HIERARCHICAL INFORMATION VISUALIZATION USING ENCCON MODEL

Quang Vinh Nguyen and Mao Lin Huang
Faculty of Information Technology
University of Technology, Sydney
Australia

Abstract

This paper describes a new efficient approach for visualizing large hierarchical information. Our technique is based on the *connection + enclosure* visualization model [1] from which the area division is used for the recursive positioning of nodes, while a node-link diagram is still drawn to present the entire hierarchical structure. We inherit the advantages of Space-Optimized (SO) Tree technique [1] that can enhance the usability of display space by using area division. However, we replace a set of polygons used in SO Tree by a set of rectangles for the area division. This not only decreases the computation cost in calculating geometrical polygons, but also greatly reduces the human perceptual and cognitive loads spent on understanding the underlying hierarchical structure. We use semantic zooming technique to enlarge a particular viewing area and filter out the rest of structure that is less interested. The navigation is accommodated by animation in order to preserve the mental map [2].

Key Words
information visualization, tree visualization, node-link diagram, tree-maps, hierarchical information, navigation.

1. Hierarchical Data Visualization

With the rapid growth of information, the size of data sets increases significantly each year, and to find out efficient information visualization techniques for the viewing, understanding, navigation and manipulation of such large data sets has become one of the crucial tasks. In fact, most advanced high-quality visualization techniques often rely on the capacities of super Computer Graphics Workstations with high display resolution and CPU power which are usually expensive. Therefore, it is important and economical to develop optimized visualization techniques that can be applied on ordinary PCs or Unix stations that are often available in normal labs or personal offices. Although several techniques [2, 3, ..., 17, etc] have been proposed and implemented to deal with large relational hierarchies, only few are good candidates in term of space efficiency, low run time and less complexity.

Researches in hierarchical visualization can be roughly classified into two main streams: *connection* and *enclosure*. They are both effective approaches for the visualization of hierarchies and which one we should use depends primarily on the properties of the data in a particular application domain. The connection approach is effective for hierarchies that have uneven shapes while enclosure approach is effective for trees where the nodes include quantitative variables, particularly when large values are important. Now we would like to discuss in more detail of both connection and enclosure approaches.

1.1. Connection Approaches

These are natural ways of presenting graphs/tree structures by using a *node-link* diagram. A set of visible graphical edges are drawn in the diagram to link nodes from the parents to their children. The nodes present the data while these edges are used to present relationships among data items. There are many researches in this direction have been done, such as *cone-tree* [3], *hyperbolic browser* [4, 5], *radial view* [6], *balloon view* [7], *disk tree* [8], *classical hierarchical view* [9], *h-tree layout* [10], *botanical visualization* [11], *NicheWorks* [12], *Rings* [13], *Narcissus* [14], *etc.* The advantage of using a *node-link* diagram to present hierarchical structures is that the viewer can directly see these relationships that are drawn as a set of graphical edges appearing in the diagram. This makes it easier for the process of human perception to understand the relational structures of the information. These techniques, however, are not often efficient in term of utilizing display space (see [1] for detailed discussion).

1.2. Enclosure Approaches

Another way to visualize hierarchical data is to use enclosure. Unlike connection, enclosure is the method of using enclosure to represent the tree structures. Figure 1 is an example of an enclosure technique: *tree-maps* [15]. This technique maps each node into a rectangular area, then that area is subdivided in horizontal or vertical direction to show the relative size of the children of the node. The process is recursively applied to the child nodes with the subdivisions on the X- or Y-axis. There are also some researches in this direction that have been well done. Typical examples of this type of visualization techniques are *tree-maps* [15], *cushion tree-maps* [16], *squarified tree-maps* [17] and *Venn diagram* [18]. Although enclosure techniques are more optimal in term of using displaying space, they do not show directly the relational structures of information. This costs extra cognitive effort of viewers in understanding the relational structures that are presented in the enclosure manner (see the discussion in [1]).

