Journal of Visual Languages and Computing (1998) 9, 623–645
Article No. vl980094

On-line Animated Visualization of Huge Graphs using a Modified Spring Algorithm

MAO LIN HUANG,* PETER EADES* AND JUNHU WANG†

** Department of Computer Science and Software Engineering, The University of Newcastle NSW 2308, Australia and †Computer Center, Hebei University of Economy & Trade, Shijiazhuang 050091, People's Republic of China*

Received 30 June 1998; accepted 7 September 1998

On-line graph drawing deals with huge graphs which are partially unknown. At any time, a tiny part of the graph is displayed on the screen. Examples include web graphs and graphs of links in distributed file systems. This paper discusses issues arising in the presentation of such graphs. The paper describes a system for dealing with web graphs using on-line graph drawing.

Keywords: graph drawing, information visualization, computer graphics.

1. Introduction

AS GRAPHICS WORKSTATIONS have now become common tools for software and information engineers, visualization of relational information has become an essential element of current software systems. Examples include web browsers, file system visualizers, CASE tools, database design systems, network design systems, visual programming interfaces and reverse engineering systems. The effectiveness of visualizing relational information is widely recognized.

Most systems use *graphs* to model relational structures: the entities are *nodes*, and the relationships are *edges* (sometimes called *links*). For example, most CASE tools use graphs to model the dependencies between modules in a large program. A module is represented as a node in a graph, and the dependency of one module *a* on module *b* is represented by an edge from *a* to *b*. These graphs are typically drawn as diagrams with text at the nodes and line segments joining the nodes as edges. As another example, the structure of the World Wide Web can be modeled as a *Web graph*: the nodes are html documents, and a link from one document to another is represented as a directed edge.

Relational models are useful only to the degree that the diagram effectively conveys information. A good diagram is worth a thousand words, but a poor diagram can be confusing and misleading. The central problem in creating good diagrams is designing algorithms to assign a location for each node and a route for each edge; this is the classical *graph drawing problem*. Since the advent of graphics workstations in the early 1980s, the graph drawing problem has been the subject of a great deal of research [1–3] .

1045-926X/98/060623+23 $30.00/0

As the amount of information that we want to visualize becomes larger and the relations become more complicated, classical graph drawing methods tend to be inadequate. Even in a small modern file system (say with a 2GB drive on a PC) there are hundreds of nodes and links. Web graphs are much larger; even a small organization such as a University has many thousands of Web documents. When the graph represents distributed data (such as the Web), the graph is not only huge but also it is partially unknown, and the visualization system displays only a tiny part of a huge unexplored graph.

Most existing information visualization systems have problems presenting very large graphs. The most common solution is to provide a *virtual* (*very large) page* to fit the layout of the whole graph, and then provide a small window and scroll bars to allow the user to navigate through the visualization. Some alternative techniques have also been proposed [4–7]. For example, *fish-eye* [4] views can keep a detailed picture of a part of a graph as well as the global context of the graph. The *hyperbolic browser* [7] technique performs fish-eye viewing with animated transitions to preserve the user's *mental map* [8] (see Figure 1). Three-dimensional methods, such as *cone trees* [5], seem to increase the density of information on the screen.

These techniques usually build a large *static* visualization of the graph, and then allow the user to navigate through the visualization. Since the amount of data that can be effectively displayed at one time is limited, these static techniques are unable to display the whole graph in detail at one time. To solve this problem, these techniques involve a mechanism to change the view (dynamic viewing); this allows the user to effectively view only at one time a small area of the whole visualization by changing the viewing area (with the virtual page and hyperbolic tree techniques), zoomed focus point (with the fish-eye and hyperbolic tree techniques), or viewpoint (with the 3D cone tree technique) of the visualization.

While these techniques (*static layout + dynamic viewing*) effectively deal with graphs of moderately large size (with hundreds or thousands of nodes), they do not handle the entire web, a huge and partially unknown graph (with millions or perhaps billions of nodes). The major problems may be outlined as below:

- All of these static visualization techniques pre-define the layout. In most cases, the whole huge graph is not known. The local node in a hypermedia system may know only a small part of the whole graph (say one web site). So it may be impossible to pre-compute the layout of the whole graph.
- Further, pre-computation of the overall geometrical structure (global context) of huge graphs is very computationally expensive. Most graph layout algorithms have super-linear time complexity, and in practice are too slow for interactive graphics if the number of nodes is larger than a few hundred. It is not necessary to build a global context, since the user always focuses on a small logical section of the graph.
- Pre-computation of the layout itself poses another problem. Since views are extracted from a pre-defined layout, changing views is a geometrical operation and not a logical operation. The user naturally thinks in terms of the logical relations in the application domain (for example, hyperlinks in the hypermedia), not in terms of the synthesized geometry of the layout; thus, logical navigation of the hyperlinks through the entire graph by using static techniques (fixed geometrical representations) is difficult.

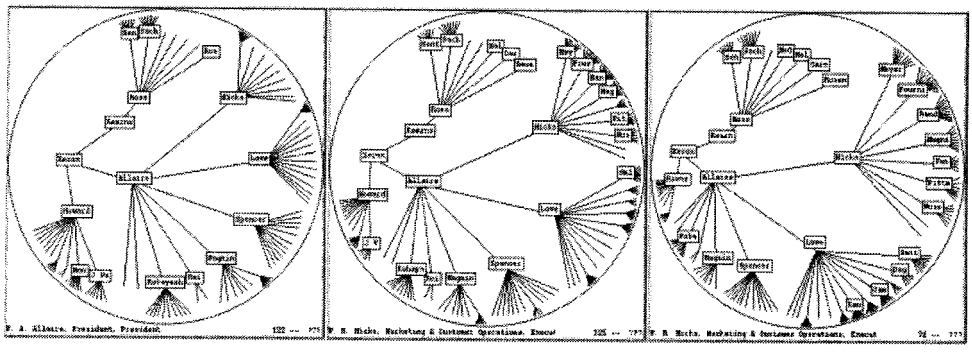


Figure 1. An example of hyperbolic tree viewing with animated transitions. It uses the 'static layout + dynamic viewing' technique (from Lamping *et al.* [7], with permission of Xerox PARC and Inxight Software)

Another way to deal with the problem of visualizing huge graphs is to *cluster* the nodes. Groups of nodes that are closely related form 'super-nodes'; the super-nodes themselves may be clustered into another level of super-nodes. Several visualization systems take this approach [9]. In effect, this approach gives a broad picture of the whole system.

Our approach is different: we present a detailed picture of a small part of the huge graph. On-line graph drawing aims to investigate the visualization of huge graphs which are partially unknown. At any time, a tiny but nonempty subgraph called the 'logical frame' is known. A picture of the logical frame is displayed on the screen. Exploration of the huge graph proceeds by changing the logical frame.

On-line graph drawing provides a major departure from traditional graph drawing methods. It does not pre-define the geometry of the whole graph at once; instead it incrementally calculates and maintains a small local visualization on-line, corresponding to the change of the user's focus. This feature enables the user to logically explore the huge graph without requiring the whole graph to be known.

We allow the user to change focus nodes by selecting another node of the logical frame, but we do not anticipate the user's selection. However, we do assume that we always can discover the neighborhood of the focus node. This is analogous to following hypertext links from the web page represented by the focus node of the current view.

The layout of the logical frame must satisfy the usual readability criteria for drawings of graphs (see Lin and Eades [10]); for example, edge crossings should be avoided, the nodes should be spread evenly over the page, and a variety of application dependent geometric constraints should be satisfied. In addition, the transition from the picture of one logical frame to the next should preserve the mental map [8], that is, the difference between successive drawings should be small enough that the user perceives the transition to be smooth.

This paper describes a model for on-line graph drawing, and describes an instantiation of that model in a system On-line force-directed animated visualization (OFDAV) for assisting web navigation. An interesting part of the system is a new force-directed graph drawing algorithm. This new algorithm can be used to produce a continuous sequence of layouts that address the above drawing criteria and preserve the mental map.

The next section describes the on-line graph model; in particular, we describe the transitions between logical frames. Section 3 describes our model for drawings of the logical frames, and the animated transitions between the drawings. The algorithms used to create the drawings and the animations are described in Section 4. The final section presents some sample screenshots which display the results of our algorithms.

2. The On-line Graph Model

In this section we describe the on-line graph model and the transitions between logical frames. Figure 2 shows a single frame F_i collected from OFDAV.

We now describe these concepts in a greater level of formality.

Our aim is to provide tools for exploring a huge graph $G=(V, E)$. The exploration of the graph G uses a sequence of *logical frames* (see Figure 3):

$$F_1=(G_1, Q_1)$$
$$F_2=(G_2, Q_2)$$
$$F_3=(G_3, Q_3)$$
$$\vdots$$

Each logical frame $F_i=(G_i, Q_i)$ consists of a connected subgraph $G_i=(V_i, E_i)$ of G and a quene Q_i of 'focus' nodes. For example, in Figure 2, we have $Q_i=\{q1, q2\}$, $V_i=\{q1, q2, a, b, c, d, e, f, g, h, i\}$ and $E_i=\{(q1, q2), (q1, a), (q1, b), (q1, c), (q1, d), (q1, e), (q2, f), (q2, g), (q2, h), (q2, i)\}$.

