

Navigating Large Networks with Hierarchies

Stephen G. Eick and Graham J. Wills

AT&T Bell Laboratories

This paper is aimed at the exploratory visualization of networks where there is a strength or weight associated with each link, and makes use of any hierarchy present on the nodes to aid the investigation of large networks. It describes a method of placing nodes on the plane that gives meaning to their relative positions. The paper discusses how linking and interaction principles aid the user in the exploration. Two examples are given; one of electronic mail communication over eight months within a department, another concerned with changes to a large section of a computer program.

I. THE PROBLEM

It has almost become a cliché to start a paper with the observation that the amount of data in the world is growing rapidly, and that current efforts to extract useful information from data lag far behind the ability to create data. However the cliché is true, and no less so in the field of network analysis and visualization than in any other. In many areas, scientists are realizing that the tools they have been using are limited in utility when applied to large, information-rich networks. Not only are networks of interest large in terms of size (as measured by number of nodes or links between nodes), but also in terms of the data collected for each node or link. The ability to examine statistics on the nodes and relate them to the network is of crucial importance.

Examples of areas in which the analysis of large networks is important include:

i. *Trade flows.* The concern in this area is monitoring imports and exports of various products at several levels; international, interstate and local. Besides examining many types of trade goods, there is also strong interest in spotting temporal patterns.

ii. *Communication networks.* This is an important and wide category, covering not only telecommunication networks, but also electronic mail (email), financial transaction, ATM/bank data transferal and other data distribution networks.

iii. *Software Engineering.* Large software projects have a number of networks of interest associated with them. One such network is the *function call graph*, a network indicating which functions call which other functions. The network of inter-file dependencies states which files must be re-compiled when any file is altered. A good understanding of these networks allows efficient compilation strategies to be designed as well as aiding the planning of new additions to the system.

Previous research into network visualization includes Unwin, Sloan and Wills [12] which describes an exploratory analysis of oil imports/exports to and from European countries. The nodes are positioned on a map of the world at the center of their countries and a number of interactive methods, such as thresholding, are used to help analyze the data. Because there are relatively few countries in the world their methods work well, but even with the interactivity there would be problems applying this model to large data sets. Further, the use of geographic location is a limiting concept. Although it has the useful property of telling the user where the countries lie in relation to each other geographically, it provides no other information. Thus it devotes much space to showing well-known, unchanging facts.

Becker et al. [2] gives another method of interactively exploring network data. The paper examines telecommunications traffic, focusing on telephone traffic within the US during the October 17, 1989, earthquake. Geographic location is again used to position the 100 or so nodes, and the links are animated over time with their width and color changing to reflect various traffic statistics. The positioning is a major problem here as the focus of interest is California which is on the edge of the picture. Thus in all the displays we see many lines all crossing the US horizontally and becoming visually confused. The authors propose a number of techniques to alleviate this problem such as hiding small links, hiding nodes and drawing only parts of lines linking nodes. Although helpful, these do not really attack the core problem; the positioning of the nodes conveys too little

information for both the screen ‘real-estate’ it uses and the clutter it creates.

In the next section we consider a section of the corporate email network and examine it with *HierNet*, a tool the authors have written for examining large hierarchical networks. Email is an interesting area of study, especially for large organizations, because dissemination of information and the synthesis that comes from good communication patterns is important for organizational efficiency. Understanding email networks has benefits when designing interfaces and electronic conferencing tools, allocating people to tasks, and investigating how people interact within a given domain. Figure 1 shows the communication patterns in our department at AT&T Bell Laboratories. In this snapshot of a screen display we have encoded information in several ways:

- The area of each node is proportional to the number of email messages sent or received by the individual (or machine id). The largest node, *MTS/Sys*, represents 1600 messages.
- The node color* shows the function; red for clerical, blue for technical staff, pink for technical staff from other departments, green for department heads (managers), yellow for machines.
- The links show email communication between individuals, where the common ‘heat’ scale has been used to code number of messages; blue for few, through green and yellow to red for many messages.

