NicheWorks—Interactive Visualization of Very Large Graphs

Graham J. WILLS

The difference between displaying networks with 100–1,000 nodes and displaying ones with 10,000–100,000 nodes is not merely quantitative, it is *qualitative*. Layout algorithms suitable for the former are too slow for the latter, requiring new algorithms or modified (often relaxed) versions of existing algorithms to be invented. The density of nodes and edges displayed per inch of screen real estate requires special visual techniques to filter the graphs and focus attention. Compounding the problem is that large real-life networks are often weighted graphs and usually have additional data associated with the nodes and edges. A system for investigating and exploring such large, complex datasets needs to be able to display both graph structure and node and edge attributes so that patterns and information hidden in the data can be seen. In this article we describe a tool that addresses these needs, the NicheWorks tool. We describe and comment on the available layout algorithms and the linked views interaction system, and detail two examples of the use of NicheWorks for analyzing Web sites and detecting international telephone fraud.

Key Words: Dynamic graphics; Exploratory data analysis; Graph layout; Networks.

1. INTRODUCTION

NicheWorks is a visualization tool for the investigation of very large graphs. By “very large” we mean graphs for which we cannot look at the complete set of labeled nodes and edges on one static display. Typical analyses performed using NicheWorks have between 20,000 and 1,000,000 nodes. On current mid-range workstations, a network of around 50,000 nodes and edges can be visualized and manipulated in real time with ease. Increased size decreases interactive performance linearly. NicheWorks allows the user to examine a variety of node and edge attributes in conjunction with their connectivity information. Categorical, textual and continuous attributes can be explored with a variety of one-way, two-way, and multidimensional views.

NicheWorks was designed to examine large telecommunications networks, and has been applied extensively to understanding calling patterns among telephone customers both for customer understanding and fraud detection. In this domain, information about

Graham J. Wills is a Member of Technical Staff in the Software Production Research Department (Data Visualization Group) of Bell Laboratories, Room 2F-323, Shuman Boulevard, Naperville, IL 60566 (Email: gwills@research.bell-labs.com).

©1999 American Statistical Association, Institute of Mathematical Statistics, and Interface Foundation of North America
Journal of Computational and Graphical Statistics, Volume 8, Number 2, Pages 190–212

the customer (type of service, geographical location, etc.) and the calls they make (date, time of day, duration) need to be understood in the context of who calls whom. NicheWorks was developed to examine such inter-relationships. Typical questions that NicheWorks was designed to answer deal with clustering users based on their calling patterns and collected statistical attributes; detecting and characterizing atypical calling patterns; understanding how an interesting subset's calling patterns differ from those of the whole; and identifying individuals who have been classified into a given category by a purely data-driven algorithm, but who do not appear to fit that category based on their calling patterns.

Since its inception, a number of data sources have been analyzed by NicheWorks, and NicheWorks has been adapted and modified into a general purpose tool. It has been applied to a variety of different problem areas, including:

- Relationships between functions and files in a large software development effort. Here the goal is to understand not only functional relationships between parts of a very large software system, but how changes made to one part impact the other parts. By examining modification history we can create links between files indicating their degree of "co-modification."
- Web site analysis. Navigation is one goal of Web analysis; another is simply to allow the user to understand how a site is laid out. A Web crawler (also known as a "spider") is used to gather the connectivity information. This information is then assembled into a graph and visualized via NicheWorks.
- Correlation analysis in large databases. In looking for patterns in a database consisting of more than 4,000 fields for around 80 million records, it is helpful to have some way of summarizing the variables' relationships to each other. Using NicheWorks to display standardized correlations allows the user to get a first look at an intimidating number of variables and see how they are related. This correlation analysis is strongly related to the graph-based statistical analysis method known as *conditional independence graphs*, or, confusingly, *graphical models* (Whittaker 1990). With NicheWorks we sacrifice the "conditional" part of the dependency information (showing simply raw correlations) for the ability to handle large numbers of variables.

This article is intended to describe both the methodology behind NicheWorks and our general approach to visualizing large, complex datasets; in this case, weighted graphs. Section 2 will give a brief overview of the tool, with Section 3 detailing layout algorithms and Section 4 the interactive interface. Sections 5 and 6 present examples of the tool being used to analyze some real-life data and Section 7 summarizes our findings.

2. OVERVIEW

The NicheWorks tool is part of a suite of visualization views that has been created by the Bell Labs Visualization Group for interactive analysis of large datasets. It shares a number of features with its sibling tools (e.g., SeeSoft; Eick 1994), including:

- Ability to show or hide parts of a graph via manipulation of views of node/edge attributes.
- Tools to color nodes and edges based on their attributes.
- Drag and drop interactive mapping of attributes to shapes and labels.
- Selective labeling of nodes under user control; also painting of nodes with labels.
- Interactive data interrogation via the mouse.

These capabilities are shared by all tools in the linked views environment and their use is discussed in Section 4. There are also methods specific to graph analysis that are incorporated into the NicheWorks view. These include:

- Automatic selection propagation from nodes to edges and vice versa.
- Selection propagation within a graph by following edges (one step or connected component).
- Interactive pan/zoom and rotate facility.

A typical session with a new dataset starts with the data definition stage. Tables of nodes and edges with associated data attributes are loaded into the tool. An initial layout algorithm is selected and the user can map data variables to the available node/edge attributes. The *weight* attribute for edges is the only one that will affect the layout algorithm; the others provide visual information only. The user can also tell the program to regard edges between nodes as directed or undirected.

The user can then select one of several available iterative algorithms to use to improve the layout; the user also allocates the amount of time allowed for these to run. While laying out the data, the user can continue to change attributes, re-draw the graph at intermediate stages, show or hide parts of the graph, and even change the weighting. Finally, the user can save the positioning results for later use.

3. LAYOUT

Among others, Coleman (1996) gave a list of properties toward which good graph layout algorithms should strive. The list includes notions of clarity, generality, and ability to produce satisfying layouts for a fairly general class of graphs. Speed is also a criterion. We want our algorithms to lay out very large general weighted graphs, producing a straight-edge layout that reflects the edge weightings and that places nodes close to other nodes to which they are similar. Di Battista, Eades, Tamassia, and Tollis (1994) gave four aesthetics that are important for the general graph case. Since our layout is straight-edge, we trivially satisfy the aesthetic of avoiding bends in edges. Instead of keeping edge lengths uniform, we wish them to reflect the edge weights; our algorithms generally try to set the edge lengths inversely proportional to the weights, so that the strongest linked nodes are closest together in the ideal layout. Due to the computational cost of multiple edge-crossings detection, we elect to ignore the criterion that edge crossings should be minimized, trusting that our algorithms will produce good results without directly involving a measure of edge-crossings. Since we wish to show clusters of nodes and discriminate nodes far from such clusters, the aesthetic of distributing nodes evenly is not obviously useful, and we relax it considerably; usually adopting only a final polishing algorithm to move overlapping nodes a small amount. A last point worth making is that algorithms that are particularly good at displaying symmetric or planar/near-planar graphs (e.g., Harel and Sardas 1995; Kant 1993) are of limited value in our domain. Large weighted graphs are rarely close to planar or symmetric.

There are two types of algorithms used for laying out graphs in NicheWorks. First, an initial layout (Sec. 3.1) is chosen. This initial layout should be fast, capable of laying out up to a million node graphs in a few minutes, so only simple algorithms should be used. The user may then choose to improve the graph layout with one or more incremental algorithms (Sec. 3.2).

In the following discussion, algorithms are run on each connected component of the graph. The final layout is achieved by placing the components close to each other, with the largest in the center. The algorithm currently in place for this stage is rather naive—components are represented by a circle sufficiently large to encompass the component. The circles are then laid out using a greedy algorithm that places the circles as close as possible to the center of the display in decreasing order of size. Figure 8(a) shows the limitations of this approach. A better approach would be to represent the components by their convex hulls and run an annealing algorithm that would move and rotate the polygons until they fit together more compactly. Since there are usually relatively few components compared to the number of nodes, this stage should take only a small percentage of the total fitting time. This is not yet performed in the current version of the tool.