Successive logical frames differ only by a few nodes. The sequence of logical frames is the sequence of subgraphs of G viewed by the user; a user interaction changes from one logical frame to the next.

To define the 'logical frame' notion more precisely, we need some terminology. Graph-theoretic terminology is from Bondy and Murty [11].

Suppose that $G=(V, E)$ is a graph, $v \in V$, and d is a nonnegative integer. The *distance-d neighborhood* $N_d(v)$ of v is the subgraph of G induced by the set of nodes

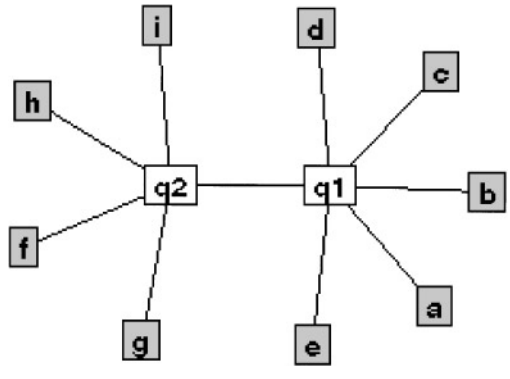


Figure 2. A single frame F_i

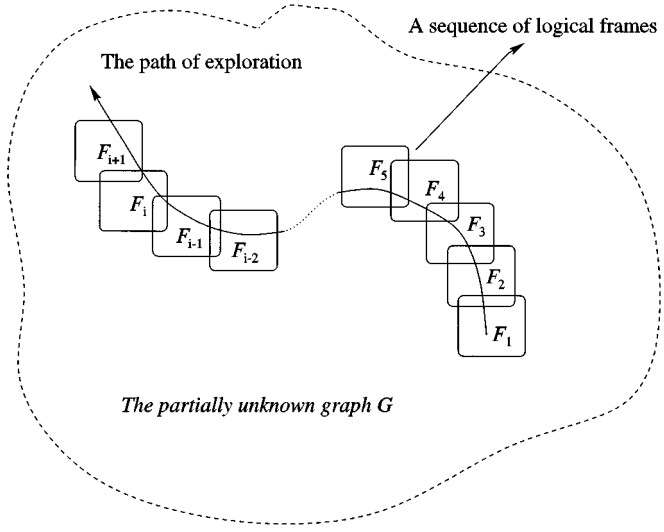


Figure 3. The exploration of the huge, partially unknown graph G uses a sequence of logical frames $F_1=(G_1, Q_1)$, $F_2=(G_2, Q_2)$, …

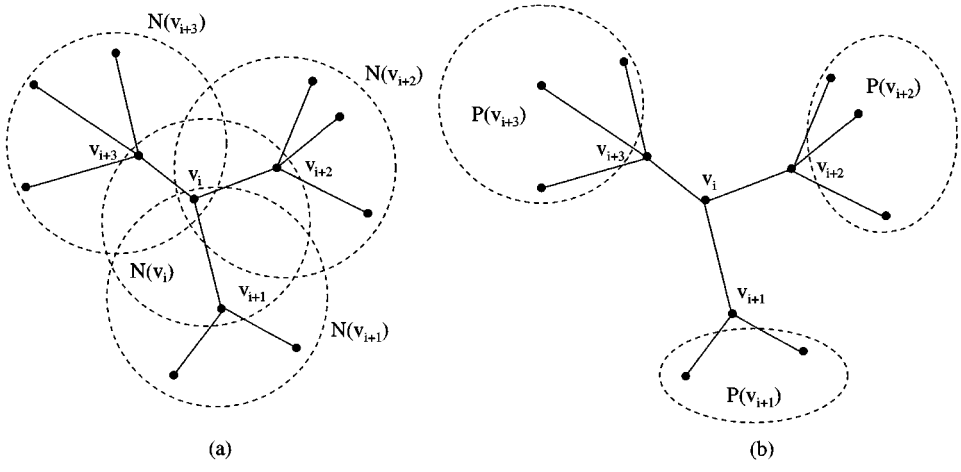


Figure 4. The logical frame, that is, the subgraph which is currently being viewed, is induced by a sequence $N(v_i)$, $N(v_{i+1})$, $N(v_{i+2})$, $N(v_{i+3})$ of neighborhoods: (a) There are four *distance-1* neighborhoods in this logical frame F_i. (b) There are three local parts of *distance-1 neighborhood* in this F_i

whose graph-theoretic distance from v is at most d. In our implementation we have chosen $d=1$, and we write $N_1(v)$ as $N(v)$ and call it the *neighborhood* of v. Note that $v \in N(v)$.

Given a queue $Q=(v_1, v_2, \ldots, v_s)$ of nodes, the subgraph of G induced by the union of $N(v_1)$, $N(v_2)$, …, $N(v_s)$ is called a *logical frame* [see Figure 4(a)]. The nodes $v_1, v_2, \ldots, v_s$ are the *focus nodes* of the logical frame.

Suppose that $Q=(v_1, v_2, \ldots, v_s)$ is the queue of focus nodes in G. We divide each neighborhood $N(v)$ into two parts, the *common part* $C(v)$, and the *local part* $P(v)$, defined as follows:

- $C(v)$ is the graph induced by the nodes of $N(v)$ which also occur in a neighborhood $N(v')$ for some focus node $v' \neq v$; i.e.

$$C(v_j)=\sum_{i=1,\, i\neq j}^{s} N(v_j)\cap N(v_i)$$

- $P(v)$ is the graph induced by those nodes of $N(v)$ which do not occur in a neighborhood $N(v')$ for any focus node $v' \neq v$ [see Figure 4(b)], i.e.

$$P(v_j)=N(v_j)-\sum_{i=1,\, i\neq j}^{s} N(v_i)$$

The exploration of the G proceeds by visualizing a sequence $F_1=(G_1, Q_1)$, $F_2=(G_2, Q_2), \ldots$, of logical frames. As mentioned above, each logical frame F_i consists of a subgraph G_i of G and a queue Q_i of nodes of G_i.

In practice, only a small number of nodes can be on the screen at a time. We assume a global constant B which is an upper bound on the queue size. We have found that for web graphs, values of B between 7 and 10 ensure that 20–100 nodes are on the screen at a time.

A logical frame F_{i+1} is obtained from F_i by the addition of a focus node u and its neighborhood, and the deletion of at most one focus node u and its neighborhood, i.e.

$$Q_{i+1}=(Q_i\cup\{u\})-\{v\}$$

In most cases, the initial logical viewing frame F_1 consists of the subgraph induced by one focus node and its neighborhood; in some cases, there may be up to B focus nodes with their neighborhoods.

To change from one logical frame F_i to the next F_{i+1}, the user selects a nonfocus node $u \in F_i$ (with a mouse click); if $P(u) \neq \emptyset$, then the node u becomes a focus node and is added to the queue Q. Note that the addition of u implies that the nodes in $P(u)$ will be added to F_{i+1}.

Whenever a new focus node u is to be added to the end of the queue while the queue is already full (the length is already B), then an old node v must be deleted from the queue. Note that the nodes in $P(v)$ will be deleted from F_i. This operation requires a *deletion policy*; two such policies are:

- *FIFO*: delete the node at the head of the Q; this is the 'least recently used' focus node.
- *Largest K-distance rule*: delete the node in Q whose graph-theoretical distance from the new focus node is the largest. If there is more than one such node, then we may use a FIFO rule to choose one of them. This rule guarantees the connectivity of F_{i+1} as long as F_i is connected.

The system OFDAV described in Section 5 uses the FIFO policy.

3. The Graph Drawing and Animation Models

A *drawing* D of a graph $G=(V, E)$ consists of a location for each node $v \in V$ and a route for each edge $e \in E$.

A visualization of the sequence $F_1=(G_1, Q_1)$, $F_2=(G_2, Q_2), \ldots$ of logical frames consists of a drawing D_i of each graph G_i; each drawing D_i is a *key frame* of the visualization.

Key frame sequences occur in many interactive systems which handle relational information. Most such systems suffer from the 'mental map' problem: a small logical change in the graph results in a large change in relative positions of nodes in the drawing. The mental map problem has been addressed in several ways [8]; here we use animation or 'in betweening' along with a force-directed layout algorithm to preserve the mental map between key frames.

The layout of the key frame must satisfy the usual readability requirements of graph drawing [2]. Experience has shown that traditional force-directed algorithms are moderately successful in achieving these readability requirements. For on-line graph drawing, we have some further requirements. We say that the convex hull of the images of the focus node and its local part is the *local region* of the focus node. For on-line graph drawing, to help the user follow the addition and deletion of nodes, the local regions of a key frame should not overlap. In Section 4 we describe a 'modified spring algorithm' which achieves this extra requirement (see Figures 10 and 11). On-line graph drawing also requires a smooth transition between key frames. This is achieved by animating the force-directed algorithm described in Section 4.

