces are increased. At the time threshold t'', $S(t)$ reaches 0, $F_{comp} = F_{intra} + F_{meta}$. Since meta forces do not change the relative positions of vertices in a subgraph, vertices in subgraphs are positioned by intra-forces like in divide-and-conquer layout.

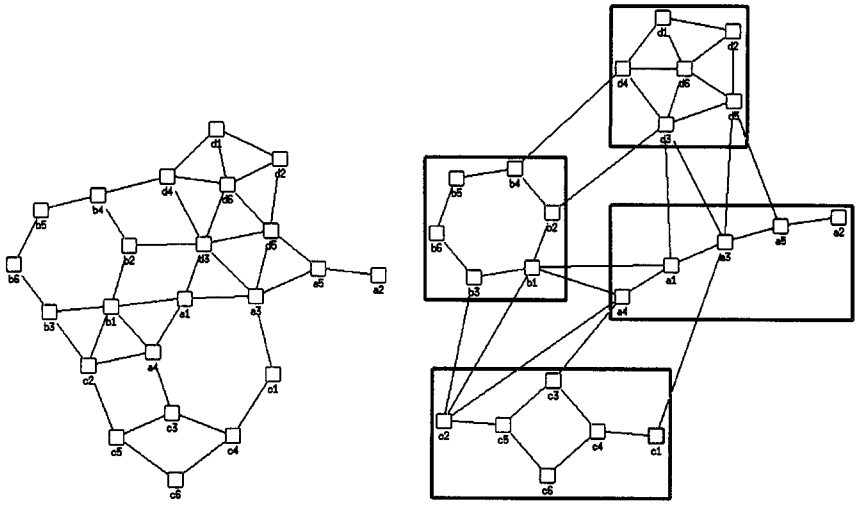
The divide-and-conquer approach displays zones of subgraphs because the meta-forces eliminate overlaps between subgraphs in the last phase of the layout process. Proximity can be reflected if vertices in subgraphs are placed closely and subgraphs are placed sparsely. This can be achieved by choosing a small optimal distance between vertices and a large optimal distance between meta-vertices. A problem of the force-directed placement is that it may not display symmetries of subgraphs if the entire graph is not symmetric. This problem is resolved in the divide-and-conquer approach since inter-forces are masked after the time threshold t''. In addition, because the resulting layout is evolved from the layout generated at time t' which has uniform edges and small number of edge-crossings, long edges and edge-crossings are avoided in the resulting layout. A structured layout created by the divide-and-conquer approach is shown in Figure 4 (b).

4 Integration of Constraint Solver and Layout Algorithm

A major limitation of the force-directed placement is that it does not support constraints. LYCA's solution to the problem is integrating the layout algorithm with a constraint solver. In this section, we focus on the issue of how to integrate the solver with the layout algorithm to ensure layout quality.

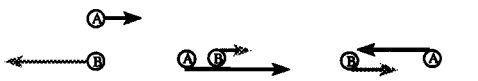
4.1 The Integrated Approach

LYCA takes an integrated approach to support constraints. A constraint solver is employed to solve three kinds of constraints: 1) an *absolute constraint* fixes a vertex at its current position, 2) a *relative constraint* constrains the position of a vertex in relation with others, 3) a *cluster constraint* clusters several vertices into a subgraph that can be processed as a whole. The solver uses a propagation style algorithm to satisfy constraints [15].



(a) Layout created at time t' (b) Resulting layout

Fig. 4. Structured Layout



(a) $A.x = B.x$ (b) $A.x < B.x$ (c) $A.x > B.x$

Fig. 5. Examples of barriers caused by solving constraints

During the layout process, the solver inputs coordinates of vertices from the layout algorithm and changes coordinates of constrained vertices to satisfy constraints. This integration may return poor layouts since solving constraints may make a vertex block others from reaching their optimal positions calculated by the force-directed placement. We define vertex B as a *barrier* for vertex A if solving a constraint between A and B prevents A from reaching its optimal position assigned by the force-directed placement. Figure 5 gives several examples of barriers. In the figure, the arrows on A and B indicate the movements of A and B assigned by the force-directed placement, respectively. Since the movements violate the constraint between A and B (shown below A and B), the solver has to change the position of either A or B to satisfy the constraint. If the solver chooses to change the position of A to satisfy the constraint, B becomes a barrier for A since it blocks A from being improved by the layout algorithm.

In Figure 5, if we move A and B together by the forces on them, we can reduce the total force on A and B. The solver will not change the position of A

to satisfy constraint because the constraint between A and B is not violated when A and B are moved together. Therefore, we can avoid barriers while improving the overall aesthetics of a graph with the principle of force-directed placement.

