Information Processing Letters 31 (1989) 7–15
North-Holland

12 April 1989

AN ALGORITHM FOR DRAWING GENERAL UNDIRECTED GRAPHS

Tomihisa KAMADA and Satoru KAWAI

Department of Information Science, Faculty of Science, University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo, 113 Japan

Communicated by E.C.R. Hehner
Received 4 July 1988
Revised 13 November 1988

Keywords: Graph, network structure, layout, drawing algorithm

1. Introduction

Graphs (networks) are very common data structures which are handled in computers. Diagrams are widely used to represent the graph structures visually in many information systems. In order to automatically draw the diagrams which are, for example, state graphs, data-flow graphs, Petri nets, and entity-relationship diagrams, basic graph drawing algorithms are required.

The state of the art in automatic drawing is surveyed comprehensively in [7,19]. There have been only a few algorithms for general undirected graphs. This paper presents a simple but successful algorithm for drawing undirected graphs and weighted graphs. The basic idea of our algorithm is as follows. We regard the desirable "geometric" (Euclidean) distance between two vertices in the drawing as the "graph theoretic" distance between them in the corresponding graph. We introduce a virtual dynamic system in which every two vertices are connected by a "spring" of such desirable length. Then, we regard the optimal layout of vertices as the state in which the total spring energy of the system is minimal.

The "spring" idea for drawing general graphs was introduced in [6], and similar methods were used for drawing planar graphs with fixed boundary [2,20]. This paper brings a new significant result in graph drawing based on the spring model.

2. Graph drawing problem

We will clarify the graph drawing problem we treat in this paper. There are really a lot of ways to visualize graph structures. In some methods, the positions of vertices are restricted, e.g., they are placed on grid points [1,19], concentric circles [4], or parallel lines [14,18]. Edges can be drawn as straight lines, polygonal lines, or curves. We treat here the drawings in which the positions of vertices are not restricted and edges are drawn as straight lines. So our purpose is to determine the positions of vertices for a given graph G. (G is expressed by the set of vertices V and the set of edges E). And we assume that a given graph is connected. The picture of a disconnected graph is obtained by drawing its connected components separately. A graph can be decomposed into connected components in running time $O(|V|+|E|)$ [15].

First of all we must discuss the fundamental conditions of the general view of a graph. The graph structure encompasses so many kinds of structures, from trees to complete graphs, that it is difficult to find out the common criteria of nice drawings. However, the following two requirements in drawing graphs

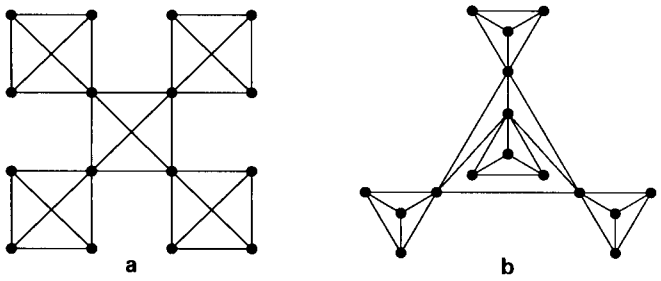

Fig. 1. Symmetric and planar drawings of a graph.

are commonly admitted [4,21]. One is to reduce the number of edge crossings, and the other is to distribute the vertices and edges uniformly. The reduction of edge crossings can, however, be shown not to be a good criterion as follows. Pictures with minimal number of edge crossings are not always the best. For example, Fig. 1 shows two different drawings of a 16-vertex graph. If they are not interested in planarity, almost all people think Fig. 1(a) is better for the representation of internal structure though it has five edge crossings. Figure 1(b) sacrifices symmetry for the reduction of edge crossings. Symmetric structures should be drawn as symmetric pictures because symmetry is a valuable characteristic of structures. Some algorithms have really tried to draw graphs symmetrically [6,11]. This example suggests that the total balance of layout is as important as or even more important than the reduction of edge crossings for human understanding. The latter condition (uniformity) does imply the total balance. Accordingly, we now pay attention to how judging the total balance of a layout. In our model, the total balance condition is formulated as the square summation of the differences between desirable distances and real ones for all pairs of vertices.