We now describe the animation method.

Each drawing D_i is a 'spring drawing', that is, it is calculated using a force-directed algorithm. This algorithm places nodes so that a global energy function is locally minimized. One can view the algorithm as follows. Each node is replaced by a steel ring, and edges are replaced by steel springs. The rings have a gravitational repulsion acting between them. This is illustrated in Figure 5. It is simple to compute the energy of a specific layout of such a linked structure. The algorithm computes a layout with

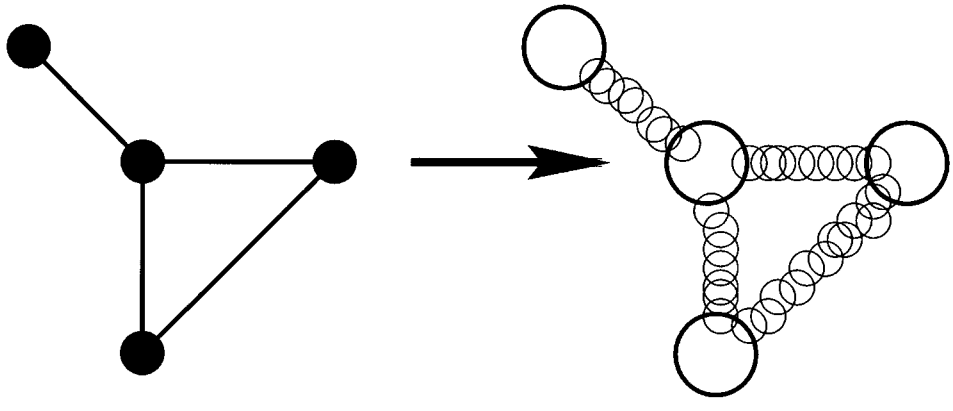


Figure 5. In the spring model, each node is replaced by a steel ring, and edges are replaced by steel springs

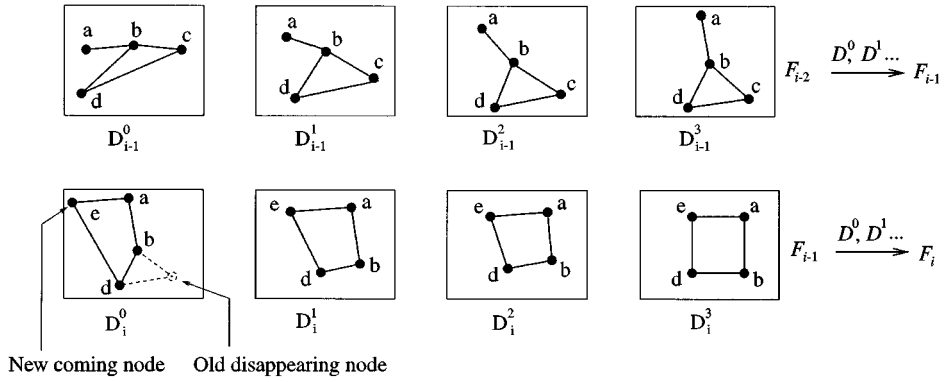


Figure 6. For each transformation from F_{i-1} to F_i, there is a sequence $D^0(F_i)$, $D^1(F_i)$, … of drawings called the *screens* of F_i. Two transformations, from F_{i-2} to F_{i-1} and from F_{i-1} to F_i consist of two sequences of animated drawings

a locally minimal energy. Spring drawings have been used in a variety of classical graph drawings systems [12–18].

Our 'in-betweening' technique (see Figure 6) is described next. It aims to achieve the twin goals of good layout and the preservation of the mental map.

The in-betweening consists of a sequence D^0_{i+1}, D^1_{i+1}, D^2_{i+1}, …, $D^k_{i+1} = D_{i+1}$ of drawings of G_{i+1} called *screens*. They are computed by a modified spring algorithm, described in the next section. The locations of the nodes in D^j_{i+1} differ only slightly from the locations of the nodes in D^{j+1}_{i+1}; the appearance is that of all the nodes moving slowly. The new focus node moves linearly toward the center of the page. The other nodes move relative to the position of the focus node according to the spring and gravitational forces; each D^j_{i+1} has energy a little lower than that of D^{j-1}_{i+1}.

We use this slightly unusual in-betweening method because it gives the user good feedback for the force system: the user can 'see' the forces in operation. This makes it easier for the user to predict and plan the layout. A simple linear interpolation between D^{j-1}_{i+1} and D^j_{i+1} would not provide such feedback.

The nodes of the first screen D^0_1 of the first logical frame are initially placed at random locations.

The change in logical frames from F_i to F_{i+1} is triggered by a mouse click on the new focus node. This click has the graphical effect of graphically deleting node images of nodes in $F_i - F_{i+1}$ and adding the nodes in $F_{i+1} - F_i$. (In our implementation, the nodes of $F_{i+1} - F_i$ appear on mouse-down and the nodes of $F_i - F_{i+1}$ fade after mouse-up.) The nodes common to F_{i+1} and F_i stay in the same locations on the screen as they were in the final screen of F_{i-1}. In Figure 6, the nodes *a*, *b* and *d* are common to F_{i+1} and F_i. Note that the locations of *a*, *b* and *d* in the final screen of F_{i-1} and the first screen of F_i are the same. The new nodes (of $F_i - F_{i-1}$) appear radially around and very close to their neighbors in F_{i-1}. The old nodes (of $F_{i-1} - F_i$) fade and disappear smoothly.

4. The Modified Spring Algorithm

This force-directed animation algorithm is based on Eades [13] which is the combination of Hooke's law springs and Newtonian gravitational forces. In order to address the

specific criteria of on-line drawing, we add extra forces among the neighborhoods, $N(v_i)$, $N(v_{i+1}), \ldots, N(v_{i+B-1})$ of the focus nodes. These extra forces are used to separate the neighborhoods so that the user can visually identify the changes. This extra force is also a Newtonian gravitational force.

We also treat these graphs as a set of rooted trees because many of the graphs that we deal with are trees, or tree-like structures.

4.1. The Force Model

Suppose that $F_i = (G_i, Q_i)$ is the logical frame which is currently being viewed on the screen, and $G_i = (V_i, E_i)$.

The total force applied on node v is

$$f(v) = \sum_{u \in N(v)} f_{uv} + \sum_{u \in V_i} g_{uv} + \sum_{u \in Q_i} h_{uv} \tag{1}$$

where f_{uv} is the force exerted on v by the spring between u and v, and g_{uv} and h_{uv} are the gravitational repulsions exerted on v by one of the other node u in F_i.

The force f_{uv} follows Hooke's law, that is, f_{uv} is proportional to the difference between the distance between u and v and the zero energy length of the spring. The Newtonian gravitational forces g_{uv} and h_{uv} follow an inverse square law. Let us denote the Euclidean distance between points p and q by $d(p, q)$, and suppose that the position of node v is denoted by $p_v = (x_v, y_v)$. Thus from Eq. (1), the x component $f_x(v)$ of the force $f(v)$ on v is

$$f_x(v) = \sum_{u \in N(v)} k_{uv}^{(1)} \frac{(d(p_u, p_v) - l_{uv})(x_v - x_u)}{d(p_u, p_v)} + \sum_{u \in V_i} k_{uv}^{(2)} \frac{(x_v - x_u)}{(d(p_u, p_v))^3} + \sum_{u \in Q_i} k_{uv}^{(3)} \frac{(x_v - x_u)}{(d(p_u, p_v))^3} \tag{2}$$

The y component $f_y(v)$ of $f(v)$ has a similar expression. The parameters l_{uv}, $k_{uv}^{(1)}$, $k_{uv}^{(2)}$, and $k_{uv}^{(3)}$ are independent of the positions of the nodes, and may be interpreted as follows.

- The zero energy length of the spring between u and v is l_{uv}. If the spring has length l_{uv} (that is, $d(p_u, p_v) = l_{uv}$), then no force is exerted by (u, v). In our experiment, we normally set $l_{uv} = 70$. (The total size of the display applet in OFDAV is 1000 units wide and 700 units high.)
- The 'stiffness' of the spring between u and v is expressed with $k_{uv}^{(1)}$: the larger the value of $k_{uv}^{(1)}$, the more the tendency for the distance between u and v to be close to l_{uv}. In our system OFDAV, we set

$$k_{uv}^{(1)} = \begin{cases} \frac{1}{3} & \text{if } u \text{ and } v \in Q \\ \frac{1}{30} & \text{otherwise} \end{cases}$$

- The strength of the gravitational repulsion between any u and v depends on $k_{uv}^{(2)}$. In OFDAV, we set $k_{uv}^{(2)} = 3$.