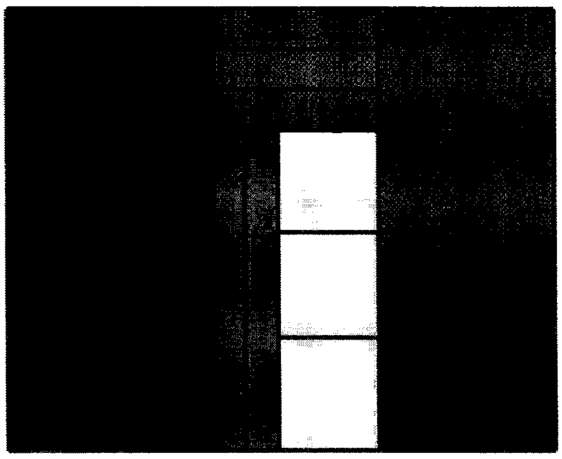


Figure 1. An example of Tree-maps.

1.3. Connection + Enclosure Approach

To overcome the above limitations of existing hierarchical visualization techniques, a new information technique called *Space-Optimized Tree* (SO Tree) [1] was proposed. It takes advantages of both worlds; the *connection* and *enclosure*, and so called a *connection + enclosure* approach. This technique can be used to optimize the drawing of trees in a geometrical plane and maximize the utilization of display space by allowing more nodes and links to be displayed at a limited screen resolution. Space-Optimized Tree recursively positions children of a sub-tree into polygon areas and still uses a node-link diagram to present the entire hierarchical structure (see Figure 2). The use of polygon partitioning effectively addresses the problem of space utilization, while the use of a node-link diagram addresses the problem of human comprehension of the underlying hierarchical structures.

However, the polygons are sometime not a good sharp to percept for some viewers, and also much hard to calculate than other kind of sharps, such as rectangles and circles. The calculation of angles for every polygon is quite computational expensive.

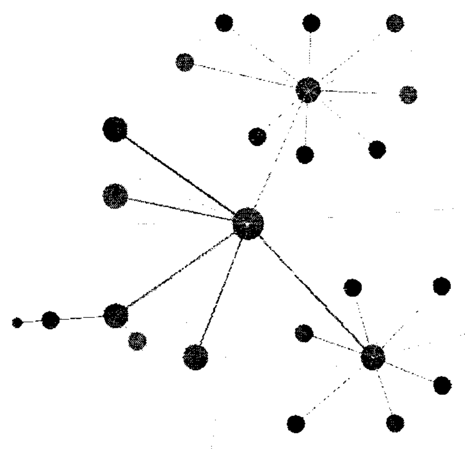


Figure 2. An Example of SO-Tree using polygon partitioning.

1.4 Our ENCCON Model

This paper proposes a new *Enclosure and Connection* (ENCCON) approach for the visualization of large relational tree-hierarchies. This paper describes a new efficient approach for visualizing large hierarchical information. Our technique inherits the advantages of SO Tree technique that can enhance the usability of display space by using area division. This technique is more straightforward than SO Tree, which uses the rectangles for the area division instead of polygons as SO tree. This not only decreases the computation cost in recursively calculating geometrical polygons, but also greatly reduces the perceptual and cognitive loads spent on understanding and perception of the underlying hierarchical structure. We use the semantic zooming to enlarge a particular viewing area and filter out the rest of structure that is less interested. The navigation is accommodated by animation in order to preserve the mental map [2]. The detail of ENCCON will be described at the next section.

2. Technical Specification

Our ENCCON technique can only be applied to rooted trees. We now review the terminology that is used in our technique.

2.1. Terminology

A tree is a connected graph without a cycle. A rooted tree consists of a tree *T* and a distinguished vertex *r* of *T*. The vertex *r* is called the root of *T*. In other words, *T* can be

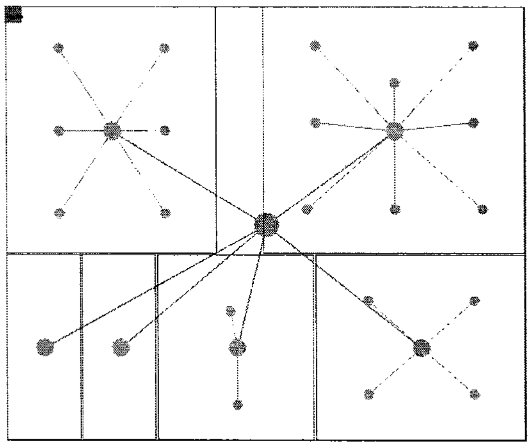


Figure 3a. An example of a rectangular area division, using C = 0.45.

