NicheWorks - Interactive Visualization of Very Large Graphs

Graham J Wills
Room 1u334, Lucent Technologies (Bell Laboratories),
1000 E. Warrenville Road, Naperville IL 60566, USA
Email: gwills@research.bell-labs.com, Web: www.bell-labs.com/~gwills

Abstract

The difference between displaying networks with 100–1000 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. A system for investigating and exploring such large, complex data sets 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. We describe a tool that addresses these needs, the NicheWorks tool. We describe and comment on the available layout algorithms and the linked views system, and detail an example of the use of NicheWorks for analyzing web sites.

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. 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 multi-dimensional views.

Originally designed for telephony applications, a number of other data sources have been analyzed NicheWorks has been adapted and modified into a general purpose tool. It has been applied to a variety of different problem areas, including:

- **Relationships in a large software development effort**. The goal is to understand functional relationships between pieces of a large software system and see how changes to one part impact 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.

- **Correlation analysis in large databases**. In looking for patterns in a database consisting of many variables, it is helpful to have some way of summarizing the variables' relationships to each other. We display standardized correlations to get a first look at the variables and see how they are related.

2. Overview

The NicheWorks tool is part of a suite of visualization views that have been created by the Bell Labs Visualization Group for interactive analysis of large data sets. It shares a number of features with its sibling tools (e.g. SeeSoft [Ei94]), including:

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

Methods specific to graph analysis that are incorporated into NicheWorks 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.

This paper describes the methodology behind NicheWorks and our general approach to visualizing large complex data sets; in this case, weighted graphs. Section 3 details layout algorithms and section 4 the interactive interface. Section 5 presents an example of the tool for some real-life data and section 6 summarizes our findings.

3. Layout

Among others, Coleman [Co96] gives a list of properties towards which good graph layout algorithms should strive. 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 [Ba94] gives 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 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. There are two types of algorithms used for laying out graphs in NicheWorks. First, an initial layout (section 3.1) is chosen. The user may then chose to improve the graph layout with one or more incremental algorithms (section 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 naïve: 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. 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

Initial layouts are not yet a strong point of our work; suggestions by this paper's referees should prove helpful for future work. Currently, the available initial component layout algorithms are fairly simple, comprising:

- **Circular Layout**. Nodes are placed on the periphery of a single circle
- **Hexagonal Grid**. Nodes are placed at the 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 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 3D visualization method for large hierarchies [Ro91], but avoids the occlusion problem induced by a 3D visual system while still coping with the size graphs typically displayed in cone tree examples. [Ba94] gives other examples of radial layout algorithms of which this is an example. The tree layout algorithm also works well for 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 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'. A naive implementation of this algorithm runs in O(DE), where D is the tree depth and E is the number of edges. Each node is then labeled with the size of its subtree. A variable indicating subtree angle is also attached to each non-leaf 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 it subtree using the size of each of its children's subtrees as weights. 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, for example, [Co96]). 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 1. 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.

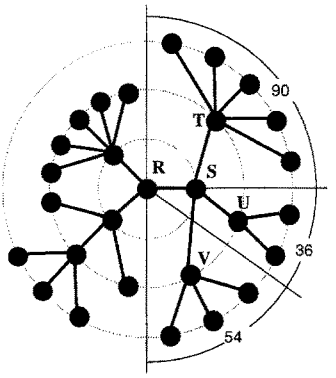

Figure 1. Radial placement

3.2 Incremental algorithms

There are three incremental algorithms available. In each case, the user defines a potential function which 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:

$$\text{a)}\ (1-dw)^2 \qquad \text{b)}\ |1-dw|$$

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 $d = 1/w + \varepsilon$

Then we get $\text{a)}\ \left(1-w\left[\frac{1}{w}+\varepsilon\right]\right)^2 = w^2\varepsilon^2$ $\qquad \text{b)}\ |1-w(1/w+\varepsilon)| = w\varepsilon$

so for a small absolute perturbation of the distance, (a) is more forgiving of minor variations than (b), assuming a transformation of the weights.

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 (it is the primary reason for the efficiency of our approach), but has the drawback that there is no force repelling nodes from each other.

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 move in that direction a suitable amount. Then we can move the configuration in 2N-space to the specified point along the gradient direction. The basic method is described for one dimensional functions in [BuFa85].

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 $O(E\sqrt{E})$.