- The strength of the extra gravitational repulsion between $u \in Q$ and v depends on $k_{uv}^{(3)}$. In OFDAV, we set $k_{uv}^{(3)} = 6$.

This modified spring model aims to satisfy four important aesthetics:

1. The spring force between adjacent nodes aims to ensure that the distance between adjacent nodes u and v is approximately equal to l_{uv}.
2. The gravitational force aims to ensure that nodes are not too close together.
3. The extra gravitational force aims to minimize the overlaps among the neighborhoods, $N(v_i), N(v_{i+1}), \ldots, N(v_{i+B-1})$, within the logical frame. The aim is to ensure that the next nodes to disappear are placed close together and to separate them from the rest of the logical frame; we have found that this makes it easier to identify the deleting objects.
4. This extra gravitational force also aims to keep the layout of the queue of focus nodes close to a straight line. This helps to give the direction of the exploration of the huge graph G, and helps the user to gain an understanding of where they are going and where they came from: new nodes appear in one end of the line and nodes disappear at the other end.

To explain this extra gravitational force, we give a simple example. The logical frame shown in Figure 7 consists of four focus nodes V_1, V_2, V_3 and V_4, and the corresponding

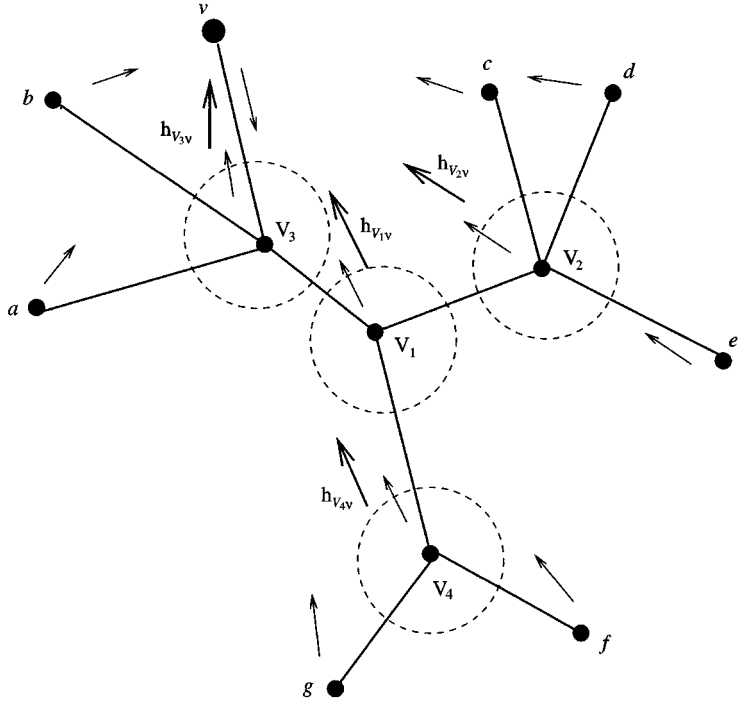


Figure 7. An example of applying Modified Spring Algorithm

neighborhoods of these focus nodes. Now, consider the total force on node v. In the traditional spring algorithm we have

$$f_v = f_{V_3,v} + \sum_{u \in \{a, \ldots, g, V_1, \ldots, V_4\}} g_{uv} \tag{3}$$

In the modified spring algorithm with extra gravitational forces we have

$$f_v = f_{V_3,v} + \sum_{u \in \{a, \ldots, g, V_1, \ldots, V_4\}} g_{uv} + \sum_{u \in \{V_1, \ldots, V_4\}} h_{uv} \tag{4}$$

Here the extra gravitational forces are $h_{V_1,v}$, $h_{V_2,v}$, $h_{V_3,v}$ and $h_{V_4,v}$.

4.2. Finding an Equilibrium Configuration of the Forces

An *equilibrium drawing* of the logical frame F_i is a drawing in which the total force $f(v)$ on each node v is zero. Equivalently, the model seeks to find a drawing in which the potential energy is locally minimal with respect to the node positions. The key frame D_i is at equilibrium, and the graph G_{i+1} differs from G_i by just a few nodes.

Consider the in-betweening sequence $D^0_{i+1}, D^1_{i+1}, D^2_{i+1}, \ldots, D^k_{i+1} = D_{i+1}$ of screens leading from the key frame D_i to the key frame D_{i+1}. This begins with a drawing G_{i+1} that is not at equilibrium, but as it differs very little from D_i, it is close to equilibrium. Each D^j_{i+1} is a little closer to equilibrium; i.e. the animation is driven by the forces moving the nodes toward equilibrium positions. This is accomplished by moving each node v a small amount proportional to the magnitude of $f(v)$ in the direction of $f(v)$ at each step. Next we consider the details of this motion.

We use a very simple numerical technique to minimize energy. The technique has two aims. The first is to find an equilibrium layout as per (1) and (2). The second is to produce a smooth motion on the screen, that is, to give a visual effect of continuous movement. The second aim implies that we do not use a complex (and perhaps faster) numerical technique because it would produce a jerky motion.

The force $f(v)$ has two components, $f_x(v)$ and $f_y(v)$; we show how to compute the movement $\Delta_x(v)$ in the x direction of a single animation step; the y movement $\Delta_y(v)$ is similar.

From Newton's second law of motion, $f_x(v) = m(v)a_x(v)$. Here, $m(v)$ is the mass of the node v, and $a_x(v)$ is the acceleration of the motion. For simplicity, we assume that the mass of every node is one; hence $a_x(v) = f_x(v)$.

The initial speed of node v on the first screen D^0_{i+1} is zero, that is, $V_x(0) = 0$. Thus, we have the speed of motion of node v at time t in the x direction as

$$V_x(t) = \int_0^t a_x(t)\,\mathrm{d}t \tag{5}$$

and the position of v at time t in the x direction as

$$x(t) = x(0) + \int_0^t \int_0^t a_x(t)\,\mathrm{d}t\,\mathrm{d}t \tag{6}$$

Let $t_0 = 0$, $t_1 = t_0 + \Delta_t, \ldots, t_{j+1} + \Delta_t, \ldots$. When Δ_t is very small, we can regard the acceleration of v in the time interval $[t_{j-1}, t_j]$ as a constant $= a_x(t_{j-1})$. Thus, we have the

following approximation:

$$V(t_j) = V(t_{j-1}) + a_x(t_{j-1})\Delta_t \tag{7}$$

and

$$x(t_j) = x(t_{j-1}) + V_x(t_{j-1})\Delta_t + \frac{a_x(t_{j-1})}{2}\Delta_t^2 \tag{8}$$

Let

$$\Delta_x(v) = x(t_j) - x(t_{j+1}) = V_x(t_{j-1})\Delta_t + \frac{a_x(t_{j-1})}{2}\Delta_t^2 \tag{9}$$

Here Δ_t is the time period of a single animation loop (normally called the *perceptual processing time constant*); we set $\Delta_t = 0.1$ s. We can also safely ignore the term $V_x(t_{j-1})\Delta_t$.

Now, we have

$$\Delta_x(v) = \frac{a_x(t_{j-1})}{2}\Delta_t^2 \tag{10}$$

Substituting (10) into Newton's second law of motion, we have

$$\Delta_x(v) = \frac{f_x(v)}{2}\Delta_t^2 = C * f_x(v) \tag{11}$$

where $C = \Delta_t^2/2$. Note that if $\Delta_t = 0{\cdot}1$ s, then $C = 1/200$.

Now, we can rewrite the force model shown in (1) and (2) to an animated motion model

$$\Delta_x(v) = \sum_{u \in N(v)} k_{uv}^{(1)} \frac{(d(p_u, p_v) - l_{uv})(x_v - x_u)}{d(p_u, p_v)} + \sum_{u \in V_i} k_{uv}^{(2)} \frac{(x_v - x_u)}{(d(p_u, p_v))^3} + \sum_{u \in Q_i} k_{uv}^{(3)} \frac{(x_v - x_u)}{(d(p_u, p_v))^3} \tag{12}$$

Once the actual value $\Delta_x(v)$ of the distance increment for the next motion in the $(i+1)$th animation loop has been generated, we have to restrict the distance of the motion in the next animation loop to achieve the goal of producing smoothly interpolated curves and avoid 'jumps' of the objects on the screen. So that the final value assigned to $\Delta_x(v)$ is

$$\Delta_x(v) = \begin{cases} -5 & \text{if } \Delta_x(v) \le -5 \\ \Delta_x(v) & \text{if } -5 < \Delta_x(v) < 5 \\ 5 & \text{if } 5 \le \Delta_x(v) \end{cases} \tag{13}$$

In our implementation of this model, we further simplify the animation to reduce the computation time: we ignore the gravitational force g_{uv} between nodes u and v if the distance $d(p_u, p_v)$ is greater than 100 units, and we ignore the extra gravitational force h_{uv} between nodes u and v if the distance $d(p_u, p_v)$ is greater than 400 units.