This display is appealing. We can see immediately that the departmental MTSs cluster at the center, surrounding the two main communication foci; the main secretary and the system administrator. (*MTS/Sys*). The departmental heads are well out from the center, with the other secretary. At the right are many external MTSs and at the lower left are a higher proportion of machines. One explanation for this last effect is that the departmental MTSs below and left of the *MTS/Sys* node do more applied work than those in the other areas, who are more theoretical in nature. The external MTSs generally tend also to be more theory-oriented, leading to the above pattern.

We characterize the domain we will investigate as that of *large* networks, with (possibly) *statistics* associated with each node and/or each edge. By large we mean ≥ 500 nodes, with millions of nodes not being an unreasonable amount. A *statistic* we take to mean some piece of

information about the node or edge. This may be numeric, categorical, or simply a label. A further observation we can make is that the nodes of large networks often have a hierarchy imposed on them; for trade flow and travel networks we may take towns as a basic unit that we group into regions and similarly group regions into countries; communications networks are often constructed as hierarchical units with central servers or hubs. We will see that the existence of such a hierarchy allows us develop methods for the visualization of large networks.

Our approach to analyzing such data has three major lines of attack:

- *Node placement*. Instead of using a pre-defined (e.g., geographic) location, or using a layout made purely to reduce clutter, we use algorithms that position the nodes by considering the weighted links between the nodes. These methods place nodes that are strongly linked close together, giving meaning to relative locations. As a pleasant side effect, clutter is also reduced.

- *Interaction*. In both [2] and [12], interaction is used to let the user change viewing parameters such as node size, line width and length, etc. We use interaction for a similar purpose, but amplify its effect by combining it with linking.

- *Linking*. Scatterplot brushing [1] is the oldest method of linking data plots, but the idea has been generalized to linking many types of plot [13]. We provide several simple displays and allowing the user to interact with each to select a focus of interest. The other displays then update to reflect such a focus, allowing interesting features found in one plot to be related to other displays. This aids navigation of large networks by allowing the user to concentrate on interesting parts of the data set.

II. THE E-MAIL NETWORK

We now take a longer look at the email data set. We are interested in a number of questions such as

Q1. Do similar people send similar amounts of mail? To similar numbers of people?

Q2. Who talks with whom? Are there communities with similar interests?

Q3. How do these patterns change over time?

Mackay [8] provides a number of other interesting questions and hypotheses about email use. The data have been collected by instrumenting the UNIX mail program. The authors compiled a version of *mail* which logged each message as it was dealt with by the mail program; recording the sender and recipient and the date at which it was sent. A number of days of data were collected and have been aggregated into months for ease of processing.

* In the black and white figure, “color” is taken to mean gray level.

Figure 2 shows a view of the data as it is being analyzed using *HierNet*. *HierNet* is an X-windows application [9] with some Motif extras [4] which provides an environment in which we can visualize hierarchical networks. There are three windows displayed; the main window, which shows the network itself, a narrow window ('time') which shows the period under scrutiny, and a smoothed histogram of the number of messages each network link represents. The display represents the state of the network in the last month of the period studied (as shown by the position of the red box in the 'time' slider). The labels for the nodes are fictitious surnames, to preserve anonymity. Figure 1 shows the network at the previous month. The sizes of the nodes have been forced to be constant over all time periods to help the user spot patterns. Hence the three largest nodes are Friedman (MTS/Sys), Hardy (MTS) and Dreyfus (MTS). They are close together in both displays – an indication that they form a community with a common interest.