3. Graph drawing algorithm

3.1. Spring model

We introduce a dynamic system in which n $(= |V|)$ particles are mutually connected by springs. Let $p_1, p_2, \ldots, p_n$ be the particles in a plane corresponding to the vertices $v_1, v_2, \ldots, v_n \in V$ respectively. We relate the balanced layout of vertices to the dynamically balanced spring system. As a result, the degree of imbalance can be formulated as the total energy of springs, i.e.,

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \tfrac{1}{2} k_{ij} \left(|p_i - p_j| - l_{ij} \right)^2. \tag{1}$$

Pleasing layouts can be obtained by decreasing E, then the best layout is the state with minimum E in our model. The original length l_{ij} of the spring between p_i and p_j corresponds to the desirable length between them in the drawing, and is determined as follows. The distance d_{ij} between two vertices v_i and v_j in a graph is defined as the length of the shortest paths between v_i and v_j [3]. Then the length l_{ij} is defined as

$$l_{ij} = L \times d_{ij} \tag{2}$$

where L is the desirable length of a single edge in the display plane. When the display space is restricted, it

is a good way to determine L depending on the diameter (i.e., the distance between the farthest pair [3]) of a given graph. That is,

$$L = L_0 / \max_{i<j} d_{ij} \tag{3}$$

where L_0 is the length of a side of display square area.

The parameter k_{ij} is the strength of the spring between p_i and p_j, and is determined as follows. The expression (1) can be regarded as the square summation of the differences between desirable distances and real ones for all pairs of particles. From this point of view, the differences per unit length is better to be used in (1). Then, we define k_{ij} as

$$k_{ij} = K/d_{ij}^2 \tag{4}$$

where K is a constant. The parameters l_{ij} and k_{ij} are symmetric, i.e., $l_{ij} = l_{ji}$ and $k_{ij} = k_{ji}$ $(i \neq j)$.

Here we analyze the properties of our spring model briefly. The density of particles does not become large, because every two nodes are forced to keep certain distance by the tension of a spring. Note that symmetric graphs correspond to symmetric spring systems, which result in symmetric layouts by minimizing E.

3.2. *Local minimization of global energy*

We prepare some definitions for describing the algorithm and outline our method. The position of a particle in a plane is expressed by x and y coordinate values. Let $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ be the coordinate variables of particles $p_1, p_2, \ldots, p_n$ respectively. Then, the energy E defined as (1) is rewritten by using these $2n$ variables as follows.

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \tfrac{1}{2} k_{ij} \left\{ (x_i - x_j)^2 + (y_i - y_j)^2 + l_{ij}^2 - 2 l_{ij} \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \right\}. \tag{5}$$

Our purpose is to compute the values of these variables which minimize $E(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n)$. (Hereafter the parameters of the function E are omitted). It is, however, quite difficult to compute the minimum, so we instead compute a local minimum.

We present a method of computing a local minimum of E from a certain initial state based on the Newton–Raphson method [14]. The necessary condition of local minimum is as follows.

$$\frac{\partial E}{\partial x_m} = \frac{\partial E}{\partial y_m} = 0 \quad \text{for } 1 \leqslant m \leqslant n. \tag{6}$$

The state satisfying (6) corresponds to the dynamic state in which the forces of all springs are balanced. The partial derivatives of (5) by x_m and y_m are as follows.

$$\frac{\partial E}{\partial x_m} = \sum_{i \neq m} k_{mi} \left\{ (x_m - x_i) - \frac{l_{mi}(x_m - x_i)}{\left\{ (x_m - x_i)^2 + (y_m - y_i)^2 \right\}^{1/2}} \right\}, \tag{7}$$

$$\frac{\partial E}{\partial y_m} = \sum_{i \neq m} k_{mi} \left\{ (y_m - y_i) - \frac{l_{mi}(y_m - y_i)}{\left\{ (x_m - x_i)^2 + (y_m - y_i)^2 \right\}^{1/2}} \right\}. \tag{8}$$

We must solve these $2n$ simultaneous non-linear equations of (6). But they cannot be directly solved by using a $2n$-dimensional Newton–Raphson method, because they are not independent of one another.