4.3 The Complete Layout Algorithm

The following pseudo-code is used to calculate the distance increment value Δ for every node in a single animation loop. This process is repeated in every time period Δ_t. We

assume that:

- N_{node} is an integer pointer that points to the last node element in array $nodes[1 \ldots n]$. The current logical frame has a set of nodes, $nodes[1]$, $nodes[2]$, $\ldots$ $nodes[N_{node}]$.
- N_{edge} is an integer pointer that points to the last edge element in array $edges\ [1 \ldots n]$. The current logical frame has a set of edges, $edges[1]$, $edges[2]$, $\ldots$ $edges[N_{edge}]$.
- $edges[i\,].startpoint$ is an integer pointer that points to a node element in $nodes[1 \ldots n]$ from which the edge starts.
- $edges[i\,].endpoint$ is an integer pointer that points to a node element in $nodes[1 \ldots n]$ to which the edge ends.
- $edges[i\,].l_{initial}$ is the zero energy length of this edge.
- $nodes[i\,].x$ and $nodes[i\,].y$ represent the position of a node in graph image. This node is stored in the ith element of the array $nodes[1 \ldots n]$.

Modified Spring Algorithm

---

for $i=1$ to $i=N_{edge}$ do $\{$Calculating Spring forces$\}$

begin

$vector_x \leftarrow nodes[edges[i\,].endpoint\,].x - nodes[edges[i\,].startpoint\,].x;$

$vector_y \leftarrow nodes[edges[i\,].endpoint\,].y - nodes[edges[i\,].startpoint].y;$

$d \leftarrow \sqrt{vector_x^2 + vector_y^2};$

$f_{spring} \leftarrow (d - edges[i\,].l_{initial}) * k^{(1)}/d;$

$\Delta x_{spring} \leftarrow f_{spring} * vector_x;$

$\Delta y_{spring} \leftarrow f_{spring} * vector_y;$

if $(nodes[edges[i\,].endpoint\,] \notin Q)$ or $(nodes[edges[i\,].startpoint\,] \notin Q)$ then

$\Delta x_{spring} \leftarrow \Delta x_{spring}/10;$

$\Delta y_{spring} \leftarrow \Delta y_{spring}/10;$

end $\{$if$\}$;

$nodes[edges[i\,].endpoint\,].\Delta x \leftarrow nodes[edges[i\,].endpoint\,].\Delta x - \Delta x_{spring};$

$nodes[edges[i\,].endpoint\,].\Delta y \leftarrow nodes[edges[i\,].endpoint\,].\Delta y - \Delta y_{spring};$

$nodes[edges[i\,].startpoint\,].\Delta x \leftarrow nodes[edges[i\,].startpoint\,].\Delta x + \Delta x_{spring};$

$nodes[edges[i\,].startpoint\,].\Delta y \leftarrow nodes[edges[i\,].startpoint].\Delta y + \Delta y_{spring};$

$i \leftarrow i+1;$

end $\{$Calculating Spring forces$\}$;

$Radius_1 \leftarrow 100;$

$Radius_2 \leftarrow 400;$

for $i=1$ to $i=N_{node}$ do $\{$Calculating gravitational & extra forces$\}$

begin

$\Delta x_{grav} \leftarrow 0;$

$\Delta y_{grav} \leftarrow 0;$

for $j=1$ to $j=N_{node}$ do

```
    begin
        if j = i then
            j ← j + 1;
            break;
        end {if};
        vector_x ← nodes[i].x − nodes[j].x;
        vector_y ← nodes[i].y − nodes[j].y;
        d ← √(vector_x^2 + vector_y^2);
        if d < Radius_1 * Radius_1 then
            Δx_grav ← Δx_grav + (vector_x / d) * k^(2);
            Δy_grav ← Δy_grav + (vector_y / d) * k^(2);
        end {if};
        if (nodes[j] ∈ Q) and (d < Radius_2 * Radius_2) then
            Δx_grav ← Δx_grav + (vector_x / d) * k^(3);
            Δy_grav ← Δy_grav + (vector_y / d) * k^(3);
        end {if};
        j ← j + 1;
    end {for};
    Δd_grav ← √(Δx_grav^2 + Δy_grav^2);
    nodes[i].Δx ← nodes[i].Δx + Δx_grav / Δd_grav;
    nodes[i].Δy ← nodes[i].Δy + Δy_grav / Δd_grav;
    i ← i + 1;
end {Calculating gravitational & extra forces};
```

```
for i = 1 to i = N_node do {Assigning new position value to nodes}
begin
    if nodes[i].Δx ≤ −5 then
        nodes[i].x ← nodes[i].x − 5;
    else if −5 < nodes[i].Δx < 5 then
            nodes[i].x ← nodes[i].x + nodes[i].Δx;
        else if 5 ≤ nodes[i].Δx then
                nodes[i].x ← nodes[i].x + 5;
    end {if};
    if nodes[i].Δy ≤ −5 then
        nodes[i].y ← nodes[i].y − 5;
    else if −5 < nodes[i].Δy < 5 then
            nodes[i].y ← nodes[i].y + nodes[i].Δy;
        else if 5 ≤ nodes[i].Δy then
                nodes[i].y ← nodes[i].y + 5;
    end {if};
    i ← i + 1;
end {Assigning new position value to nodes};
```

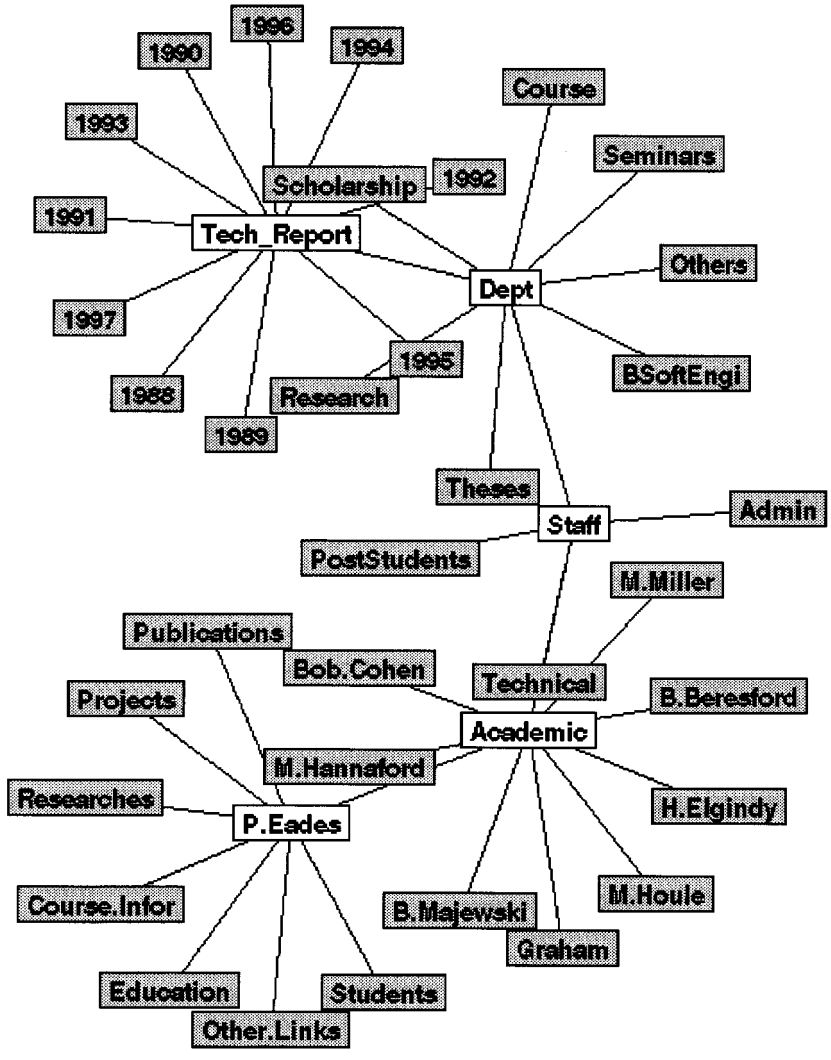


Figure 8. A key frame with focus nodes Dept, Tech_Report, Staff, Academic, P.Eades. computed by the traditional spring algorithm. This layout does not clearly show the direction of the exploration

The parameters, $k^{(1)}$, $k^{(2)}$, $k^{(3)}$, $Radius_1$ and $Radius_2$, may be altered to adjust the layout for a better view. The best value of these parameters that we should choose mostly depend on the actual size of graphical nodes and the initial length $edges[i].l_{initial}$ of the edges. A demonstration of this algorithm is available on the web [19].

5. Examples

In this section, we present some examples from our system OFDAV that show:

- How the modified spring algorithm addresses specific criteria of on-line graph drawing.
- How we can use OFDAV to navigate the web.

5.1. Examples of Addressing Specific Criteria of On-line Graph Drawing

For comparative purposes, we also show some graphs drawn by a traditional spring algorithm [13].