The 'link strengths' window shows the distribution of the number of messages per link, where the horizontal axis is mapped to the link strength and the vertical axis to the number of links with the given strength. This is smoothed using a *kernel density smooth* [10] to give a useful representation. This window can be interacted with in two ways. First, the degree of smoothing can be altered by moving the slider to the right of the window. The smoothed view is updated constantly and rapidly as the smoothing parameter is changed, allowing the user not only to choose a suitable value, but also to see different features. For example, consider Figure 3, in which we have shown four different levels of smooth. In the topmost display we have the highest level of smooth; the global features of the data are all that are visible. This shape conforms to a statistical distribution called the *Poisson* distribution – a plausible model for this kind of data [11]. This is an over-smoothed view, however, and as the degree of the smooth becomes less and less, it becomes clear that there are distinct clumps of link strengths, with the size of the clumps diminishing as they represent bigger and bigger links. Although a Poisson model could be considered as an initial model, a more realistic model would have to explain this phenomenon.

The other interaction method supported by this view is the ability to select sub-areas of the smoothed histogram and either show or hide them. Since this view is linked to the main view, the main view will show or hide links as this selection is made. In Figure 2 we have hidden all except the links with many messages. This gives a picture that shows the main communication pathways more clearly than Figure 1, but does not show any links for Wilde, the fourth largest node with over 600 messages to his credit in the month. Wilde has a different pattern of

email usage to the other large nodes, sending few messages to many people rather than many messages to a few. He is also not part of the highly communicative group of Friedman, Dreyfus and Hardy.

This pattern is true for the last two time periods. Is it true for earlier time periods? Has the group of three always been together? We can use the time slider to move through the history of the network and observe the results, as follows. In Figure 4, the main view shows the earliest state of the process. The circle of nodes represents those email ids that have not yet 'entered the picture'. Only Friedman is in the department at this time. As we progress forward in time, we see the nodes begin to move in from the edges and Dreyfus (the next to move) moves immediately beside Friedman. At a later date, Hardy and Wilde both move into the picture. It takes them two time periods to reach the positions shown in the inset – close to where they end up in Figure 3. In fact the group of three stay together for the rest of the period, but Wilde moves around a lot. Generally, we observe that it takes about two months for most people to move to a reasonably stable position and that Friedman, Hardy and Dreyfus form a stable group whereas Wilde moves around a lot.

In summary, we have learnt that for this network:

A1. The amount of mail sent varies widely, but smoothly as shown by the node sizes in Figure 1. A smoothed histogram view of the node sizes confirms this. There is considerable variety in how the messages are sent, for example in the difference between Wilde and the other nodes. The view of link strengths shows that mail traffic between people is more clumped than might have been suspected.

A2. We have identified one community of three individuals. These are people with excellent computer and UNIX skills, who we may hypothesize regularly exchange email related to computer issues.

A3. This group formed as soon as the three settled into the organization and remained together from then on. Conversely, Wilde has no fixed communication patterns. By examining individuals as they join the department we can see that it takes about two months for most to establish communication patterns and settle in to their new position.

III. SOFTWARE ENGINEERING

A more complex example is shown in Figure 5. The data are based on a large software project, the 5ESS switch, the source files of which are allocated to various *modules*, and the modules themselves to subsystems. This forms a large hierarchy of files; the subsystem we investigate contains about a hundred modules with 6000

files. Displaying such a hierarchy alone is a formidable task. Tree displays become cluttered with a hundred nodes and even advanced techniques such as [5] have difficulty with over a thousand nodes. Figure 5 shows the modules and the links between them, calculated as explained below.

When a change to the program is required, an MR (*modification request*) is created, and every change to the source code required to implement the change is registered under that MR. Thus the MR database contains a history of modifications to source files. We generate links between files A and B equal to the sum of MRs which changed both files. Hence each link is a measure of ‘co-change’ and large links denote files that are changed together a lot. This method generates 8 million links. To manage this many links, we make use of the hierarchical information. Instead of looking at all file-file links, we form 3 smaller sets of links to summarize the important properties of the network:

- We retain all *file-file* links between files within a module. This gives us a model of how each module is changed internally.
- We sum all links between a file and files in another module to give a *file-module* link showing how files co-change with other modules.
- We sum all links between files in one module and files in another to give a *module-module* link denoting how changes in modules affect other modules.