On an implementational note, we use any available parallel machine architecture to process each component separately. This is trivial to implement as no synchronization is necessary until all the components are placed together at the end.

3.1 INITIAL LAYOUT

The currently available initial component layout algorithms are:

- *Circular layout*—Nodes are placed on the periphery of a single circle.
- *Hexagonal grid*—Nodes are placed at the grid points of a regular hexagonal grid.
- *Tree layout*—Nodes are placed with the root node in the center, then each connected node is put in a circle around that. The spacing between nodes reflects the number of nodes in the subtree originating from that node.

The first two layout algorithms simply place the nodes at random locations either on the circle or on the grid. The tree layout algorithm was inspired by the cone-tree 3-D visualization method for large hierarchies (Robertson, Mackinlay, and Card 1991) but avoids the occlusion problem induced by a 3-D visual system while still easily coping with the size graphs typically displayed in cone tree examples. Di Battista et al. (1994) gave other examples of radial layout algorithms of which this is an example. The tree layout algorithm is designed to work for (surprise) trees, but also works well for directed acyclic graphs (DAGs) and has proved to be useful for both general directed and undirected graphs. In the latter cases we find the source of the graph by working inward

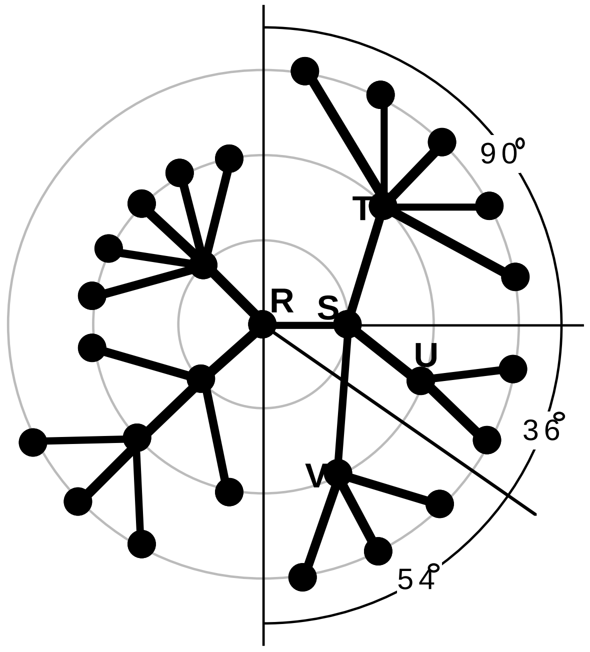

Figure 1. Radial placement.

from the leaves until we find the center-most node(s). If there are multiple sources, the algorithm creates a fake root node as parent to all the real root nodes and lays out this enhanced graph.

The algorithm creates a tree from the enhanced graph by creating a subgraph G', initially consisting of just the root node. An iterative scheme is performed whereby all nodes that are one step away from G' are added to G', along with the strongest weighted edge from that node to G' (i.e., if there are two edges connecting G' to a node, we choose the edge with the strongest weight). This builds up a tree on the graph and terminates when all the nodes are added. A naive implementation of this algorithm runs in $O(DE)$ time. (In this article, N is the number of nodes, E the number of edges, and D represents tree depth.) Each node is then labeled with the size of its subtree (leaf nodes only). A variable indicating subtree angle is also attached to each nonleaf node.

The root node is positioned at the center and given an angle of 360 degrees. This indicates the angular span of its subtree. We then perform the following iterative layout method:

For each of the leaf nodes of the positioned graph, we divide up the angular span available to its subtree using the size of each of its children's subtrees as weights. Thus, if we had a node with angle $20'$ and three children with subtree sizes 3, 2, and 5, the respective angles allotted to them would be 6, 4, and 10 degrees, respectively. The children are placed on a circle with radius proportional to their distance from the root node and are placed at the midpoint of their individual angular ranges, with their parent in the center of the overall range (complying with a common criterion for hierarchical layouts mentioned in, e.g., Coleman 1996). An example is shown in Figure 1. The root node (R) is drawn at the center, with its children on a circle centered at R of radius l. R has a subtree of size 20 and its child S has a subtree of size 10, so S acquires an angular span of $360 * 10/20 = 180$ degrees. Its child T with subtree of size 5 gets a span of $180 * 5/10 = 90$ degrees, U gets $180 * 2/10 = 36$ degrees, and V gets $180 * 3/10 = 54$ degrees.

This process continues until all the nodes have been positioned. The order of placing subnodes around the circle is the final element requiring definition. We have decided on the approach of placing the strongest weighted edges from a node to a child node in the center of the span, with the weaker ones to the edges. Thus, the child with the strongest edge to its parent will be in the center of the range. This works well with a principle used in the iterative algorithms of Section 3.2, that the shortest edges should indicate the strongest weights.

In practice we have found that a slight modification to this algorithm that does not use all the available angle to place children improves the layout. The slight loss in available space is more than offset by the improvement in the visible separation between subtrees.

3.2 INCREMENTAL ALGORITHMS

There are three incremental algorithms available. In each case, the user defines a potential function that describes the disparity between a weighted edge and the length of that edge. The edge length should be inversely proportional to its weight, so that strongly tied nodes are close together. Two of the more useful functions are a sum of terms of the following form:

$$\text{(a)} \quad (1 - dw)^2$$

and

$$\text{(b)} \quad |1 - dw|,$$

where d is the edge length and w is the edge weight. Each potential contribution is minimized when $d = 1/w$. The difference between (a) and (b) can be seen if we add a small perturbation to the optimal solution, making it instead $d = 1/w + \epsilon$ Then (a) gives

$$\big(1 - w\,[1/w + \epsilon]\big)^2 = w^2\epsilon^2,$$

whereas (b) gives

$$|1 - w(1/w + \epsilon)| = w\epsilon.$$

So for a small absolute perturbation of the distance, (a) is more forgiving of minor variations than (b), assuming a transformation of the weights. Note that one natural layout method, multidimensional scaling (MDS) (Kruskal and Wish 1976) is inappropriate as it minimizes

$$(d - 1/w)^2,$$

for which a perturbation of $d = 1/w + \epsilon$ gives potential difference of ϵ^2, completely ignoring the weights, so that more irrelevant weak edges require the same precision of fitting as do strong edges. When an MDS potential function was used in the NicheWorks algorithms, it produced consistently worse layouts for graphs with differing edge weights—the drawback has practical as well as theoretical problems.

An important point to note is that the potential is a function *only* of the graph edges—if two nodes do not have an edge between them, then the distance between them is irrelevant to the potential function. This characteristic ensures that the potential calculations are fast, but has the substantial drawback that there is no force repelling nodes from each other. One outcome of this is that nodes that are unconnected may be drawn very close together if they are connected to similar other nodes with similar weights. In some contexts this is a very useful characteristic. One case is when examining large networks of phone calls to detect fraudulent calls; the grouping of telephone numbers that have similar, unusual calling patterns but do not call each other is highly desirable. In other contexts it is not so useful, and we have provided a specific incremental algorithm to ameliorate the overlapping problem (Sec. 3.2.3).

3.2.1 Steepest Descent

For this method we consider the potential of the graph to be a function of the $2N$-dimensional vector of locations of its nodes. Moving the location of the graph in this high-dimensional space is equivalent to moving every node in the graph simultaneously. We calculate the gradient of this vector and want to move in that direction a suitable amount. To decide how far to move, we take three small trial steps in the direction of the unit gradient, using the potential at those locations to fit a polynomial of degree three to the potential function in that direction. We then solve for the distance that minimizes the fitted polynomial. Then we can move the configuration in $2N$-space to the specified point along the gradient direction. We terminate the algorithm when we cannot move in any direction and improve our fit. The basic method is described for one-dimensional functions in Burden and Faires (1985, chap. 9) and has been modified for our higher dimensional problem.

This method works well when the initial position is close to a local minimum. It does not require the initial configuration to be as close as is required for quadratic methods such as Newton's method, but it can take a long time to reach a good solution, especially for very large graphs. For this reason we suggest that either a good initial placement algorithm should be used, such as the tree layout, or that the simulated annealing swapping algorithm (3.2.2) be run prior to this method.