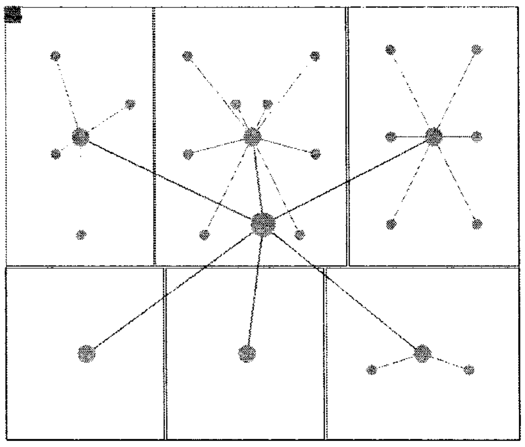


Figure 3b. An example of a rectangular area division, using C = 0.10.

viewed as a directed acyclic graph with all edges oriented away from the root. If *(μ, v)* is a directed edge in *T*, we then say μ is the father of v and v is a child of μ. A leaf is a vertex with no children. If *T* contains the vertex v, then the sub-tree *T(v)* rooted at v is the sub-graph induced by all vertices on paths originating from v. We also use a node to represent a vertex v with its displaying properties. This terminology is mainly mentioned in display section. Each vertex v has an associated value *w(v)*, which we call the weight. The local region *R(v)* of the vertex v is a rectangle, and it contains the drawing of a sub-tree *T(v)*. The rectangle $R(v_i)$ is proportional to the weight $w(v_i)$ of the vertex v_i.

2.2. Geometrical Layout

This part is responsible for defining the position of all vertexes of the given tree hierarchy in 2-dimentional space. Each vertex v_i is bounded by a rectangular local region $R(v_i)$ that the drawing of the sub-tree $T(v_i)$ is restricted within the area of $R(v_i)$. The position of vertex v_i is at the centre of the rectangle defined by $R(v_i)$. See the examples in Figures 3a and 3b.

Weight calculation: we assign a weight *w(v)* to each vertex *v* for the calculation of the local region *P(v)* that relates to the regions of its father and siblings. The calculation is done recursively from leaves to a vertex using the following formula:

$$w(v) = 1 + C\sum_{i=1}^{n} w(v_i) \qquad (1)$$

Where C is a constant (0 < C < 1), and $w(v_i)$ is the weight assigned to the i^{th} child of vertex *v*. Constant C is a scalar that determines the difference between the vertexes' local region with more descendants and vertexes' local region with fewer descendants (see Figures 3a and 3b). All screen snapshots in this paper use the value C = 0.45.

Local region partitioning: we firstly define the local region of the root to be the entirely rectangular display area. Then, the root vertex is placed at the centre of this rectangle. The partitioning starts from the root and ends when all the leaf vertexes are reached. Suppose that the local region *R(v)* of the vertex *v* is defined, the position of *v* is at the centre of *R(v)*. We need to calculate the local regions $\{R(v_1), R(v_2), ..., R(v_n)\}$ for all the children $\{v_1, v_2, ..., v_n\}$ of vertex *v*. The partitioning ensures that the area that the area of rectangle $R(v_i)$ *is* proportional to the weight $W(v_i)$ of the vertex v_i.

The division of *R(v)* into sub-regions $\{R(v_1), R(v_2), ..., R(v_n)\}$ is processed as below:

1. Suppose the vertex μ is the parent of vertex *v* and we firstly find the *initial side* of rectangle *R(v)* on which the vector $\overrightarrow{\mu v}$ cuts the rectangle *R(v)* or the side which is closest to μ. If *v* is the root then the *initial side* is defined as the bottom side. We start the partitioning on the opposite side of the *initial side*, and the partitioning is applied respectively to each of four sides of *R(v)* in a clock-wise direction (see Figure 5).