Then, we adopt another way in which only one particle $p_m(x_m, y_m)$ is moved to its stable point at a time, freezing the other particles. That is, viewing E as a function of only x_m and y_m, we compute a local minimum of E by using a two-dimensional Newton–Raphson method. We can obtain a local minimum which satisfies (6) iterating this step. In each step, we choose the particle that has the largest value of Δ_m, which is defined as

$$\Delta_m = \sqrt{\left\{\frac{\partial E}{\partial x_m}\right\}^2 + \left\{\frac{\partial E}{\partial y_m}\right\}^2}. \tag{9}$$

Starting from $(x_m^{(0)}, y_m^{(0)})$ which is equal to the current position (x_m, y_m), the following step is iterated:

$$x_m^{(t+1)} = x_m^{(t)} + \delta x, \qquad y_m^{(t+1)} = y_m^{(t)} + \delta y \qquad \text{for } t = 0, 1, 2, \ldots \tag{10}$$

The unknowns δx and δy satisfy a pair of linear equations as follows:

$$\frac{\partial^2 E}{\partial x_m^2}\left(x_m^{(t)}, y_m^{(t)}\right)\delta x + \frac{\partial^2 E}{\partial x_m \partial y_m}\left(x_m^{(t)}, y_m^{(t)}\right)\delta y = -\frac{\partial E}{\partial x_m}\left(x_m^{(t)}, y_m^{(t)}\right), \tag{11}$$

$$\frac{\partial^2 E}{\partial y_m \partial x_m}\left(x_m^{(t)}, y_m^{(t)}\right)\delta x + \frac{\partial^2 E}{\partial y_m^2}\left(x_m^{(t)}, y_m^{(t)}\right)\delta y = -\frac{\partial E}{\partial y_m}\left(x_m^{(t)}, y_m^{(t)}\right). \tag{12}$$

The coefficients of above equations (11) and (12), which are the elements of Jacobian matrix, are computed from the partial derivatives of (7) and (8) by x_m and y_m as follows.

$$\frac{\partial^2 E}{\partial x_m^2} = \sum_{i \neq m} k_{mi}\left\{1 - \frac{l_{mi}(y_m - y_i)^2}{\left\{(x_m - x_i)^2 + (y_m - y_i)^2\right\}^{3/2}}\right\}, \tag{13}$$

$$\frac{\partial^2 E}{\partial x_m \partial y_m} = \sum_{i \neq m} k_{mi}\frac{l_{mi}(x_m - x_i)(y_m - y_i)}{\left\{(x_m - x_i)^2 + (y_m - y_i)^2\right\}^{3/2}}, \tag{14}$$

$$\frac{\partial^2 E}{\partial y_m \partial x_m} = \sum_{i \neq m} k_{mi}\frac{l_{mi}(x_m - x_i)(y_m - y_i)}{\left\{(x_m - x_i)^2 + (y_m - y_i)^2\right\}^{3/2}}, \tag{15}$$

$$\frac{\partial^2 E}{\partial y_m^2} = \sum_{i \neq m} k_{mi}\left\{1 - \frac{l_{mi}(x_m - x_i)^2}{\left\{(x_m - x_i)^2 + (y_m - y_i)^2\right\}^{3/2}}\right\}. \tag{16}$$

The unknowns δx and δy can be computed from (11)–(16). The iteration (10) terminates when the value of Δ_m at $(x_m^{(t)}, y_m^{(t)})$ becomes small enough.

3.3. Algorithm

In our graph drawing algorithm, first the distance d_{ij} must be computed for all pairs of vertices in a given graph. For the present, we are using a simple shortest-path algorithm of Floyd [8]. Next l_{ij} and k_{ij} are computed for all pairs from d_{ij} by the use of (2), (3) and (4). Before starting the minimization process, the initial positions of particles must be determined. The experiments have shown that the initial positions do not have a great influence on the resultant pictures, except for the special cases: e.g., all particles lie on a single line. Now we are using a simple initialization by which the particles are placed on the nodes of the

regular n-polygon circumscribed by a circle whose diameter is L_0 (as for L_0, see (3)). After initializing the positions, the energy E is decreased, step by step, by moving a particle to a stable position.