Although we lose links between files and files in other modules, the file-module links capture the important aspect of those links; the ‘breaking’ of module independence, and we gain the informative module-module links. We reduce the number of links to about a quarter of a million, a manageable amount. In Figure 5 and 6 we retain only the strongest 1% of such links, reducing clutter. Note also that this hierarchical aggregation is scaleable. If we had many subsystems we can form module-subsystem and subsystem-subsystem links by taking the module-module links as a base and repeating the above method.

The interest in this data set lies in understanding what modules and files need to be changed at the same time. They are likely to form sets that have a similar function or which depend on common structures. Being able visually to indicate such classes and see how they interact is of importance in understanding how to divide modules among teams, form new modules that may interact with existing modules and re-organize the hierarchy for greater efficiency. Another important area is that of re-engineering; looking at the way the system has evolved and trying to model the process and re-implement it in better ways.

For Figure 5 we have used two statistics to code the nodes’ size, shape and color. The number of files in the module gives the width of the node, and the number of changes made to files in the module is used to give the height. The node is colored to emphasize the aspect ratio of the resulting rectangle. Tall thin red nodes indicate modules with a large number of changes per file, whereas wide short green nodes indicate where the ratio of changes to files is low. All the links involving files are hidden in this Figure – we focus on comparing modules only.

There are several interesting features in this plot; we will indicate just two. The most immediate feature is the circular pattern to the lower right of center. There are about 25 small modules in a circle or inside a circle, and all are linked to all the others with the same weight. These links produce the blue spike in the smoothed histogram of link strengths. By dragging the mouse across the nodes it can be seen that each of the modules is named CC7sigv??, where ‘??’ is a pair of numerals, and that each module contains exactly two files. Clearly there is an important association among these files. The authors consulted a developer who had worked in this area, who explained that all the modules had been created simultaneously and that they contained no executable code, but were simply descriptions of how certain data structures should be instantiated. These files are thus logically separate, since only one is used for each version, but are highly related in terms of function. The use of change history to spot this feature is in contrast to methods based on semantic analysis of the code, which could not have found such a relationship.

A second feature of interest is the group of six files at the far right, slightly above center. They are linked in a fairly strong way, but have only a weak connection to other modules. This shows a successful attempt at functional separation, resulting in a set of modules that does not need to change when other parts of the subsystem change. Investigation found that this part of the system dealt with ordinary telephone signaling, whereas the rest of the system dealt mainly with ISDN telephone communication.

One useful interactive technique that takes advantage of the hierarchy is the ability to expand and collapse levels of the hierarchy. After studying a module in terms of its links to other modules, we can replace it by its component files and see the file-file and file-module links for them. This can lead to some valuable insights. In Figure 6 the gray insets indicate where we have expanded a module into its component files. The large red module to the left has many files and its color indicates a high number of changes per file. When we expand it and zoom in, we see that there are few links shown; when changes are made to files, other files within the module are not

affected. Furthermore, the absence of links leading out from the files to other modules means that changes do not necessitate many changes in other modules. This is an example of a module that should be reasonably easy to maintain.

In contrast, the other module we have expanded (the wide green one in the center of the right box) has a large number of files (562) and fewer changes per file. When we expand it, though, we do not even need to zoom to see that it is radically different. Many links come from the files of this module to other modules. This seems to be a contradiction. How can the individual files be linked strongly to other modules, but the module itself not strongly linked to other modules? The reason is that most of the files are linked to different modules, so that the average link from this module to another is actually small, since it depends only on links involving a few files. This module is not really a module at all. Its name 'hdr' gives a clue what function it performs. It is a collection of files describing the other modules, and hence its files change with those other modules.

For this data set we have:

- Used the hierarchy to reduce the size of the data set significantly without reducing the information content and allowed the user to interact with the hierarchy to see levels of information, hiding links and nodes to allow important features to be seen.
- Found a large group of modules that are practically identical. This information can be used to abstract the information in the modules further and produce smaller modules with greater independence.
- Located a group of modules performing a function independently of the others – useful information when allocating modules to developers; these should not be split up.
- Identified an anomalous module; one whose files are linked with most other modules.