2. On each side, the partitioning creates and fills in *m* (*m*<*n*) rectangular sub-regions of the same width (or height). Thus, these sub-regions will form a large filled rectangle and leave a *remaining empty region* (see Figure 4) which is also a rectangle. Then, this rectangle is used for the partitioning on the next side. The number *m* is determined by the size of the *remaining rectangle* as well as the smallest width/height ratio λ_{k+i} of all ratios $\{\lambda_{k+1}, \lambda_{k+2}, ..., \lambda_{k+m}\}$, where we have:

$$\lambda_i = \frac{wR(v_i)}{hR(v_i)} \qquad (2)$$

Suppose that the partitioning is on one side of the *remaining rectangle*. We firstly check the numbers of children that can be added into this side. We want to maximize the number *m*, but also ensure that every sub-rectangle on the side is not too thin, which mush satisfy with the following equation:

$$\frac{1}{\rho} < \lambda_i < \rho \qquad (3)$$

Where $wR(v_i)$ and $hR(v_i)$ are respectively the width and height of a sub-rectangle $R(v_i)$. ρ is a constant that equals 1.43 in our implementation.

3. Suppose that there are *m* children $\{v_{k+1}, v_{k+2}, ..., v_{k+m}\}$ of vertex *v* need to be added into a side of the *remaining rectangle*, with respectively local regions: $\{R(v_{k+1}), R(v_{k+2}), ..., R(v_{k+m})\}$. $R(v_{k+i})$ are rectangles and each has a width $wR(v_{k+i})$ and height $hR(v_{k+i})$. Suppose that the widths of sub-rectangles are same as the direction of the division-side, and the length of this side is l_1 and the length of the other side is l_2. Then the width $wR(v_{k+i})$ and height $hR(v_{k+i})$ of the rectangle $R(v_{k+i})$ are calculated by the following formulas:

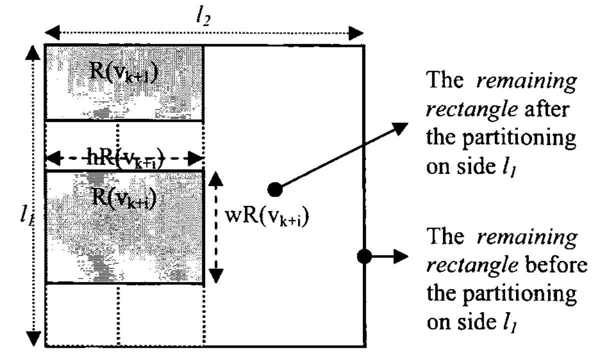
$$wR(v_{k+i}) = l_1 \frac{w(v_{k+i})}{\sum_{j=1}^{m} w(v_{k+j})} \qquad (4)$$

$$hR(v_{k+i}) = l_2 \frac{\sum_{j=1}^{m} w(v_{k+j})}{RW} \qquad (5)$$

Where $w(v)$ is the weight of the vertex *v*. *RW* is the temporary weight of the *remaining rectangle*. *RW* is initially defined as the total weight of all the children $\{v_1, v_2, ..., v_n\}$. The value of *RW* after this division is:

$$RW = RW - \sum_{j=1}^{m} w(v_{k+j}) \qquad (6)$$

The above formulas ensure that the area of each child is proportional to the weight of the child. If the division cannot be completed, we increase the initial weight of *RW* and implement the division again. The algorithm works best with the list of vertexes in ascendant order of weight. Thus, the list of vertexes $\{v_1, v_2, ..., v_n\}$ is sorted in increasing order of weights before we calculate the sub-regions for these vertexes. Figure 4 illustrates the partitioning on the left-hand side of the *remaining rectangle*.



l_2
$R(v_{k+1})$
$hR(v_{k+i})$
$R(v_{k+i})$
l_1
$wR(v_{k+i})$
The *remaining rectangle* after the partitioning on side l_1
The *remaining rectangle* before the partitioning on side l_1

Figure 4. An example of the partitioning on the left-hand side of the rectangle.

Example of the partitioning: suppose we have a rectangle with width 6 and height 4, we need to divide this rectangle into 5 rectangles whose weight are respectively *{4, 4, 2, 2, 1}*, and the starting partitioning side is the left side. The weight of this rectangle is 13.

The first step is to add a single rectangle with the weight 4 into the first division side (hR_1 = 4*6/13, wR_1 = 4). Next, we add the second rectangle (weight 4) above the first, i.e. they share the common left side from the original large rectangle. This two rectangles will have the dimension respectively of (hR_1 = 8*6/13, wR_1 = 4*4/8) and (hR_2 = 8*6/13, wR_2 = 4*4/8). Step 3 inserts the third rectangle (weight 2) on top of two rectangles. However, this step is dismissed because the last produced rectangle is too thin (hR_3 = 10*6/13, wR_3 = 2*4/10, $\lambda_3 = 0.174$). We now start the second partitioning circle, moving from the left side to the top side. In the remaining rectangle, the division continues on the second side (on the top). Two more circles are repeated on other two sides of the *remaining rectangle* until all rectangles have been positioned (see Figure 5).