The algorithm is summarized in a simple form as follows.

```
compute d_ij for 1 ≤ i ≠ j ≤ n;
compute l_ij for 1 ≤ i ≠ j ≤ n;
compute k_ij for 1 ≤ i ≠ j ≤ n;
initialize p_1, p_2, ..., p_n;
while (max_i Δ_i > ε){
   let p_m be the particle satisfying Δ_m = max_i Δ_i;
   while (Δ_m > ε){
      compute δx and δy by solving (11) and (12);
      x_m := x_m + δx;
      y_m := y_m + δy;
   }
}
```

Here we discuss the computational cost of above algorithm briefly. As for d_{ij}, $O(n^3)$ time is required preliminarily. (We can use instead more efficient algorithms [13,16] for large graphs.) Required computational time is mainly determined by the nested *while* loops. In the inner loop which is the Newton–Raphson iteration, $O(n)$ time is required to compute Δ_m and to compute δx and δy respectively at each step. And in the outer loop, $O(n)$ time is required to compute the maximum of Δ_i because updating each Δ_i $(i \neq m)$ after moving p_m can be performed in O(1) time by memorizing the old position of p_m. As a result, the time needed to terminate the *while* loops is $O(T \cdot n)$ where T is the total number of inner loops. It is

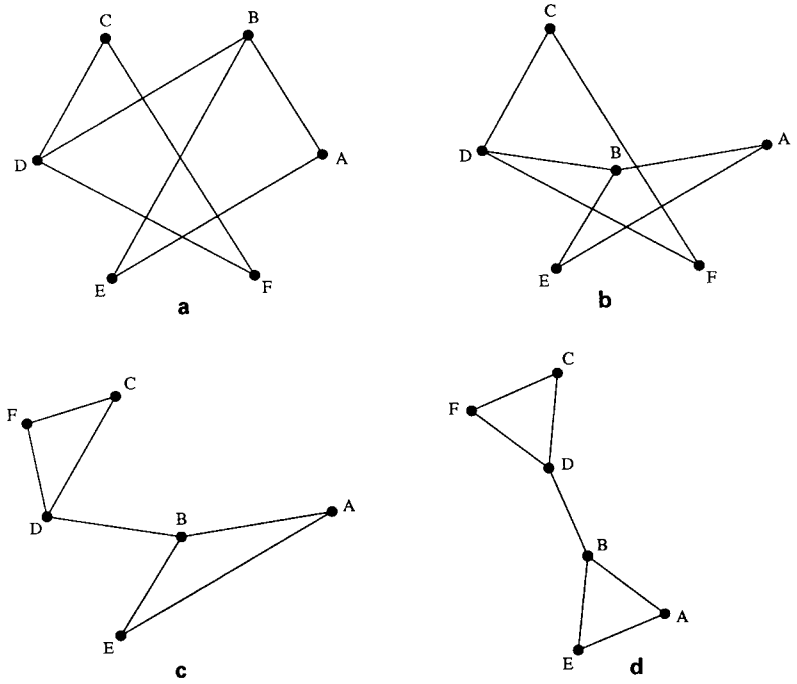

Fig. 2. The energy minimization process.

difficult to estimate T because T depends both on the given graphs, especially the number of vertices (n), and on the initial positions of vertices. However, lowering the convergence precision (ϵ) is an effective way to reduce T.

Figure 2 illustrates the process of minimizing E in the case of a 6-vertex graph. First the particle B is moved (Fig. 2(b)) and next the particle F is moved (Fig. 2(c)). The final state (Fig. 2(d)) is obtained after 21 moving steps. The algorithm proposed here does not guarantee to compute the true minimum of E. In order to prevent the energy from converging to a large local minimum, we have added a simple test to the algorithm [10].

4. Examples

4.1. Symmetric and asymmetric graphs

We show some pictures generated by the system implementing our graph drawing algorithm. Figure 3 shows four pictures of symmetric graphs. Figure 3(a) and 3(d) are the pictures of a regular hexahedron (cube) graph and regular dodecahedron graph respectively. They look as if they were the projected images of a cube and a dodecahedron. Figure 3(b) shows a picture of the graph which has the triangulated internal structure. Figure 3(c) shows a picture of the graph cited in Fig. 1. As these examples show, symmetry of