The samples are from a visualization of the web graph, which begins with web context of the Department of Computer Science and Software Engineering at the University of Newcastle.

The layout of F_i must show the direction of the exploration: The layout must clearly show the direction of the exploration of the huge graph. This is done by presenting the queue Q of focus nodes as a line that is nearly straight; this helps the user to gain an understanding of where they are, where they are going, and and where they came from. New nodes appear in one end of the frame and old nodes disappear at the other end.

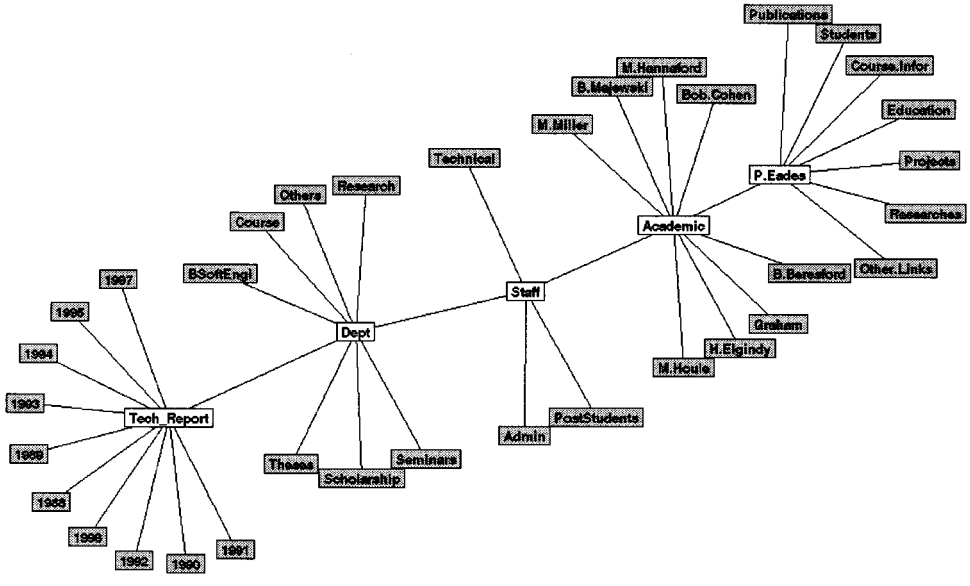


Figure 9. The same logical frame as Figure 8, but using our modified spring algorithm. Here the focus nodes are roughly in a straight line and clearly show the direction of the exploration

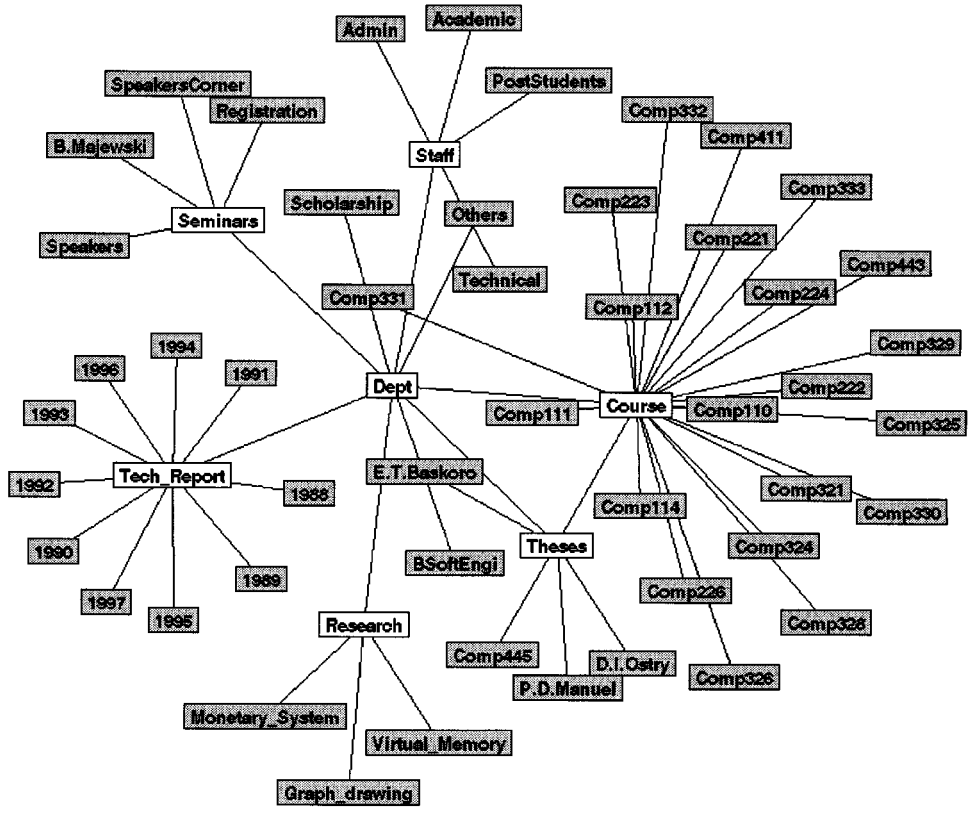


Figure 10. A key frame with focus nodes Dept, Tech_Report, Staff, Course, Seminars, Research, Theses computed with the traditional spring algorithm. This layout has five overlaps among the local regions of the neighborhoods of the focus nodes

The straightness of this line is induced by the extra forces in the modified spring algorithm.

Figures 8 and 9 give a comparison of the traditional spring model and our modified spring model.

Reducing the overlaps among the local regions: Overlaps between the regions occupied by the local parts of the neighborhoods can lead to user confusion, especially when the transition between frames deletes some nodes. Our modified spring algorithm tends to reduce (in most cases, to eliminate) these overlaps.

Figures 10 and 11 compare the traditional spring model and our modified spring model.

Reducing the number of edge crossings: For tree-like structures (such as web graphs and file system graphs) the modified spring algorithm tends to reduce or eliminate edge crossings. The traditional spring algorithm does not.

This is illustrated in Figures 12 and 13.

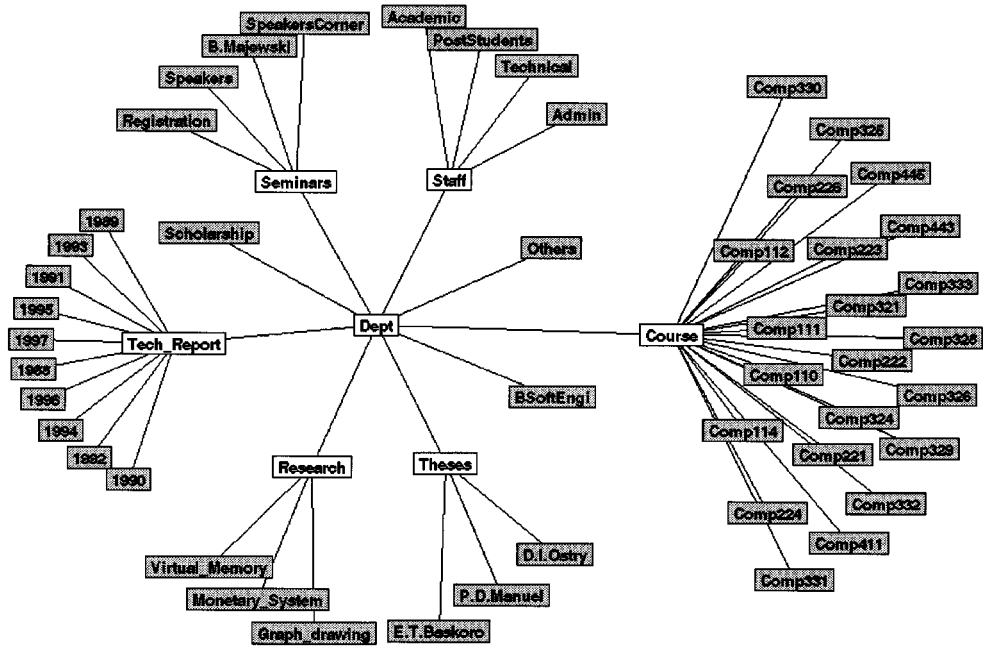


Figure 11. The same logical frame as in Figure 10, but the drawing is computed with our modified spring algorithm. There are no overlaps among the local regions of the neighborhoods

5.2 An Example of a Web Exploration

We now explain our concept of *exploratory* navigation. Let us imagine that we are exploring a large snowfield in Antarctica. We cannot see the whole of Antarctica. We only see the things that are located within a circular area of a certain radius. We call this circular area the current *visual field* (see Figure 14). When we walk from position P_i to P_{i+1}, the *visual field* changes; some new views come into the front of the *visual field*, and some are lost at the back. However, since this change of the view is slow and smooth, it does not disturb the mental map [8] of the explorer.

Our *on-line* visualization technique is based on this simple natural concept (the difference is that exploring the snowfield is a geometrical operation and our system is used for logical exploration of huge graphs). In this paper,

1. 'Antarctica' is analogous to 'the entire graph'.
2. The 'visual field' is analogous to 'logical frame'.
3. The 'changes of the visual field' is analogous to 'transitions between logical frames'.