This method is a relatively slow method, with each step requiring the calculation of several gradient potential functions for offsets from the current location in $2N$-space. Although each calculation is of order $O(E)$ so the order of the whole process is $O(E)$, the constant multiplier is quite high and our informal experience suggests that the number of iterations required to achieve good results is around $O(\sqrt{E})$ giving an overall order of $O(E\sqrt{E})$. Even for a million edges, this is not an overwhelming number, especially since the gradient calculations can all be performed easily with coarse-grained parallelism, providing up to a four-fold speed increase (fitting a polynomial of degree three to the potential function in the optimal direction requires calculating the potential at four different points, each of which can be performed in parallel).

3.3 SIMULATED ANNEALING SWAPPING ALGORITHM

This algorithm is designed to improve a random layout rapidly; it is especially useful for random grid layouts. The algorithm randomly picks a pair of nodes and calculates the difference in potential if the nodes were swapped. If the swap decreases the potential, or if the increase is allowed by the annealing algorithm, then the nodes' positions are swapped.

The annealing schedule is based on the amount of time the user allocates for the swapping routine to be run. Details of annealing algorithms in a graph layout context can be found in Davidson and Harel (1996). They used an annealing approach to decide whether to move a node to a new randomly chosen position, and we use annealing to decide whether to swap nodes, but the process is conceptually very similar and much of their discussion is appropriate for our algorithm. One important difference is in the number of iterations and the cooling schedule. Davidson and Harel (1996) suggested $30N$ iterations, which is impractical for problems of our size. Instead our algorithms prompt the user for the maximal amount of time to run the algorithm, and the annealing algorithm uses the proportion of allotted time taken to reduce the temperature for each iteration.

Calculating the effect on the potential of a swap is linearly dependent on the number of edges involving either node. Sparse graphs are more common for large networks than near-complete graphs, with the average degree of the nodes remaining nearly constant as more nodes are added to the network (this reflects the author's experience with a range of different datasets; mainly telephony, software call graphs and modification histories, and text document associations via n-gram analysis). Thus, the potential calculation is typically very fast and can be performed many times. Because nodes can be moved very long distances with one swap, this method is a powerful way of improving random layouts rapidly.

3.3.1 Repelling Algorithm

The descent algorithm of (3.2.1) can produce layouts with nodes placed very close to each other since it only uses inter-node distances if there is an edge between them. To solve this problem, we introduced a last-stage algorithm—to be run a few times only—which calculates the nearest neighbors for all nodes and then moves the closest ones apart a small distance. Running this a few times will move overlapping nodes apart.

This algorithm uses a quad-tree with an implementation as described by Nievergelt and Hirichs (1993, chap. 23.3); that is, $O(\log N)$ for all three operations of adding, deleting, and calculating nearest neighbors. Thus, each step of the algorithm is $O(N \log N)$, which is completely acceptable.

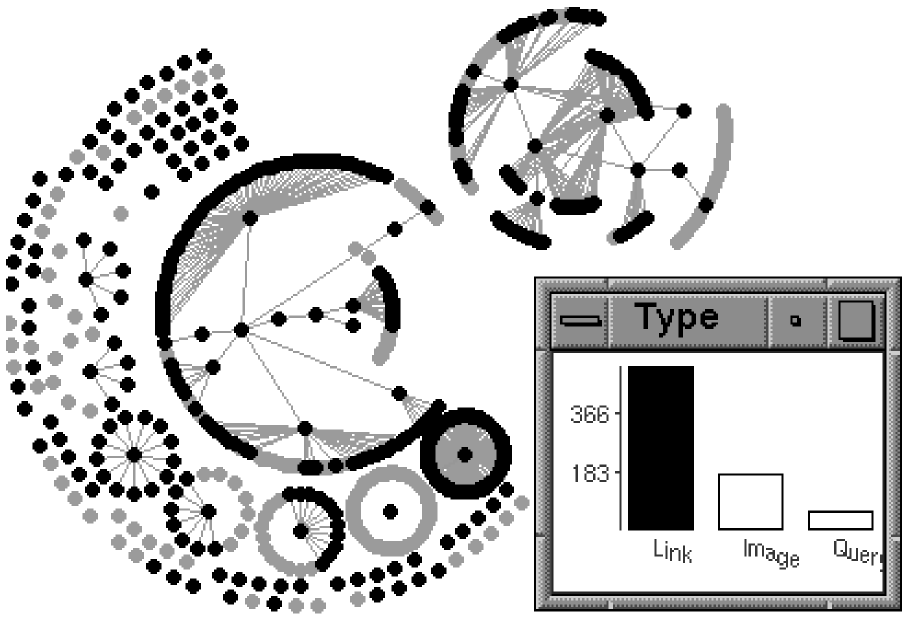

Figure 2. Web site with nodes of type "link" highlighted.

4. INTERACTIVE INTERFACE

The interface to NicheWorks falls under the general classification of a linked views environment, described in detail by Wills (1997) and Eick and Wills (1995) and implemented at least partially in systems such as Velleman (1988); Wills, Unwin, Haslett, and Craig (1990); Swayne, Cook, and Buja (1991); and Tierney (1990). Under this paradigm, each view of the data is required to represent both the data themselves and a state vector that is attached to the data. This state vector indicates how each data point should contribute to the view appearance. In our implementation the possible states are:

- *Deleted*—treat the data point as if it were not present
- *Normal*—show the data
- *Highlighted*—show the data so it will stand out against *normal* data
- *Focused*—show as much detail as possible on the data

Furthermore, the user should be allowed to modify the state vector by interacting with the data views. For example, selecting a specific bar from a bar chart view and highlighting it will change the data state vector for items represented by that bar, causing other views of the data immediately to update their representation.

In NicheWorks there are a number of different options for displaying the graph using the state vector, and for interacting with the graph. The crucial point is that nodes and edges each have a state vector which must be taken into account when drawing the graph. If an edge is *highlighted*, but each of its end nodes are *deleted*, should it be shown, and if so, how? We have not performed solid user trials in this area, but our experience with using the tool has led us to create a set of possible options, some of which are examined

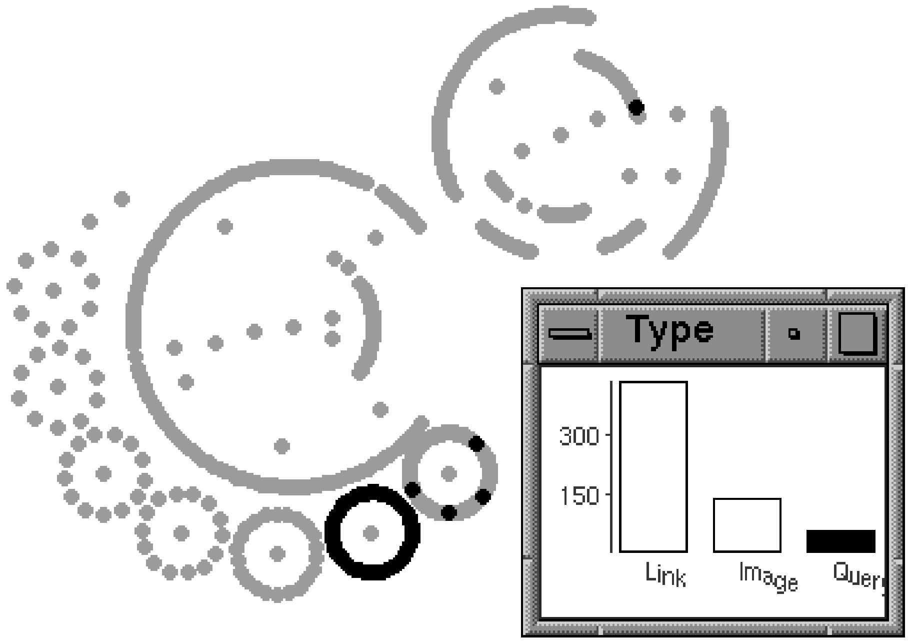

Figure 3. Nodes with degree zero have been deleted *and of the rest, nodes with type "query" have been* highlighted.

in this section. We use an example dataset consisting of a few hundred Web pages and links between them to exemplify the approach. This dataset was collected by listing all the pages near the top level of the author's directory and feeding the references to them to MOMspider (Fielding 1994) which uses references in those documents to search out new pages on the Web.