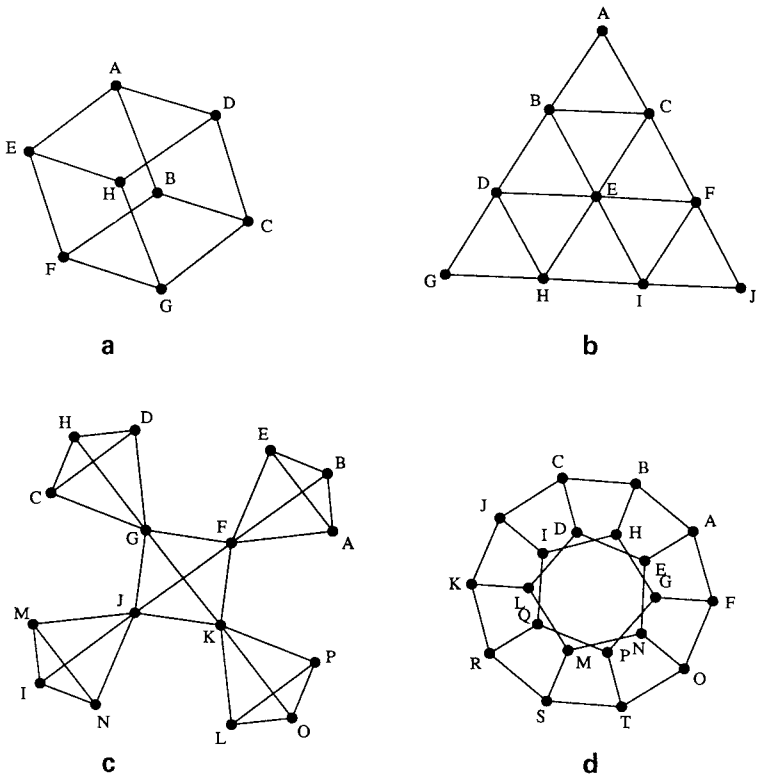

Fig. 3.

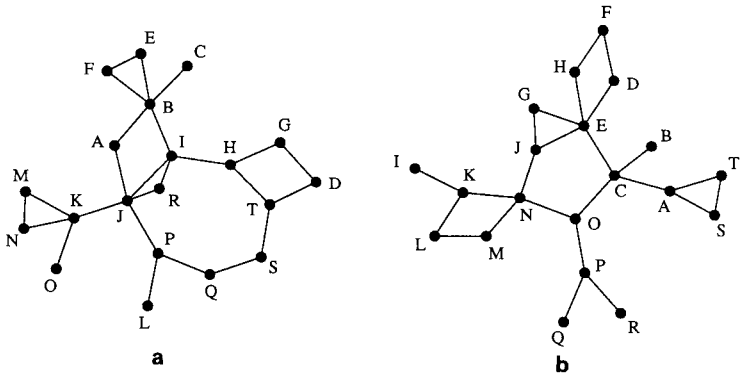

Fig. 4. Pictures of asymmetric graphs.

graphs is visualized pleasingly as symmetric pictures in our system. Figure 4 shows two pictures of asymmetric graphs. In these cases needless edge crossings are avoided completely.

The CPU time needed to compute a layout in these pictures is from 0.4 seconds (Fig. 3(a)) to 7.6 seconds (Fig. 3(d)) on a VAX 8600.

4.2. Isomorphic graphs

When the viewer wants to compare some graphs, it is highly required of the system to display isomorphic graphs as the same picture. Otherwise the viewer cannot make proper comparisons from the