Based on the above observation, we introduce *rigid sticks* in the force-directed placement to represent constraints. If vertex v_1 becomes a barrier for vertex v_2, a rigid stick is introduced between v_1 and v_2 such that v_1 and v_2 must move together like one rigid object. The movements of v_1 and v_2 are determined by the weighted average of forces on them:

$$f = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2}$$

where f is the new force on v_1 and v_2, f_1 and f_2 are the original forces on v_1 and v_2, respectively, and w_1 and w_2 are weights of v_1 and v_2, respectively. During the layout process, the solver and the layout algorithm cooperate to remove barriers caused by constraints with following rules:

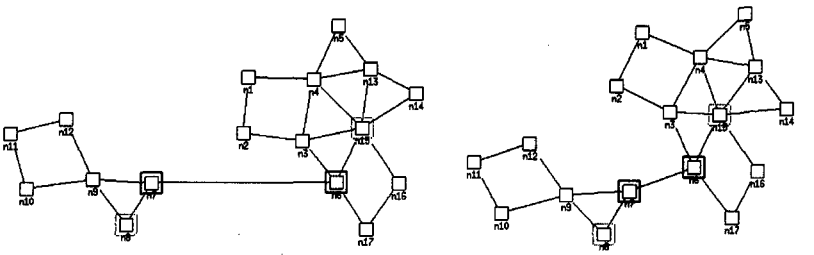
- If vertices v_1 and v_2 are aligned by an "*equal*" or a "*neighbor*" constraint, v_1 and v_2 are connected by rigid stick.
- If v_1 and v_2 are constrained by a "*less−than*" or "*greater−than*" constraint, a rigid stick is introduced between v_1 and v_2 only when one of the vertices becomes a barrier for the other one.
- If v is constrained as in the center of a set of vertices $v_1, \ldots, v_n$, the force on v is evenly distributed on $v_1, \ldots, v_n$.

With the cooperation between the solver and the layout algorithm, each layout iteration consists of four steps:

- Step 1. Calculate forces.
- Step 2. Introduce sticks and distribute forces.
- Step 3. Calculate new positions.
- Step 4. Satisfy constraints.

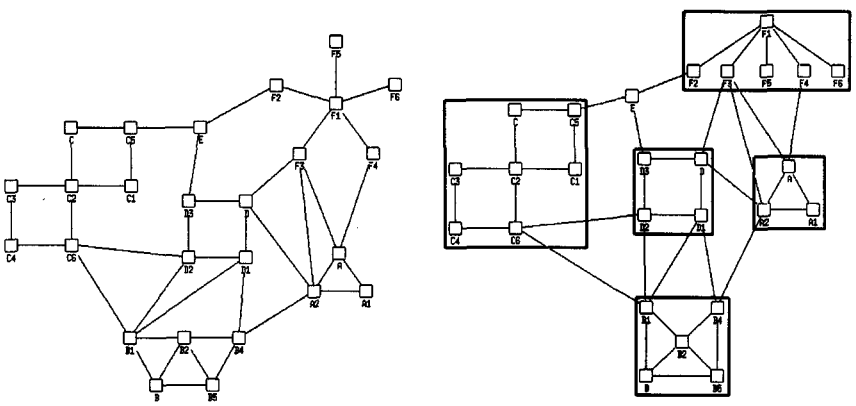
Step 1 and step 3 are performed by the layout algorithm. Step 2 and step 4 are performed by the solver. Our experiments showed that the heuristic works reasonably well to remove barriers and improve layout quality. As illustrated by the example in Figure 6, if the solver changes the positions of $n6$ and $n7$ to satisfy constraints "$n6.x > n15.x - 32$" and "$n7.x < n8.x + 32$", $n15$ and $n8$ become barriers for $n6$ and $n7$, respectively. If the barriers are not removed, the resulting layout has a long edge (see Figure 6 (a)). The problem is resolved when the solver and the layout algorithm cooperate to remove barriers, as shown in Figure 6 (b).

The integrated approach enhances the expressive capability of the force-directed placement significantly. Two example layouts generated with constraints are shown in Figure 7 and Figure 8. The constraints used to generated the layout in Figure 8 are given below:



(a) Without cooperation (b) With cooperation

Fig. 6. Remove barriers caused by solving constraints



(a) Example one (b) Example two

Fig. 7. Example layouts generated with constraints

```
abs(a4.x - a2.x) <= 120
abs(a1.x - a5.x) <= 70
a5.y = a1.y = a2.y - 64
a4.y = a3.y = a2.y
c2.y = c5.y = c4.y = c1.y
d3.y = d5.y
```

4.2 Analysis

LYCA uses a propagation style algorithm with linear time-complexity to solve constraints [15]. Internally constraints as represented as constraint graphs. Each edge in a constraint graph represents one constraint. To overcome barriers, the solver first marks the edges that are causing barriers as rigid sticks. The solver

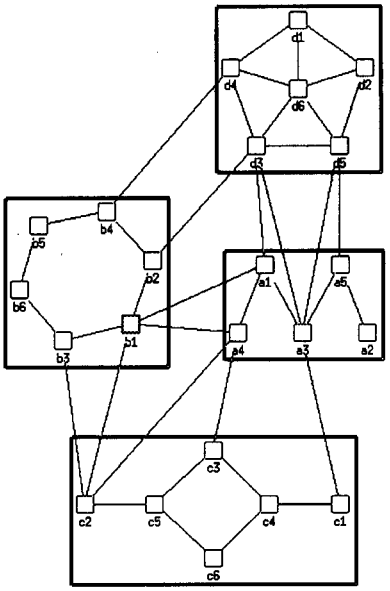


Fig. 8. Layout with constraints

then retrieves the vertices that are connected by rigid sticks and distributes forces on the retrieved vertices. The two steps also can be done in linear time. Therefore, the integrated approach is efficient. In our experiment, LYCA took less than 6 seconds on a Sparc 10 workstation to generate the layout shown in Figure 4. Since the current implementation of LYCA is not optimal, we expect that the time performance could be further improved.