Using OFDAV is somewhat akin to walking through this vast unexplored snowfield. The user can see a small portion of the snowfield (part of the global context), that is, the area (logically) close to the user. As the user walks, some new nodes come into the view (logical frame), and some are lost at the back of the view (logical frame).

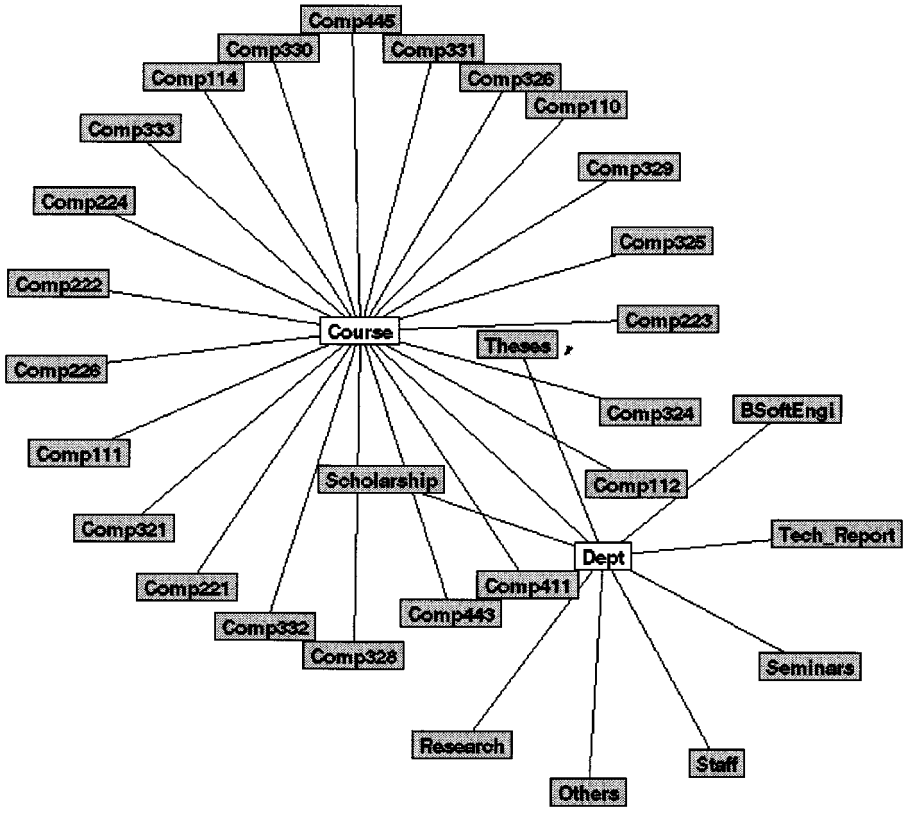


Figure 12. A key frame with focus nodes Dept, Course computed with the traditional spring algorithm. This layout has four edge crossings

To illustrate how the system works, a simple example session is presented in this section. The system first displays an initial logical key frame according to a user-specified URL at the beginning of a navigation. It then updates the frame on-line as the user moves the focus by clicking on a succession of new focus nodes.

Suppose that we start to navigate the web from the first initial frame F_1, the web context of the Department of Computer Science and Software Engineering at the University of Newcastle (see Figure 11). Then, we incrementally change the context to the next frame F_2 by clicking on the node labeled PostStudents (see Figure 15). This change is done smoothly through multiple animations preserving the user's mental map.

We see that the node PostStudents becomes a new focus node and it is added into the focus queue; its neighborhood appears and the node Course is deleted from the focus queue and its neighborhood disappears.

Figure 16 shows one frame further from Figure 15 after the user has selected the node Mao. Figure 17 shows five frames further into the user's exploration of web, after clicking on Links(M) , News, USA.news, CNN and then World. We see that the current

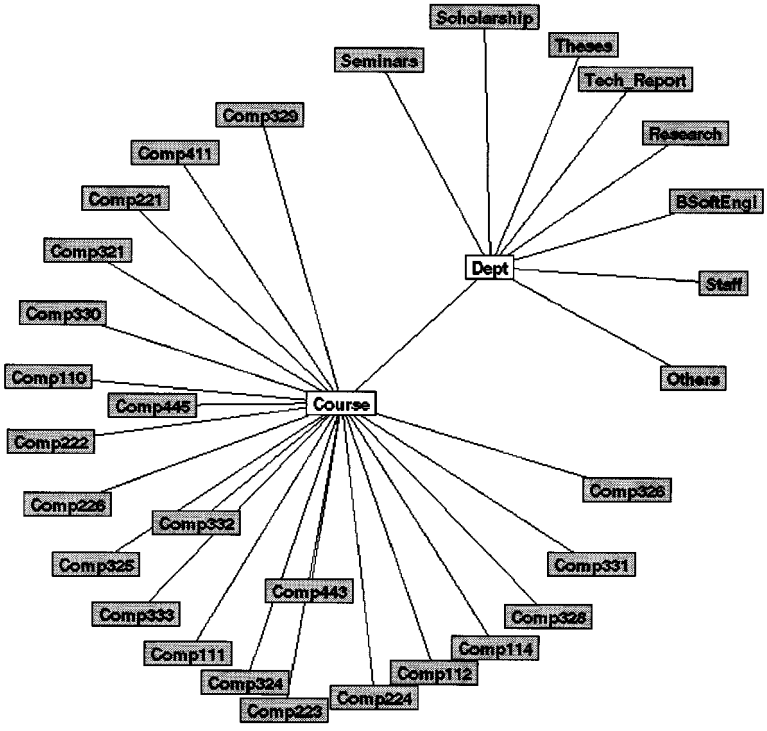
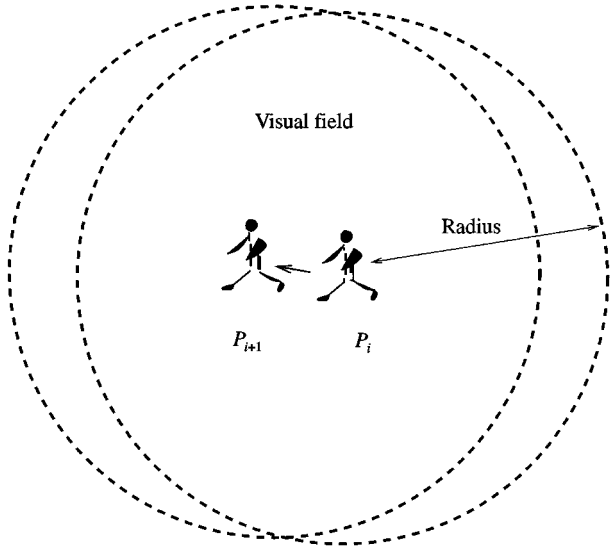


Figure 13. The same logical frame as in Figure 12, but the drawing is computed with our modified spring algorithm. There are no edge crossings



Radius = the maximun vision distance of human eyes.

Figure 14. The concept of exploratory navigation

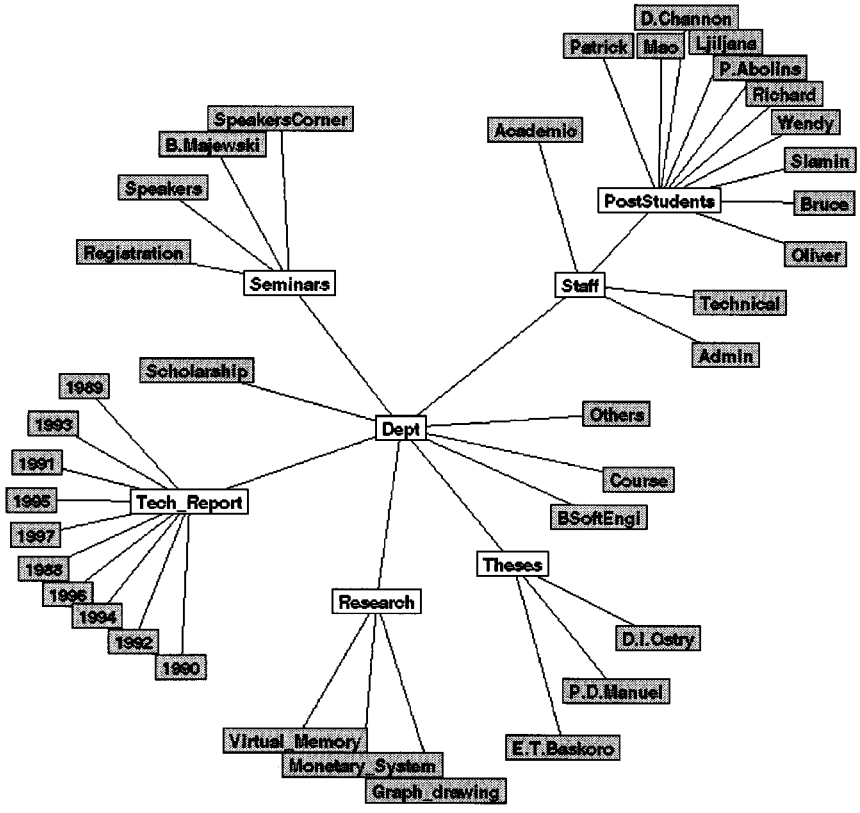


Figure 15. This is the frame F_2 following the graph in Figure 11. It has been formed after a user mouse click on the PostStudents node. Here, the queue of focus nodes is $Q_2 = \{$Dept, Tech_Report, Staff, Seminars, Theses, Research, PostStudents$\}$

web subgraph has been smoothly changed from the local department web site to the CNN web site.