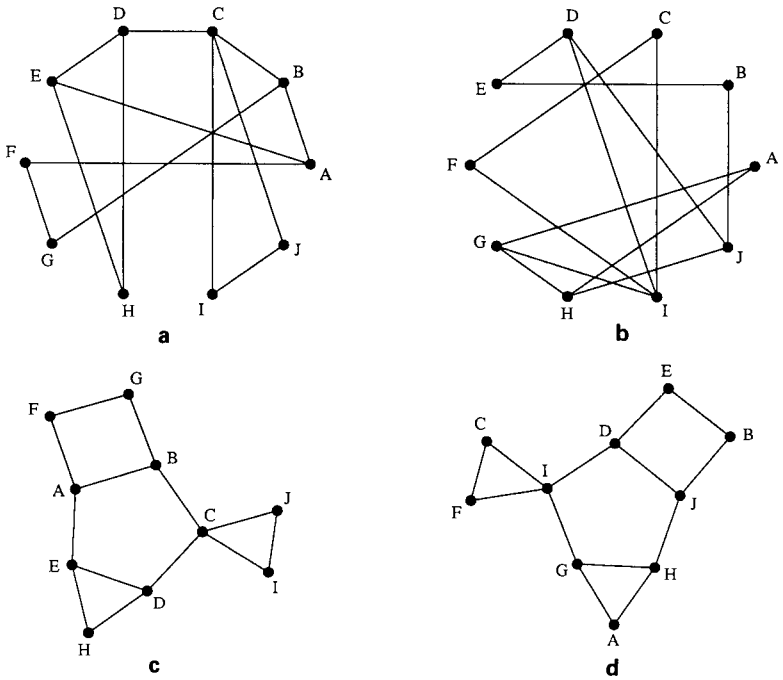

Fig. 5. Pictures of isomorphic graphs.

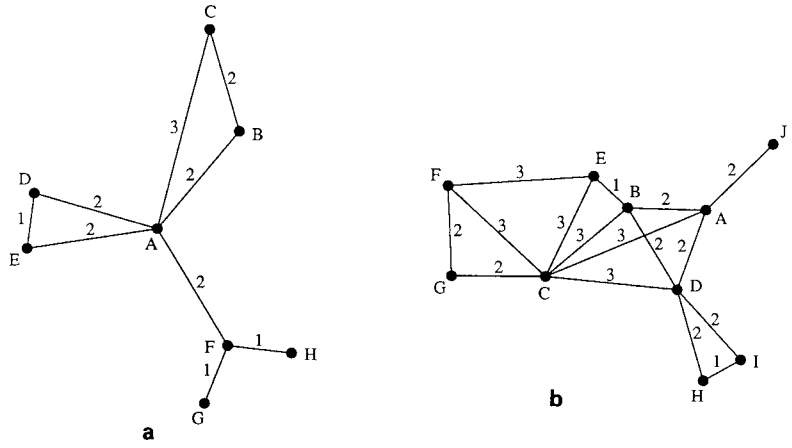

Fig. 6. Pictures of weighted graphs.

pictures. Therefore, graph drawing algorithms should meet this requirement of congruity or uniqueness of generated pictures [17]. The problem of isomorphism determination of general graphs itself is an intractable problem, and all known algorithms have a running time exponential on the number of vertices in the worst case [5,9,12].

In our system, isomorphic graphs are mostly drawn as the pictures one of which can be identified with the other by translating, rotating, and sometimes reflecting it. Figure 5 illustrates pictures of isomorphic graphs. Figure 5(a) and 5(b) show the picture of two initial states in our algorithm. It is difficult for the viewer to detect isomorphism from these pictures, while it is pretty easy to find out isomorphism from the respective resultant pictures in Fig. 5(c) and 5(d). The CPU time needed to compute both layouts in Fig. 5(c) and 5(d) is about 2 seconds on a VAX 8600. On the other hand, it takes about 18 minutes CPU time to solve this isomorphism problem combinatorially by a naive thorough search. This "drawing" approach can be said to be a good approximate method for graph isomorphism, though the pictures of isomorphic graphs cannot always be regarded as identical because the total spring energy does not always converge to the same minimum in our algorithm. The judgement of isomorphism from the resultant pictures could be done automatically by making use of the geometric attributes yielded by the drawing system. In addition, our algorithm brings a satisfactory result as for the "likeness" of graphs. A small difference between graphs can be found out easily from the pictures.

4.3. Weighted graphs

In order to model practical problems, weighted graphs in which "weights" are associated with edges are often used. It is naturally required to visualize weighted graphs by regarding weights as geometric distances in the drawing. We apply our graph drawing algorithm to weighted graphs straightforwardly. In weighted graphs, the distance between vertices is defined as the summation of weights. Figure 6 shows two pictures of weighted graphs generated by our system. The overall images regarding weights can be understood intuitively from these pictures.