3.2.2 Simulated Annealing Swapping Algorithm

This algorithm randomly picks a pair of nodes and calculates the difference in potential if the nodes were swapped. If the potential increase is allowed by the annealing algorithm, then the nodes' positions are swapped. Details of annealing algorithms in a graph layout context can be found in Davidson and Harel's paper [DaHa96]. They use an annealing approach to decide whether to move a node to a new position and we use annealing to decide whether to swap nodes, but the process is conceptually very similar.

3.2.3 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 a 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 in [NiHi93] 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 acceptable.

4. Interactive interface

The interface to NicheWorks is an instance of a linked views environment, described in [Wi90], [Wi97] and [EiWi95]. In this paradigm, each view of the data displays both the data themselves and a state vector that is attached to the data. This vector

dictates how each datum 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.

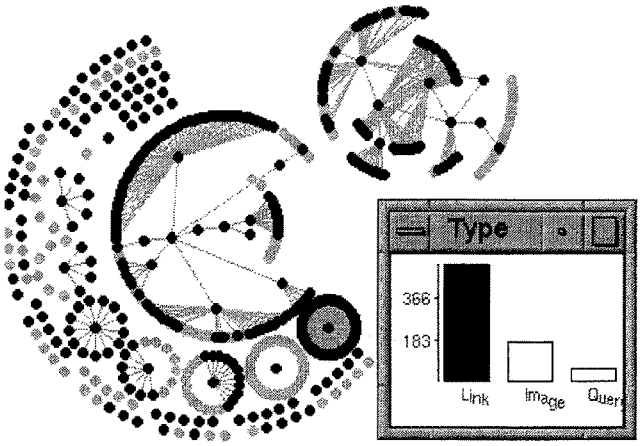

Figure 2. Web site with nodes of type 'link' highlighted

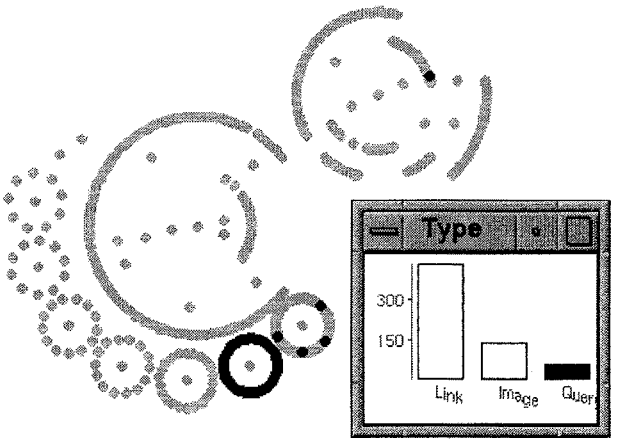

Figure 3. Nodes with degree zero have been *deleted* and of the rest, nodes with type 'query' have been *highlighted.*

In NicheWorks there several options for displaying the graph using the state vector, and for interacting with the graph. We use a small data set consisting of a few hundred web pages and links between them to exemplify the approach. This data set was collected by listing all the pages near the top level of the author's directory and feeding the references to them to MOMspider [Fi94] 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). 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. 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. We move the pointer over those nodes so that 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.

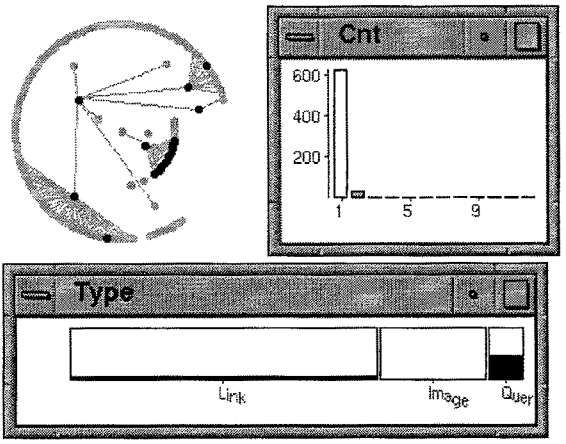

Figure 4. Using edge statistics to highlight nodes and show the distributions of statistics for those selected nodes

In figure 4, we show how fairly complex queries can be posed naturally through the linked views metaphor. 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, showing the number of times a URL refers to another URL. We have selected all counts above one; i.e. 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. 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.

The state vector can be 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 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 5(a) and 5(b). In the former we have selected an important web page (a bibliography) and use the *one step* menu option twice to expand 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.