Figure 2 shows the results of selecting only nodes labeled as "link" (a standard Web page) in the default configuration. Selected nodes are drawn in a highlight color, with unselected nodes in gray. Edges are only drawn from selected nodes to other selected nodes. To create Figure 3, we created a histogram of the degree of each node (not shown) and then used the mouse to select those of degree zero (i.e., those with no edges). We then set their state to "deleted" as we are not interested in these degenerate components. In the "Type" bar chart we then select the "query" type to see which links called query routines.

There is an interesting component where all the child nodes are queries. In Niche-Works we can move the pointer over those nodes so that as the mouse is moved over the data the labels appear and disappear rapidly. We see that all the query nodes are searches into a film database for various films, and the central node is called films96.html—it looks like a page of film reviews.

Compare Figures 4(a) and 4(b) in which we have used NicheWorks to select only one component and then hidden the others. In (b) we have hidden the unselected or *normal* data—we see only those data that are highlighted. This can be very useful when visualizing large graphs as it allows us to focus in on subsets of nodes or edges which

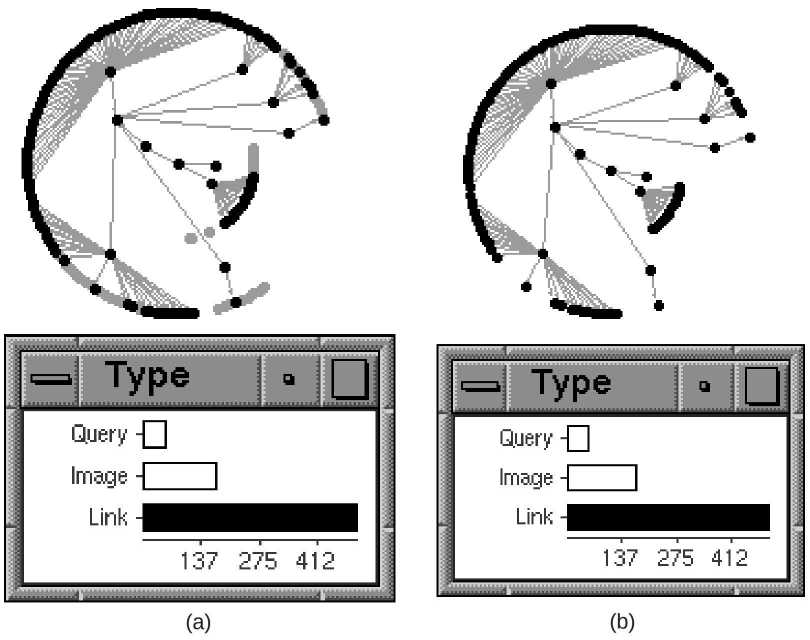

Figure 4. Option to show unselected links in gray on (a) off (b).

fulfill certain conditions. One commonly observed use among novice NicheWorks users in the telephony domain is that they will hide all nodes (telephone numbers) except those that make a large number of calls. After working with the usage patterns of these high-runners, they will then broaden the scope of the exploration.

In Figure 5, we show how fairly complex queries can be posed naturally through the linked views metaphor. In this figure we have changed the "Type" bar chart to a Spineplot, where each bar has a fixed height and the *width* of the bar indicates its count. Within each bar the darker area shows the percentage of selected cases within the bar as a height. We have also created a bar chart of edge counts. This bar chart shows the number of times a given URL refers to another URL (A "URL" is the address of a piece of information on the Web). We have selected all counts above one—that is, all those links to a URL from a URL that occur multiple times. This selection defines a subset of highlighted edges which in turn highlights those nodes that are endpoints of the edges. These are the nodes that either refer to the same URL more than once or are referred to more than once by the same URL. The type bar chart and the NicheWorks view both immediately show this result; we can see that images never have multiple edges to them, regular pages sometimes do and queries do about half the time.

As well as selection mechanisms, the user can use attributes directly to encode information onto the view. In Figure 6 we have zoomed in on a section of the site showing a set of bookmarks and can see the arrowheads as well as the node encoding. This graph has been defined as a directed graph. The nodes' shapes have been coded by type; regular URLs are squares, images are circles, and queries are diamonds. The color represents the action taken by the Web spider on the node. Although they are hard to

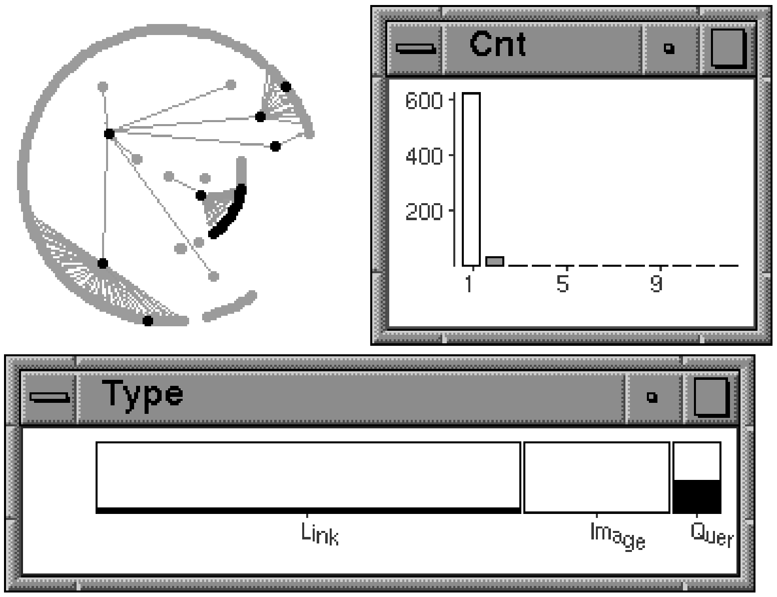

Figure 5. Using edge statistics to highlight nodes and show the distributions of statistics for those selected nodes.

distinguish in this gray-scale representation of the view, the darker gray represents pages that were successfully accessed, and lighter grays represent ones avoided or ones where access to them failed. We have also used the mouse to selectively label nodes.

4.1 SPEED CONSIDERATIONS

Research into human-computer systems has shown that if an operation is performed sufficiently fast (under 200 ms for the average person), then it is perceived as being instantaneously connected to the initiating actions. One important application for a graph-drawing program is that using the mouse to zoom in or out and to pan around or rotate the graph should have the graph being continuously re-displayed within this time period. Failure to do this means the user will have difficulty manipulating the view. Of course, re-drawing a million nodes and edges of varying shapes and colors within 200 ms is impossible on most machines, and by the time it is possible, we will want to visualize a billion nodes and edges. We need some partial-drawing solution.

A naive approach would be to start drawing at time $t0$ and continuously check the time until we reach time $t0 + 200$ ms, then stop drawing. This has two drawbacks. First, and less importantly, polling the time is a notoriously slow operation on most computer systems, and checking more than a few times will degrade performance—exactly what we want to avoid. The other failing is critical; if we adopt the naive approach for a very large, dense graph which has a lot of overplotting, then the partially drawn graph is guaranteed to look very different from the completed drawing. This is because the partial drawing shows only those nodes and edges that lie *below* the ones that are usually drawn

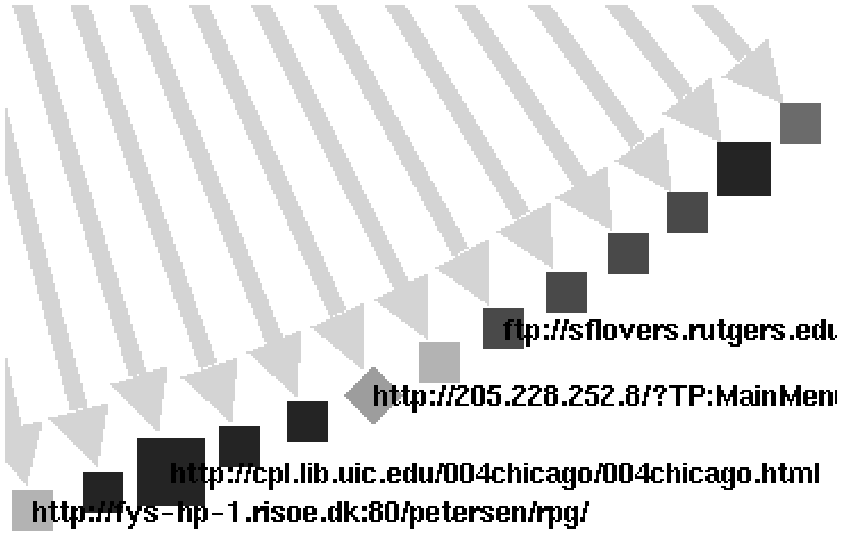
ftp://sflovers.rutgers.edu
http://205.228.252.8/?TP:MainMen
http://cpl.lib.uic.edu/004chicago/004chicago.html
http://fys-hp-1.risoe.dk:80/petersen/rpg/

Figure 6. *Node appearance details.*

later and so appear on top.