6. Remarks

This paper describes a model and some methods for on-line graph drawing. Our methods include a modified spring algorithm, which improves on the traditional spring algorithm in several ways that are suitable for on-line graph drawing. These methods have been implemented in a system OFDAV for navigating web documents. Unlike most existing visualization techniques that build an overall geometrical representation of the graph (the global context) before navigating through it, OFDAV builds the visualization incrementally, as the user is exploring the huge graph. This feature enables the user to explore the huge graphs (with, perhaps, billions of nodes) without requiring the whole graph to be known.

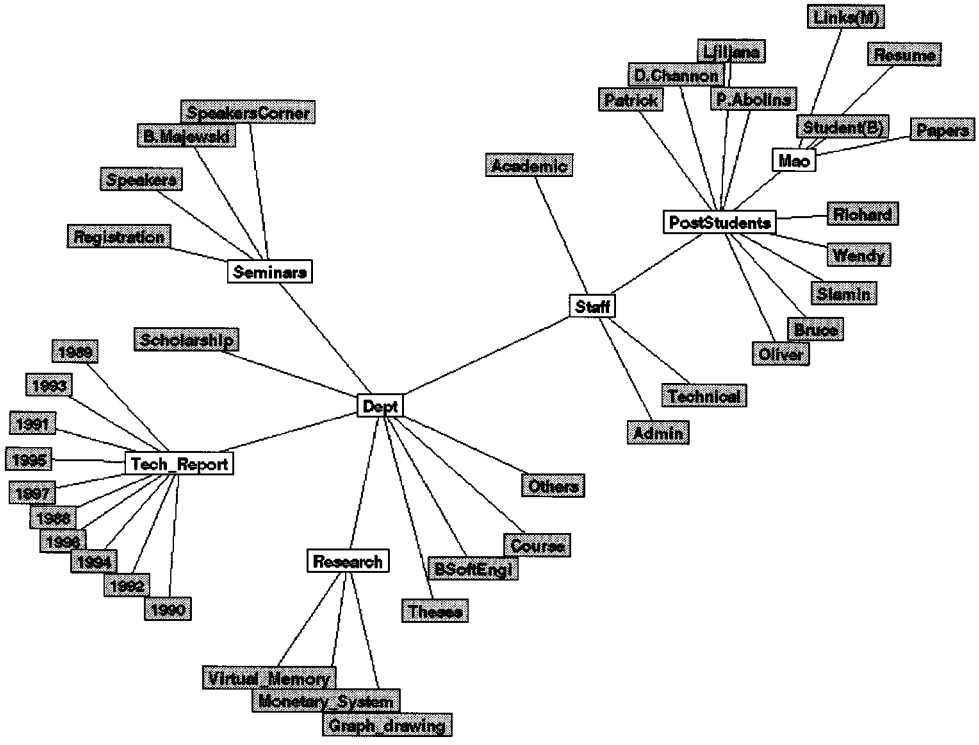


Figure 16. This is the frame F_3 following the graph in Figure 15. It has been formed after a user mouse click on the Mao node. Here, the queue of focus nodes is $Q_3 = \{$Dept, Tech_Report, Staff, Seminars, Research, PostStudents, Mao$\}$

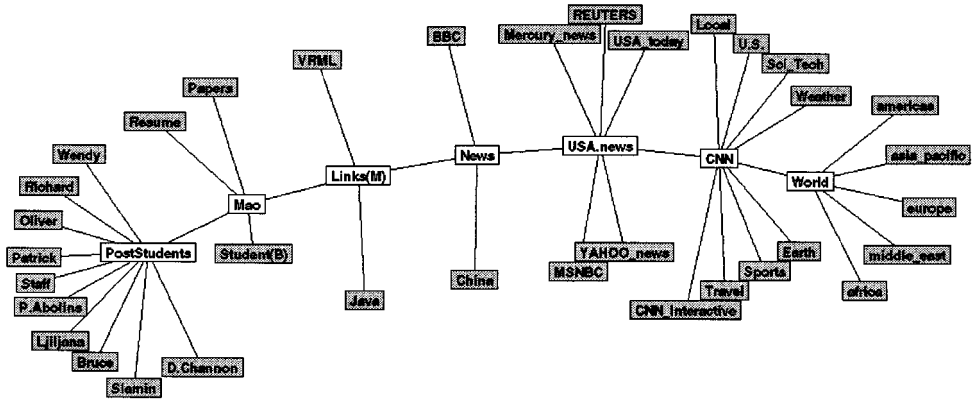


Figure 17. Five frames further from Figure 16 into the user's exploration. The frame F_8 has a focus queue, $Q_8 = \{$PostStudents, Mao, Links(M), News, USA.news, CNN, World$\}$

Our experience with OFDAV indicates that it is an effective navigation tool. Future directions for this research include consideration of more formal assessment of the effectiveness of our methods, and the application of the model and methods to create other systems for navigating large and partially unknown relational structures.

References

1. C. Batini, L. Furlani & E. Nardelli (1985) What is a good diagram? A pragmatic approach. In: *Proceedings of the 4th International Conf. on the Entity Relationship Approach*.
2. G. Di Battista, P. Eades, R. Tamassia & I. G. Tollis (1994) Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry Theory and Applications* 4, 235–282.
3. G. Di Battista, P. Eades, R. Tamassia & I. G. Tollis (1999) *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice-Hall, Englewood Cliffs, NJ, (to appear).
4. M. Sarkar & M. H. Brown (1994) Graphical fisheye views. *Communications of the ACM* 37, 73–84.
5. G. G. Robertson, J. D. Mackinlay & S. K. Card (1991) Cone trees: animated 3D visualizations of hierarchical information. In: *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 189–193.
6. G. G. Robertson, S. K. Card & J. D. Mackinlay (1993) Information visualization using 3D interactive animation. *Communications of the ACM* 36, 57–71.
7. J. Lamping, R. Rao & P. Pirollli (1995) A focus-context technique based on hyperbolic geometry for visualization large hierarchies. In: *Proceedings of InterCHI'95*.
8. P. Eades, W. Lai, K. Misue & K. Sugiyama (1995) Layout adjustment and the mental map. *Journal of Visual Languages and Computing* 6, 183–210.
9. K. Sugiyama & K. Misue (1991) Visualization of structural information: automatic drawing of compound digraphs. *IEEE Transaction on Systems, Man and Cybernetics* 21, 876–896.
10. T. Lin & P. Eades (1995) Integration of declarative and algorithmic approaches for layout creation. In: *Graph Drawing* (*Proceedings GD'94*), R. Tamassia & I. G. Tollis, eds, Lecture Notes Computer Science, Vol. 894. Springer, Berlin, pp. 376–387.
11. J. A. Bondy & U. S. R. Murty (1976) *Graph Theory with Applications.* North-Holland, New York, NY, U.S.A.
12. I. F. Cruz & J. P. Twarog. (1996) 3-d graph drawing with simulated annealing. In: *Proceedings of the GD'95*, Lecture Notes in Computer Science, Vol. 1027. Springer, Berlin, pp. 162–165.
13. P. Eades (1984) A heuristic for graph drawing. *Congressus Num antium* 42, 149–160.
14. A. Frick, C. Keskin & V. Vogelmann (1997) Integration of declarative approaches. In: *Proceedings of the GD'96*, Lecture Notes in Computer Science, Vol. 1190. Springer, Berlin, pp. 184–192.
15. A. Frick, A. Ludwig & II. Mehldau (1995) A fast adaptive layout algorithm for undirected graphs. In: *Proceedings of GD'94*, Lecture Notes in Computer Science, Vol. 894. Springer, Berlin, pp. 388–403.
16. T. Fruchterman & E. Reingold (1991) Graph drawing by force-directed placement. *Software - Practices and Experiments* 21, 1129–1164.
17. T. Kamada (1988) On visualization of abstract objects and relations. Ph.D. thesis, Department of Information Science, University of Tokyo.
18. T. Kamada (1989) Symmetric graph drawing by a spring algorithm and its applications to radial drawing.
19. URL: http://www.cs.newcastle.edu.au/mhuang/spring/spring.html. *Modified Spring Algorithm* (*System Demo*).