IV. NODE PLACEMENT ALGORITHMS

In section I, we stated that we use algorithms that position nodes based on the weighted links between the nodes. In this section we outline ways of doing so and explain our preferred algorithm. A number of link-based placement algorithms exist. Many do not take the weights of links into account, being concerned only with connectivity [6], and others are aimed purely at creating as pleasant a layout as possible, for example by minimizing the number of links that cross each other. These methods are inappropriate in an interactive environment as the number of links displayed varies as the user desires. They also do not give any real meaning to the relative positions of nodes: What does a cluster of nodes mean when the positioning criterion is to minimize crossed links? One

further drawback is that most of them have not been designed for use with large networks.

We therefore examine methods to position nodes such that the distance between nodes is related to strength of the link between them. Clearly such a relationship should be an inverse one, with nodes that have large links having short separation. The simplest form of such a relation is to require that the distance between two nodes be inversely proportional to the size of the link:

$$d_{ij} \propto \frac{1}{w_{ij}}$$

where d_{ij} is the displayed link length, w_{ij} is the size (weight) of the link.

The obvious candidate for such a placement algorithm is the statistical technique of *multidimensional scaling*, or MDS [7]. Given a matrix of dissimilarities, δ_{ij} , this method fits positions to the nodes that minimize $\sum (d_{ij} - \delta_{ij})^2$. Setting $\delta_{ij} = 1/w_{ij}$ would give a layout that should attempt to satisfy the above relationship.

Unfortunately this does not work well for two reasons. The practical reason is that the method requires the inversion of the matrix of dissimilarities, which can be expensive for large systems such as we have explored. Another drawback can be seen by considering the quantity to be minimized:

$$\begin{aligned} \sum (d_{ij} - \delta_{ij})^2 &= \sum (d_{ij} - \frac{1}{w_{ij}})^2 \\ &= \sum \frac{(d_{ij}w_{ij} - 1)^2}{w_{ij}^2} \end{aligned}$$

The method tries to set $d_{ij}w_{ij}$ to be close to unity (as required), but the minimization is weighted by $1/w_{ij}^2$, with the result that small links have an inordinate importance relative to large ones – exactly the reverse of the situation we would like. For these reasons we cannot use MDS as a node placement algorithm.

Another method that has been used in a number of guises is that of directly minimizing an error function on the nodes. Using the function $\sum (d_{ij} - \delta_{ij})^2$, for example, gives a method equivalent to MDS. To implement minimization methods we start with some reasonable positioning of the nodes and use a method such as Newton's method or steepest descent to find a layout that minimizes the function (see [3] for details). Minimization methods are also known as *force-based* algorithms, as they are equivalent to releasing a set of nodes under a system of inter-node forces and letting them find an equilibrium layout. A common example of such an algorithm is to model the nodes as having springs attached between them, the strength of each spring being proportional to the size of the corresponding link. Some repulsive force between

nodes is also required to prevent the nodes from coalescing.

Although widely used, the spring algorithm is not a perfect model. It certainly brings nodes closer together when the links are stronger, but it has a tendency simply to bring nodes which have a lot of strong links together, regardless of whether they have strong links to each other. It also tends to fill an area, since all nodes want to get as close to each other as the repulsive force will allow. Thus nodes will tend to fill in gaps. This behavior is useful in that it fits as many nodes as possible into an area, but it tends to hinder the ability to spot groups, as groups are always embedded in a sea of less important nodes that have drifted beside them.

We elect instead directly to minimize the function

$$\sum (w_{ij} - \frac{1}{d_{ij}})^2 = \sum \frac{(d_{ij}w_{ij} - 1)^2}{d_{ij}^2}$$

This gives us the required relation, and also states that our interest lies most strongly in close nodes, where the weights are high. In contrast to the spring method, nodes with weak links are moved to positions well away from the other nodes, so that important groups are readily identified.