For these reasons we adopt a different approach in NicheWorks. Using the results of previous drawing times (number of nodes drawn and time to draw that many) we predict how many items can be drawn within the next 200 ms. We then commence drawing midway through the ordered list of items until we reach the end. This method ensures that the picture is as similar as possible to the complete version.

The details of the prediction algorithm are quite complex, as it needs to be robust, especially to the sudden spikes associated with processor sharing, and yet smooth, as we do not want nodes and edges flickering as we zoom and pan. It must also be adaptive, since as we zoom and pan we are actually drawing different numbers of nodes and edges (the major source of delay time). We intend to deal with this algorithm in detail in another article, but in brief the algorithm uses a robust moving regression model that ignores the most extreme times in the moving average range. It simultaneously calculates several different such models and uses past performance to decide which of the models to choose for the current decision. Empirical results on UNIX X-Windows systems and Windows/DOS boxes indicate that the prediction method works well, rarely overshooting more than 50

4.2 IMPACT ON LAYOUT ALGORITHMS

The state vector can be very useful when laying out large graphs. If we set a node's state to *deleted*, then it plays no part of the layout process, nor do any edges involving it. Thus, we can use the deletion mechanism to look carefully at subsets, trying layouts only for them, or using partial layouts to speed up positioning a very large graph. An example of the former is shown in Figures 7(a) and 7(b). In the former we have selected an important Web page (a bibliography) and use the *one step* menu option twice to expand

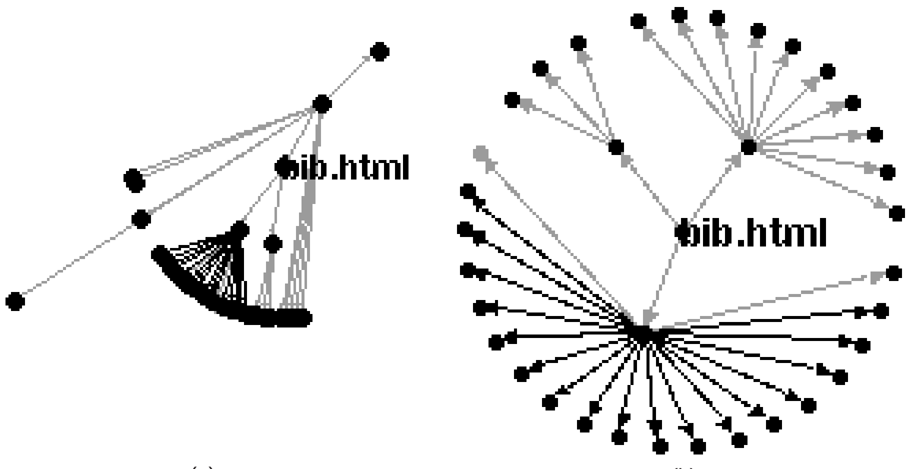

Figure 7. A subset of the Web site positioned as part of the whole site (a) and positioned as if it were the whole graph (b).

the selection to nodes up to two steps away. The result is not very clear, so we *delete* the unselected points and choose the tree layout option to arrive at Figure 7(b), which shows the layout more clearly. We can see that the main page references three sub-pages, each of which reference a number of other pages.

Another use of the state vector is to allow the user to *fix* the selected nodes. These nodes are not permitted to be moved by any of the subsequent layout algorithms. This allows the user to layout one subset of nodes, then fix their positions and use, for example, the descent method to move the other nodes into the best positions relative to the fixed nodes.

A final consequence seems trivial, but is very useful in practice. By showing only the most important nodes and/or strongest edges, the user can watch the layout algorithm perform without taking too much time away from the algorithm to display the data. This helps the user understand the algorithm's operation more clearly and thus aids the layout process.

5. EXAMPLE: WEB SITE VISUALIZATION

The MOMspider Web crawler was used to search and index all Web pages accessible from the author's home page (with a few termination conditions limiting the depth of the tree and preventing access to off-site pages). The resulting information totaled 733 pages (nodes) and 758 links between pages (edges), not including the home page itself, which was only used as a source of originating pages. A number of statistics were collected on both nodes and edges, including number of times a link was referred to in a page. This statistic was used for the weight. In this section we demonstrate how we use NicheWorks to understand the structure of this fairly small site.

Figure 8 shows the layouts for each of our three methods. We also ran the swapping algorithm for ten seconds on the hexagonal grid to achieve the layout. Each view shows

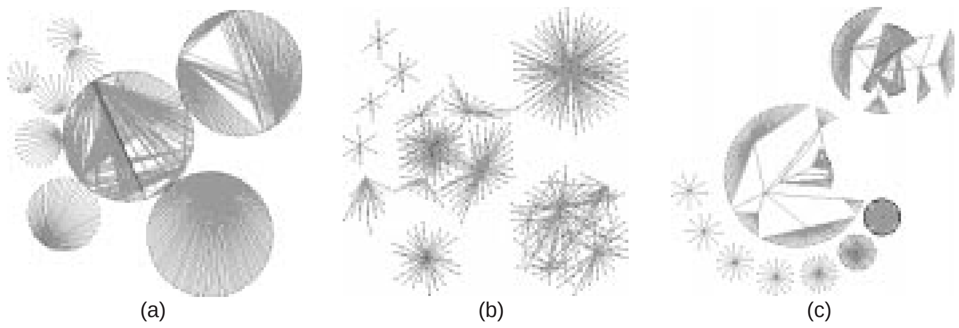
Figure 8. Circle (a), hexagonal (b), and tree (c) layout methods for Web site data.

nine separate components of differing sizes. The circle layout does not look very promising as it shows the size of the clusters well, but not their structure. The tree layout hides the size to an extent (e.g., consider the very dense cluster toward the bottom right) to show structure better. The hexagonal grid method shows a bit of each. We run the move algorithm for 10 seconds on the most promising two—the hex and tree layouts—to give Figures 9(a) and 9(b). As might be expected, the two layouts appear fairly similar as far as individual components are concerned. The tree layout followed by move appears best; we will use it from now on.

There is an immediately noticeable pattern in several of the components; a central node with connections to every other node in the component and no other edges. These are collections of information with one index page referencing many others. Figure 6 in Section 4 shows part of one of these components; it is a list of bookmarked pages. Although most of these "central node" components in Figure 9(b) are symmetrical, there is one that is very asymmetrical at the top left. We zoom in on it and selectively label the center node to produce Figure 10. When we set node shapes to indicate the type of page or run the mouse over the outer nodes, we see that they are all queries for a database

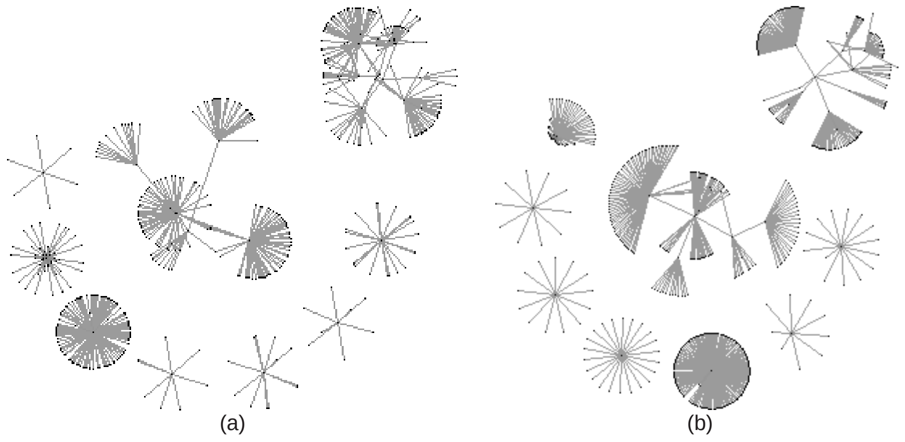
Figure 9. Results of the move algorithm for hex layout (a) and tree layout (b).