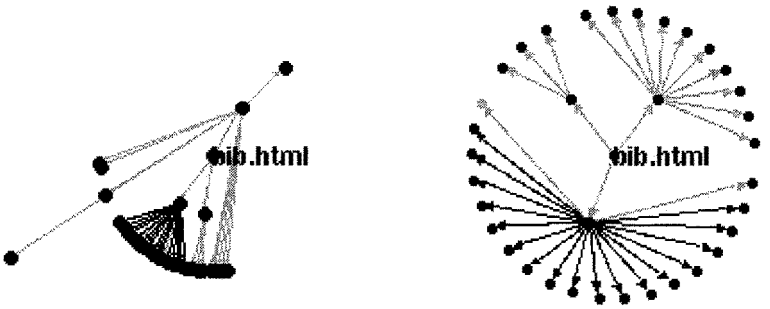

Figure 5. A subset of the web site positioned (a) as part of the whole site (b) by itself

5. Example: Web Site Visualization

The MOMspider web crawler was used to search and index all web pages accessible locally from the author's home page. The resulting information totaled 733 pages (nodes) and 758 links between pages (edges). 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 small site.

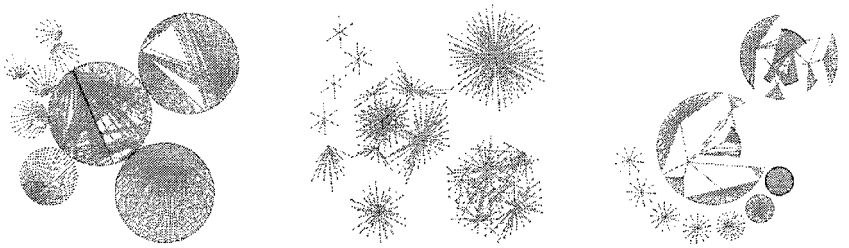

Figure 6. Circle, Hexagonal and Tree layout methods for web site data

Figure 6 shows the layouts for each of our three methods. We ran the swapping algorithm for ten seconds on the hexagonal grid to achieve the above layout. Each view shows 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 (for example consider the very dense cluster towards 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 7(a) and 7(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'll use it from now on.

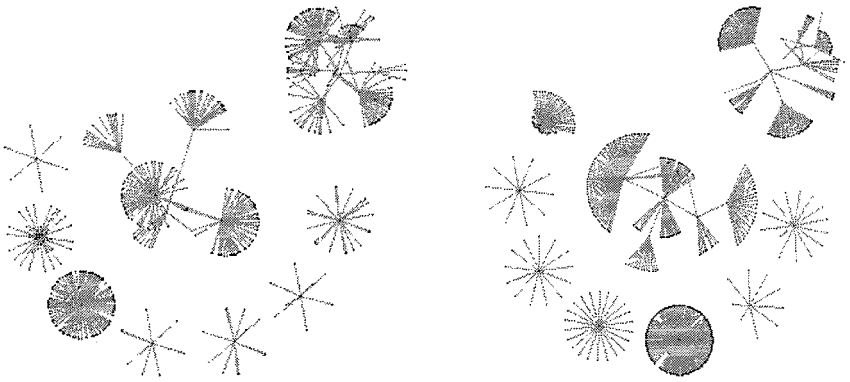

Figure 7. Results of the move algorithm for (a) Hex layout (b) Tree layout

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. Although most of these 'central node' components in figure 7(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 8. By interrogating, we see that they are all queries for a database 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.

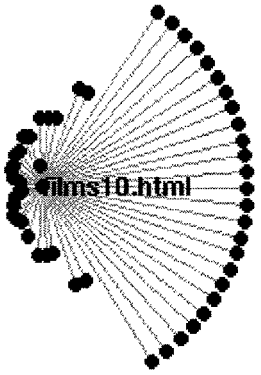

Figure 8. An asymmetrical component

This component was created to index a list of best films of 1996. 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 focus on the innermost nodes

and note that "The English Patient" and "Fargo" are the most commonly mentioned.

We zoom in on the large central component in figure 7(b) and label some representative nodes to give figure 9. The central page here is the home page for the author's department, with a ring of general purpose pages around it, most of which are not accessed since they go off-site. 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 both site administrators and for users navigating a site. The size of the networks and their ad-hoc complexity make it a natural candidate for this form of network visualization.