The hierarchy is also taken advantage of in the layout algorithm. Starting with the root node (which we place at some appropriate center, usually the origin), we consider the sub-network generated by its children and use our minimization algorithm to place them. We then re-center and re-scale to place the children at the parent's location and recursively use this algorithm on the child nodes. This gives us a layout algorithm that can deal with any depth of hierarchy, and which is comparatively fast as the slow part (the minimization routine) is only required to work on sub-networks.

V. CONCLUSIONS

We have demonstrated our approach on two real data sets; the email data set – with a small number of nodes and temporally indexed links, and a software engineering example – with a medium number of nodes and many links. For both we showed how the beginning of an exploration might start, drawing useful insights into how people communicate and how big programs are changed.

We have demonstrated a method of investigating large, hierarchical networks which is based on three lines of attack:

- Since the focus of the visualization is the layout of nodes and edges, a good *placement algorithm* is vital. We have described an algorithm that is fast, even for large networks, interpretable and which yields appealing and informative results.

- Because we are threatened with an excess of information, we use *linking* to allow us to focus on specific features of the data while retaining an overall context and providing ancillary information.

- We use *interaction* to allow the user to adjust parameters and edit the visualization, query for hidden information and directly manipulate the display.

With a good basic method and facilities for interaction, we are able to give the user the ability to navigate and explore large networks. The user can see small effects in the context of large trends and can see the evolution of networks over time and relate added information at the nodes and links to the overall display. We believe this approach to be a valuable method for the exploration of large, information-rich, hierarchical networks.

VI. REFERENCES

- [1] Becker, R.A. and Cleveland, W.S. (1987). *Brushing Scatterplots*. Technometrics, Vol.29, No.2, pp. 127-142.
- [2] Becker R.A., Eick, S.G., Miller, E.O. and Wilks, A.R. (1990). *Dynamic Graphics for Network Visualization*. Proceedings of IEEE Visualization Conference, 1990.
- [3] Burden, R.L. and Faires, J.D. (1985). *Numerical analysis*. Duxbury Press, Boston, MA.
- [4] Heller, D. (1991). *Motif Programming Manual*. O'Reilly and Associates, Sebastopol, CA.
- [5] Johnson, B.J. and Shneiderman, B. (1991). *Tree-maps: A space-Filling Approach to the Visualization of Hierarchical Information Structures*. Proceedings of IEEE Visualization Conference, 1991.
- [6] Koutsofias, E. and North, S.C. (1992). *Drawing Graphs with Dot*. Bell Laboratories technical report.
- [7] Kruskal, J.B. and Wish, M. (1979). *Multidimensional Scaling*. Sage Publications, Newbury Park, CA.
- [8] Mackay, W.E. (1988). *Diversity in the Use of Electronic Mail: A Preliminary Inquiry*. ACM Transactions on Office Information Systems, 6, 1988.
- [9] Nye, A. (1991). *Xlib Programming Manual*. O'Reilly and Associates, Sebastopol, CA.
- [10] Silverman, B.W. (1990). *Density Estimation for Statistics and data Analysis*. Chapman & Hall, New York.
- [11] Taylor, H.M. and Karlin, S. (1984). *An Introduction to Stochastic Modeling*. Academic Press, London, UK.
- [12] Unwin, A.R., Sloan, B. and Wills, G.J. (1992). *Interactive Graphical Methods for Trade Flows*. Proceedings of New Techniques and Technologies for Statistics, to appear.
- [13] Velleman, P.F. (1988). *The DataDesk Handbook*. Odesta Corporation. Ithaca, NY.

Figure Captions:

Figure 1. Email network for a small department

Figure 4. Observing the behaviour of the email period of time.

Figure 2. The *HierNet* program being used to analyze the email data.

Figure 5. Using *HierNet* to show groupings within a large software project.

Figure 3. Interactively smoothing a histogram; four degrees of smoothing.

Figure 6. Expanding and collapsing nodes within the hierarchy to see more or less detail.