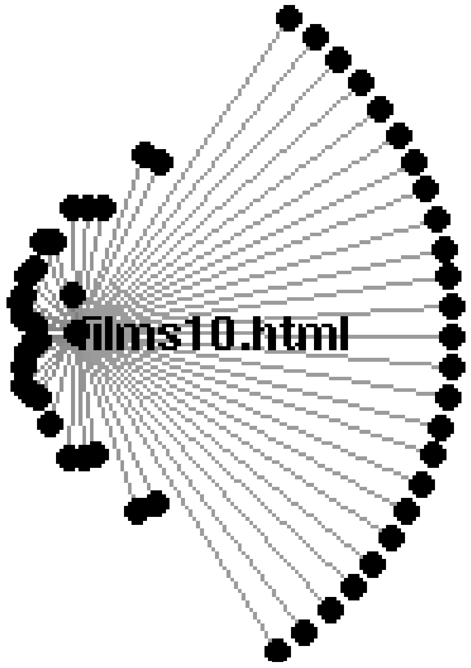

Figure 10. An asymmetrical component.

server, each query being a request for a film name. The different line lengths indicate that some films are referred to more frequently than others from the central page.

This component was created to index a list of best films of 1996. A script was run that identified all names in block capitals and added a link to the film database to the document. Some films were mentioned only once (the ring of far away circles on the right), others more often. Since simple components like this can be solved for a zero potential, the distance from the central node is exactly inversely proportional to the number of times the film is mentioned in the article. We use the mouse to focus on the innermost nodes and note that *The English Patient* and *Fargo* are the most commonly mentioned films.

We zoom in on the large central component in Figure 9(b) and label some representative nodes to give Figure 11. The central page here is the home page for the author's department (Software Production Research Department Home Page), with a ring of general purpose pages around it, most of which are off-site and so are not accessed by the MOMspider program. Two interesting exceptions are the "who.html" page, with its list of images of people and links to their home pages, and a set of pages for ordering books on-line using the local "bookbot" system.

Exploring Web networks is an emerging field. It is important for administrators who want to ensure that there is a consistent and logical pattern in their site, and for tracking visits to individual pages to see what types of path people are following. It is also important for users navigating a site to see what is out there. The size of the networks and their ad-hoc complexity make it a natural candidate for this form of network visualization.

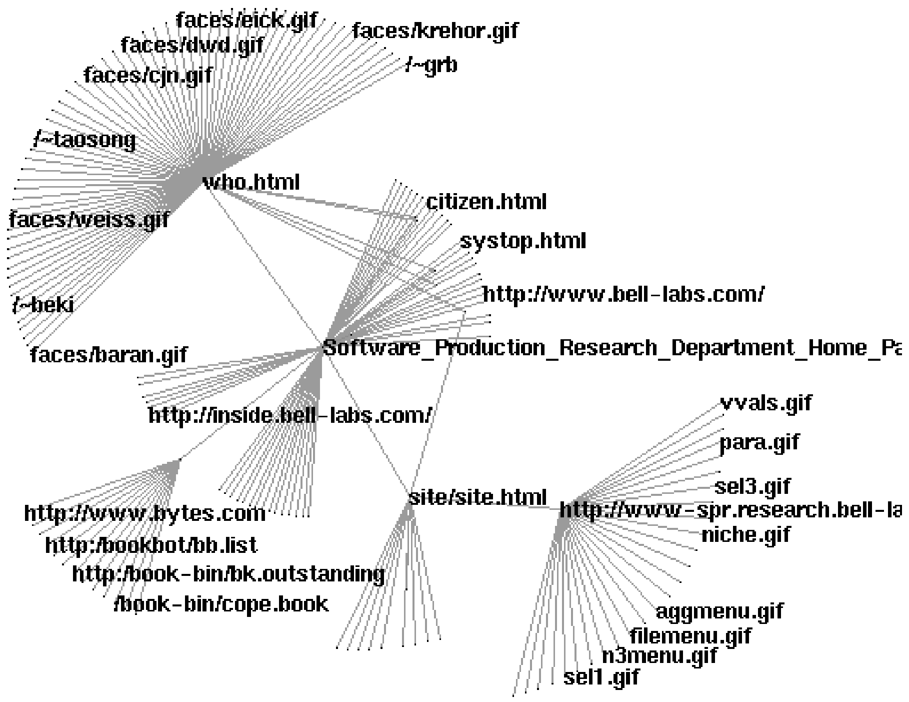

Figure 11. Department home page component. The user has hand-labeled some nodes representing interesting or representative Web locations.

By adding information about the nodes (type, number of hits, number of modifications, size, content flags) and edges (multiplicity, number of follows) the form of visualization provided by NicheWorks allows the structure of the site to be understood in the context of the data about it.

6. EXAMPLE: INTERNATIONAL CALLING FRAUD

Discovering fraudulent calls made to overseas locations is very important for telephone companies. Not only do they lose revenue if they cannot collect money for a call, but they also have to pay overseas telephone companies to use their equipment, resulting in an actual loss of money. But telephone fraud is very resilient to automatic or data mining techniques for detection. Fraudsters adapt very rapidly to new algorithms and share their knowledge on preventive systems. For this reason a visual approach to fraud detection is very effective; it exploits users' ability to interpret and understand new patterns in calls, allowing them to process more data and see new fraudulent methods. The dataset in this section originally consisted of more than 20 million international telephone calls over a weekend in 1994. This has been segmented by geographical origination, and here we are investigating a moderate 40,000 calls involving slightly less than 35,000 callers.

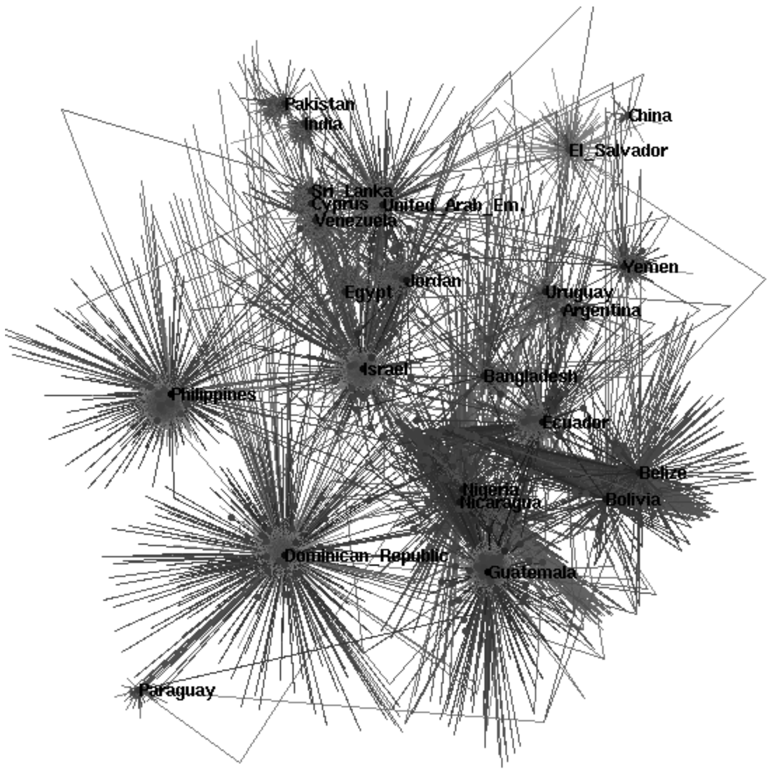

Figure 12. Overview of calling patterns.