5 Concluding Remarks

In the paper, we first discussed how to revise the force-directed placement in [6] to generate compact layouts for graphs with vertices of different sizes. A related work is the *force-scan* algorithm reported in [12]. The force-scan algorithm can keep the user's mental map on an existing layout while resolving overlaps between vertices in layout adjustment. In LYCA, the force-directed placement is mainly used for layout creation so it does not maintain the user's mental map on existing layouts.

LYCA's divide-and-conquer method draws a graph twice to create a structured layout: it first generates an aesthetically pleasing layout, it then transforms the first layout into a structured one. This may double the layout time. But the penalty is necessary to avoid long edges and edge-crossings in the resulting layout. Otherwise, manual modifications of the resulting layout or recursive adjustment of subgraph layouts have to be applied to ensure the layout quality in divide-and-conquer layout [7, 11]. Both LYCA and ANDD [3] use spring

algorithm to generate visually organized layouts. The difference between LYCA and ANDD is that ANDD processes constraints and aesthetics together, while LYCA processes constraints and aesthetics separately.

About layout quality, our initial experiments showed that the integrated approach works reasonably well. In general, LYCA can return good layouts when a few constraints or a lot of constraints are defined on the graph. In the former case, the layout algorithm dominates the layout process and the solver performs minor adjustment to satisfy constraints. In the latter case, the solver dominates the layout process and the layout algorithm "beautifies" the layout generated by the solver into a nice-looking one. However, if neither the solver nor the layout algorithm can dominate the layout process, poor layout may be returned and manual adjustment is needed. Obviously there is no trivial solution since the constrained optimization problem is computationally intractable. It will be interesting to investigate how to improve the layout quality with some known techniques, e.g., local temperature [5] or positioning vertices in certain order [14]. More experimental study is also needed to evaluate the performance of the integrated approach.

Acknowledge

I would like to thank Peter Eades, Kozo Sugiyama, and Joe Marks for their helps and suggestions. I would also like to thank anonymous referees for their useful comments.

References

1. G. D. Battista, P. Eades, R. Tamassia and I. G. Tollis, "Algorithms for drawing graphs: An annotated bibliography," Tech. Report, Computer Science Dept., Brown Univ., June, 1993.
2. R. Davidson and D. Harel, "Drawing graphs nicely using simulated annealing," *Technical Report CS89-13*, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, Israel, 1989.
3. E. Dengler, M. Friedell and J. Marks, "Constraint-driven diagram layout," *Proc. of Visual Language 93*, 1993.
4. P. Eades, "A heuristic for graph drawing," *Congress Numeratium*, Vol. 42, 1984.
5. A. Frick, A. Ludwing, and H. Mehldau, "A fast adaptive layout algorithm for undirected graphs," *Graph Drawing 94*, Princeton, New Jersey, October, 1994.
6. T. J. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software – Practice and Experience*, Vol. 21, No. 11, Nov. 1991, pp. 1129–1164.
7. T. R. Henry, "Interactive graph layout: The exploration of large graphs," Tech. Report 92–03, Computer Science Dept., Univ. of Arizona, Tucson, Arizona, 1992.
8. T. Kamada and S. Kawai, "An algorithm for drawing general undirected graphs," *Information Processing Letters*, Vol. 31, 1989.
9. C. Kosak, J. Marks and S. Shieber, "Automating the layout of network diagrams with specified visual organization," *IEEE Trans. on Syst., Man, and Cyb.*, Vol. 24., No. 3, March 1994.

10. T. Lin and P. Eades, "Integration of declarative and algorithmic approaches for layout creation," *Graph Drawing 94*, Princeton, New Jersey, October, 1994.
11. S. C. North, "Drawing ranked digraphs with recursive clusters," *Proc. of ALCOM Int'l Workshop on Graph Drawing*, Paris, France, Sept. 1993.
12. K. Misue, P. Eades, W. Lai and K. Sugiyama, "Layout adjustment and the mental map," Research Report ISIS-RR-94-6E, FUJITSU Lab. Ltd., Shizuoka, Japan, 1994.
13. K. Sugiyama and and K. Misue, "A simple and unified method for drawing graphs: magnetic-spring algorithm," *Graph Drawing 94*, Princeton, New Jersey, October, 1994.
14. D. Tunkelang, "An aesthetic layout algorithm for undirected graphs," Thesis for Master Degree, Computer Science and Engineering Department, M.I.T., 1992.
15. X. Wang, "Generating Customized Layouts Automatically," PhD thesis, Univ. of Hawaii at Manoa, August, 1995.