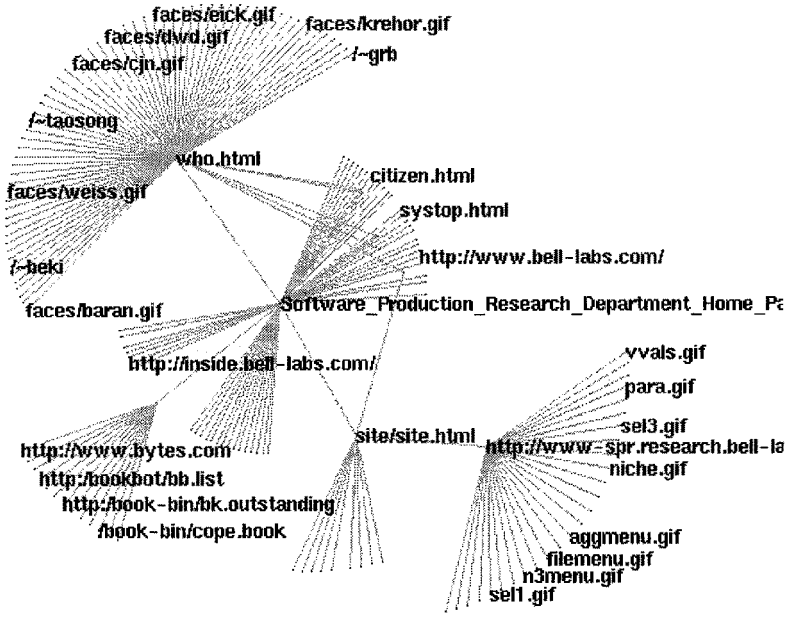

Figure 9.. Department home page component with user-labeled nodes

6. Conclusions and Future Work

Displaying and navigating 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.. 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.
- Giving immediate and reversible control over mappings from data attribute to node and edge attributes (labelling, olor, size, shape, line style, etc.)
- 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 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. A number of selection tools and methods are available which have been described elsewhere [WI96]. 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 [ClMc84, ClMc88] and expanded principles of Tufte [T83, Tu90] and Bertin [Be83] into the interactive domain (via the linked windows environment, section 4). 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 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.*

7. References

Ba94 Di Battista, G., Eades, P., Tamassia, R. and Tollis, I. (1994) *Algorithms For Drawing Graphs: An Annotated Bibliography* Computational Geometry 4 (1994) 235-282

Be83 Bertin, J. (1983) *Semiology of Graphics* University of Wisconsin Press

BuFa85 Burden, R. and Faires, J.D. (1985) *Numerical Analysis (3rd ed.)* PWS publishers, Duxbury Press, Boston MA 02116

ClMc84 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 pp 531-554

ClMc88 Cleveland, W. S. and McGill, R., eds. (1988) Dynamic Graphics for Statistics Wadsworth & Brooks, California

Co96 Coleman, M. K. (1996) *Aesthetics-Based Graph Layout For Human Consumption* Software Practice And Experience, 1996, Vol 26(12), Pp 1415-1438

DaHa96 Davidson, R. and Harel, D. (1996) *Drawing Graphs Nicely Using Simulated Annealing* ACM Transactions On Graphics, Vol 15, No. 4, 1996, Pp 301-331

EiWi95 Eick, S. G. and Wills, G. (1995) *High Interaction Graphics* European Journal of Operations Research #81 (1995) pp. 445-459

Ei94 Eick, S.G. (1994) *Graphically displaying text* Journal of Computational and Graphical Statistics, 3(2), pp. 127-142

EiWi93 Eick, S. and Wills, G. (93) *Navigating Large Networks with Hierarchies* Proceedings of IEEE Visualization '93

Fi94 Fielding, R. (1994) Maintaining Distributed Hypertext Infostructures: Welcome to MOMspider's Web Proceedings of 1st intl. conf. on the World-Wide-Web, Geneva

NiHi93 Nievergelt, J. and Hirichs, K. (1993*) Algorithms and Data Structures with Applications to Graphics and Geometry* Prentice Hall, Englewood Cliffs, NJ 07632

Ro91 Robertson, G. G., Mackinlay J. D., and Card, S. K. *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. ACM Press, 1991.

Tu83 Tufte, E.R. (1983) *The Visual Display of Quantitative Information* Graphics Press, PO Box 430, Cheshire, Connecticut 06410

Tu90 Tufte, E.R. (1990) *Envisaging Information* Graphics Press, PO Box 430, Cheshire, Connecticut 06410

Wi97 Wills (1997) *Visual Exploration of Large Structured data Sets* New Techniques and Technologies for Statistics II, IOS Press, Washington DC

Wi96 Wills, G. J. (1996) *Selection: 524,288 Ways to say "This is Interesting"* Proceedings of IEEE InfoVis '96, pp 54-60

Wi90 Wills, G. Unwin, A., Haslett, J. and Craig, P. (1990) *Dynamic Interactive Graphics For Spatially Referenced Data* Softstat '89 Fortschritte Der Statistik-Software 2, Gustav Fischer Verlag, Stuttgart, Pp 278-287