Figure 12 shows the results of the hex layout/swapping/movement algorithms. The most noticeable feature is the number of callers clustered around each country; these are people who called exactly one country, with the distance to the country indicating for how long they were on the phone to it. The analyst then decides to look only at people who made multiple calls (other possibilities might be to look at long calls only or to look at calls to certain known international fraud hot-spots) and also trims the graph by specifying only callers that spent more than a certain significant time on the phone. After playing around interactively with various thresholds (static parameters are easily evaded by experienced fraudsters) and various layout methods, Figure 13 was arrived at. The analyst noted an interesting couple of callers who called both Israel and Jordan a lot. By using the mouse to focus on the callers, and viewing the results in a linked view, it appeared that each of the callers made more than 120 calls to Israel and more than 80 to Jordan. Zooming in (Figure 14) and labeling the nodes showed that not only were their calling patterns very similar, but they also had very similar numbers (which have been scrambled in this article). It is almost certain that these callers are part of the same operation.

The fraud method they might be engaged in is a fairly elaborate one. At the time of the study, it was impossible to call directly between Israel and Jordan. The fraudsters would set up a phone account in a rented apartment in the U.S. and charge Israeli and

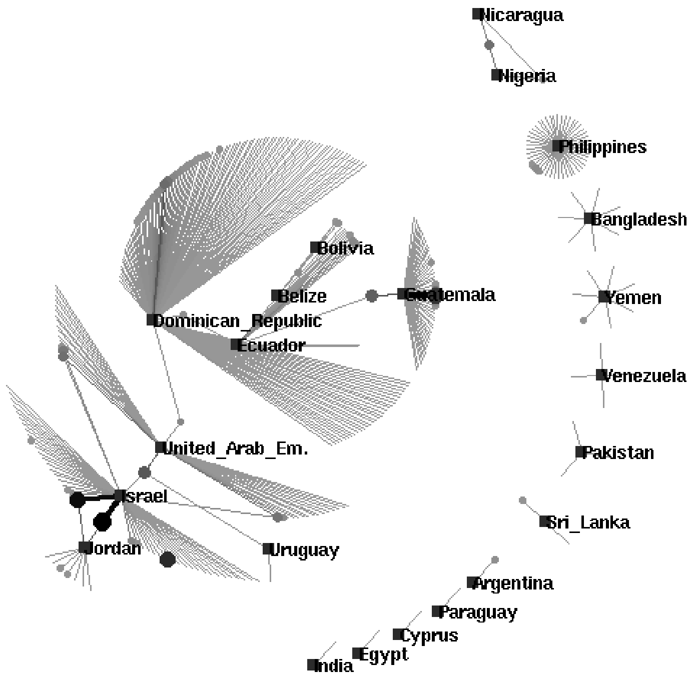

Figure 13. High users' calling patterns.

Jordanian business people for third-partying the call through to the other country. When their bills came in at the U.S. end, they would simply ignore them and leave to set up a new location. The distribution of durations of calls made by these numbers is consistent with the investigator's expectations for this type of fraud.

Looking at these callers in detail, the analyst noticed that they also called the United Arab Emirates (UAE) occasionally. Following this line of inquiry, the analyst showed all calls and callers involving any of these three countries (around 12,000) and produced Figure 15(a), and, zooming in, Figure 15(b). By labeling interesting numbers the analyst identified one number that called Israel for a long time and, importantly, a group of other telephone numbers having similar patterns of calling to the original pair and having numbers similar to each other. This set of numbers had much lower call volume, partly because there were more phone numbers involved, and so were less easy to detect. Since standard methods would not have noticed that the numbers were similar, they would not have been able to identify this type of fraud.

Using the NicheWorks tool, the analyst was able to explore a medium-sized network of more than 40,000 calls, using different criteria to selectively show different aspects of the data and using different positioning methods to display the results to best effect. The linking mechanism was used both to control the visible and highlighted data and as feedback when interrogating the data using the mouse. In the end, the analyst was able to spot a pattern based on a fairly easily detectable pattern of fraud (due to the high volume) and customize the graph to show more possible incidences of this type of illegal activity. This analysis was performed on an IRIS Indigo2, with interactive selection and

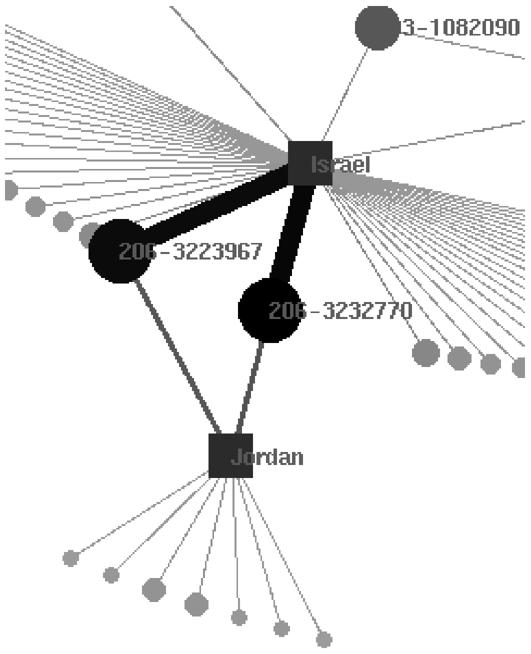

Figure 14. Possible fraud pattern involving Israel and Jordan.

display results and positioning algorithms taking a maximum of 10 minutes for the large overviews.

7. CONCLUSIONS AND FUTURE WORK

Displaying and navigating very large networks is a hard problem. With current and foreseeable limitations on display size and resolution, it is clear that labeled views of complete large networks are impossible with static layouts. One approach which has been tried is to use the two dimensional screen as a view on a three-dimensional layout (e.g., Robertson, Mackinlay, and Card 1991; Sheelagh, Carpendale, Cowperthwaith, and Fracchia 1996), but these methods have not shown themselves effective for large networks; indeed, since the layout can now be viewed from many angles, the problem is worse in 3-D. Our approach is to provide a tool that allows the user to interact with the weighted graph, making it possible to position and focus rapidly on different subsets of the whole, thus building up knowledge about the entire graph. In our opinion, methods for the following operations are essential:

- Defining a subset of the graph based *both* on graph structure and on values of any available node/edge variables.
- Providing a range of robust layout tools suitable for different types of graph.
- Laying out subgraphs.

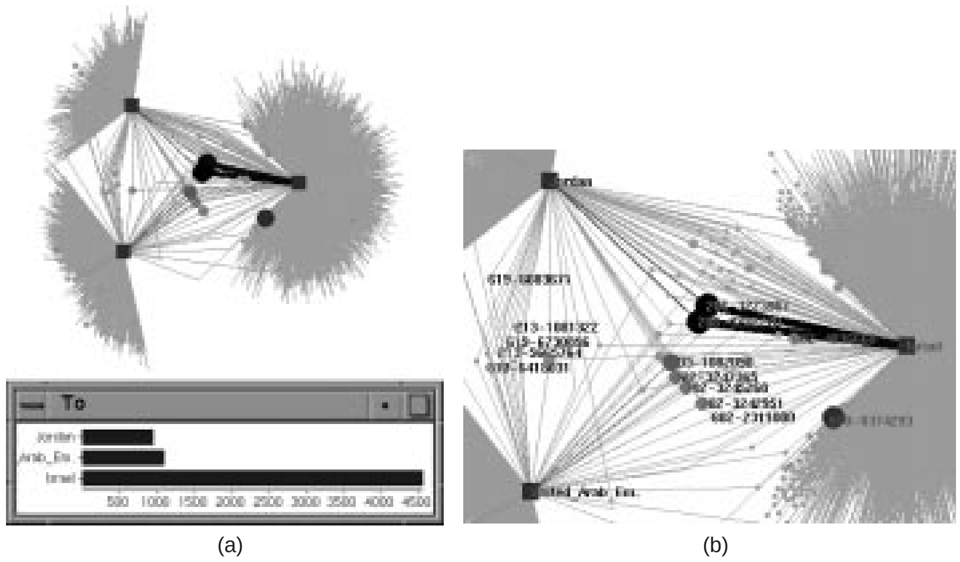

Figure 15. The Israel-Jordan-UAE generated subset (a) overview (b) zooming in to those callers calling more than one country.