5. Concluding remarks

We have proposed a method of drawing general undirected graphs for human understanding. It can be widely used in the systems which deal with network structures. Our idea is quite simple and intuitive; the

graph theoretic distance between vertices in a graph is related to the geometric distance between them in the drawing. As described above, our spring algorithm has many good properties; symmetric drawings, a relatively small number of edge crossings, and almost congruent drawings of isomorphic graphs. The precise proof of these properties is left as a challenging problem. And the study of a "constrained" version of our algorithm by which vertices are constrained, for instance, to lie on concentric circles or on parallel lines is underway.

Acknowledgments

We would like to thank Mr. Katuhiro Ota (University of Tokyo), who studies graph theory, for many helpful discussions about the problems in drawing graphs and our drawing algorithm. We are also grateful to the referee for useful suggestions.

References

[1] C. Batini, E. Nardelli and R. Tamassia, A layout algorithm for data flow diagrams, *IEEE Trans. Software Eng.* **SE-12** (1986) 538–546.

[2] B. Becker and G. Hotz, On the optimal layout of planar graphs with fixed boundary, *SIAM J. Comput.* **16** (1987) 946–972.

[3] M. Behzad, G. Chartrand, and L. Lesniak-Foster, *Graphs & Digraphs* (Prindle, Weber & Schmidt, Boston, MA, 1979).

[4] M. Carpano, Automatic display of hierarchized graphs for computer-aided decision analysis, *IEEE Trans. Syst., Man, Cybern.* **SMC-10** (1980) 705–715.

[5] D.G. Corneil and C.C. Gotlieb, An efficient algorithm for graph isomorphism, *J. ACM* **17** (1970) 51–64.

[6] P. Eades, A heuristics for graph drawing, *Congr. Numer.* **42** (1984) 149–160.

[7] P. Eades and R. Tamassia, Algorithms for drawing graphs: An annotated bibliography, Technical Rep. No. 82, Dep. of Comp. Sci., Univ. of Queensland, Australia, 1987.

[8] R.W. Floyd, Algorithm 97: shortest path, *Comm. ACM* **5** (1962) 345.

[9] J.E. Hopcroft and R.E. Tarjan, A V log V algorithm for isomorphism of triconnected of planar graphs, *J. Comput. Syst. Sci.* **7** (1973) 323–331.

[10] T. Kamada and S. Kawai, Automatic display of network structures for human understanding, Technical Rep. No. 88-7, Dep. Inf. Sci., Univ. of Tokyo, Japan, Feb. 1988.

[11] R. Lipton, S. North and J. Sandberg, A method for drawing graphs, *Proc. ACM Symposium on Computational Geometry*, 1985, pp. 153–160.

[12] H.B. Mittal, A fast backtrack algorithm for graph isomorphism, *Inform. Process. Lett.* **29** (1988) 105–110.

[13] A. Moffat and T. Takaoka, An all pairs shortest path algorithm with expected running time $O(n^2 \log n)$, *Proc. Conf. Found. Comp. Sci.*, 1985, pp. 101–105.

[14] L.A. Rowe, M. Davis, E. Messinger, C. Meyer, C. Spirakis and A. Tuan, A browser for directed graphs, *Software Pract. Exper.* **17** (1987) 61–76.

[15] R. Sedgewick, *Algorithms* (Addison-Wesley, Reading, MA, 1983).

[16] P.M. Spira, A new algorithm for finding all shortest paths in a graph of positive arcs in average time $O(n^2 \log^2 n)$, *SIAM J. Comput.* **2** (1973) 28–32.

[17] K. Sugiyama, Achieving uniqueness requirement in drawing digraphs: optimum code algorithm and hierarchy isomorphism, Research Rep. No. 58, IIAS-SIS, FUJITSU, Japan, 1985.

[18] K. Sugiyama, S. Tagawa and M. Toda, Methods for visual understanding of hierarchical system structures, *IEEE Trans. Syst. Man, Cybern.* **SMC-11** (1981) 109–125.

[19] R. Tamassia, G. Battista and C. Batini, "Automatic graph drawing and readability of diagrams", *IEEE Trans. Syst., Man, Cybern.* **SMC-18** (1988) 61–79.

[20] W.T. Tutte, How to draw a graph, *Proc. London Math. Soc.* **3** (1963) 743–768.

[21] J.N. Warfield, Crossing theory and hierarchy mapping, *IEEE Trans. Syst., Man, Cybern.* **SMC-7** (1977) 505–523.