- Giving immediate and reversible control over mappings from data attribute to node and edge attributes (color, size, shape, line style, etc.).
- Interactive labeling.
- Free and rapid ability to pan, zoom and rotate the graph in the viewing window. The speed must be sufficient to make the action appear truly interactive.
- Means for the user to retrieve full details on nodes and edges.

We have described the NicheWorks tool and given an overview of the linked windows environment in which it is embedded. The linked windows metaphor, in which a state vector is attached to data tables for nodes and edges and is shared among different views of the data, allows selection, deletion, and focus operations on data to be visible in the graph structure and vice versa. The NicheWorks layout algorithms have been designed to work well for large, weighted graphs and have been implemented so as to be robust against slight data irregularities; for example, the tree layout method uses the “best” tree within the graph, rather than assuming that the data must be a tree. The NicheWorks graph view itself creates a self-calibrating predictive model for the amount of drawing possible within a given time to allow maximally efficient, interactive panning, zooming, and rotation. A number of selection tools and methods are available which have been described elsewhere (Wills 1996).

The goal of NicheWorks is to allow the user to interact with large graphs; to allow them to try ideas, focus on different aspects of the data, and to create views that spark intuition. We have used perceptually good attribute encodings (Cleveland and McGill 1984, 1988) and expanded results by Tufte (1983, 1990) and Bertin (1983) into the interactive domain. One of the most pleasing aspects of the project is that domain experts with little or no graph-theoretic or statistical background can use it to gain knowledge about graphs in their own area.

NicheWorks is an evolving tool; although the basic idea has remained stable for several years, it is continually being improved and updated. It is available as a component of a library of C++ linked data exploration tools for both Windows and UNIX. It is also available as an ActiveX object. The commercial version is called “Data Constellations” and is supported by Visual Insights (a Lucent Technologies venture).

Some recent thrusts which are in a prototype stage and some future areas we wish to incorporate include:

- *Networks with hierarchies*. The ability to represent a set of nodes by a super-node and collapse and expand them under interactive control is useful both when such a structure is predefined and when the user interactively or semi-automatically imposes one. Web pages and sites form such a hierarchy, as do software files, modules and programs, international telecommunication data, and many other transactional record systems. Methods of creating hierarchical information include graph clustering and hierarchy definition via node variables. Early efforts to visualize hierarchical data were described by Eick and Wills (1993).
- *Time series*. Especially with transactional data, it is often necessary to look at networks in the context of when edges were created, measuring evolution and structure changes in graphs.
- *Additional layout algorithms*. Improving our existing algorithms and implementing promising new ones which can be adapted for large graphs will always be important.
- *Parallel implementations*. Several of our algorithms and many in the literature are suitable for a simple, coarse-grained style of parallelism available on many platforms. Without parallelization, datasets of up to about 100,000 nodes and links can be explored slowly, and data sets of around 20,000 nodes and links can be explored with ease on a moderate UNIX workstation. With six processors, layout is four to five times faster. A fine-grained parallel implementation could take advantage of many processors; this has not been attempted to date.

NicheWorks allows users to visualize weighted networks with hundreds of thousands of nodes and edges. It combines statistical data views with graph layouts and visualization methods from the computer science disciplines using an interactive linked views environment. It is currently being used for a number of tasks including software analysis, fraud detection, and document correlations. We welcome all comments and suggestions as we continue to improve it to be better, faster, and ultimately more *informative*.

ACKNOWLEDGMENTS

The author thanks the referees for their helpful comments

[Received November 1997. Revised May 1998.]

REFERENCES

Bertin, J. (1983), *Semiology of Graphics*, Madison, WI: University of Wisconsin Press.

Burden, R., and Faires, J. D. (1985), *Numerical Analysis* (3rd ed.), Boston, MA: PWS Publishers, Duxbury Press.

Cleveland, W. S., and McGill, R. (1984), "Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods," *Journal of the American Statistical Association*, 79, 531–554.

——— (eds.) (1988), *Dynamic Graphics for Statistics*, Pacific Grove, CA: Wadsworth & Brooks.

Coleman, M. K. (1996), "Aesthetics-Based Graph Layout For Human Consumption," *Software Practice And Experience*, 26, 1415–1438.

Davidson, R., and Harel, D. (1996), "Drawing Graphs Nicely Using Simulated Annealing," *ACM Transactions On Graphics*, 15, 301–331.

Di Battista, G., Eades, P., Tamassia, R., and Tollis, I. (1994), "Algorithms For Drawing Graphs: An Annotated Bibliography," *Computational Geometry*, 4, 235–282.

Eick, S. (1994), "Graphically Displaying Text," *Journal of Computational and Graphical Statistics*, 3, 127–142.

Eick, S., and Wills, G. (1993), "Navigating Large Networks with Hierarchies," in *Proceedings of IEEE Visualization '93*, pp. 204–210.

——— (1995), "High Interaction Graphics," *European Journal of Operations Research*, 81, 445–459.

Fielding, R. (1994), "Maintaining Distributed Hypertext Infostructures: Welcome to MOMspider's Web," in *Proceedings of the First International Conference on the World-Wide-Web*, Geneva.

Harel, D., and Sardas, M. (1995), "Randomized Graph Drawing With Heavy-Duty Preprocessing," *Journal Of Visual Languages And Computing*, 6, 233–253.

Kant, G. (1993), "Algorithms for Drawing Planar Graphs," unpublished Ph.D. thesis, Utrecht University.

Kruskal, J., and Wish, M. (1976), *Multidimensional Scaling*, Sage University Series on Quantitative Applications in the Social Sciences, Beverley Hills and London: Sage Publications.

Nievergelt, J., and Hirichs, K. (1993), *Algorithms and Data Structures with Applications to Graphics and Geometry*, Englewood Cliffs, NJ: Prentice Hall.

Robertson, G., Mackinlay J., and Card, S. (1991), "Cone Trees: Animated 3D Visualizations of Hierarchical Information," in *Proceedings of the ACM Conference on Human Factors in Computing Systems* (CHI'91), pp. 189–194.

Sheelagh, M., Carpendale, T., Cowperthwaite, D., and Fracchia, F. D. (1996), "Distortion Viewing Techniques for 3-Dimensional Data," in *Proceedings of IEEE InfoVis '96*, pp. 46–53.

Swayne, D., Cook, D., and Buja, A. (1991), "XGobi: Interactive Graphics In The X Window System With A Link To S," in *American Statistical Association Proceedings Of The Section On Statistical Graphics*, 1–8.

Tierney, L. (1990), *Lisp-Stat: An Object-Oriented Environment for Statistical Computing and Dynamic Graphics*, New York: Wiley.

Tufte, E. R. (1983), *The Visual Display of Quantitative Information*, Cheshire, CT: Graphics Press.

——— (1990), *Envisioning Information*, Cheshire, CT: Graphics Press.

Velleman, P. (1988), *The Datadesk Handbook*, Odesta Corporation.

Wills, G. (1996), "Selection: 524,288 Ways to say 'This is Interesting'," in *Proceedings of IEEE InfoVis '96*, pp. 54–60.

——— (1997), "Visual Exploration of Large Structured Data Sets," in *New Techniques and Technologies for Statistics II*, Washington, DC: IOS Press.

Wills, G., Unwin, A., Haslett, J., and Craig, P. (1990), "Dynamic Interactive Graphics For Spatially Referenced Data," in *Fortschritte Der Statistik-Software 2*, Stuttgart: Gustav Fischer Verlag, pp. 278–287.

Whittaker, J. (1990), *Graphical Models In Applied Multivariate Statistics*, Chichester: Wiley.