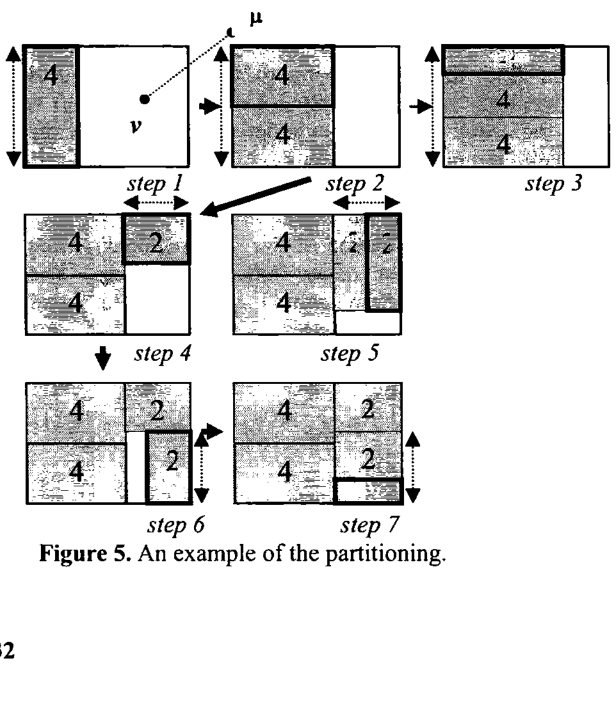


Figure 5. An example of the partitioning.

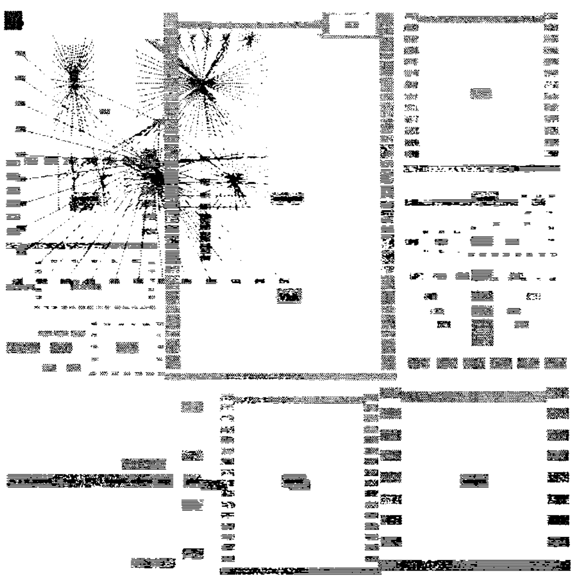


Figure 6b. An example of implementing animated zooming in after a left mouse-click.

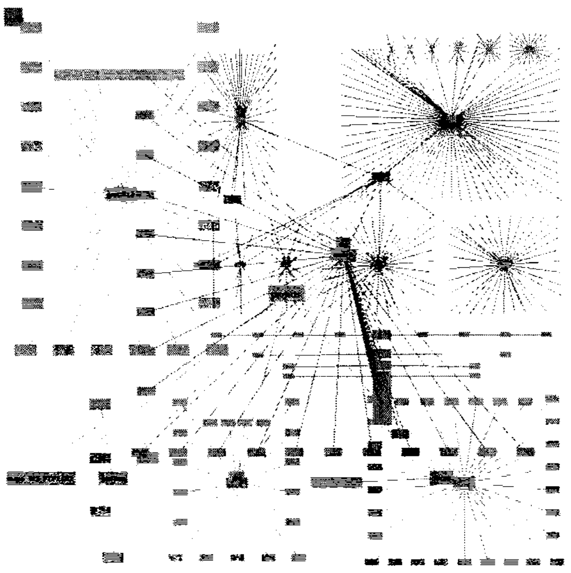


Figure 6d. An example of implementing animated zooming out after a right mouse-click.

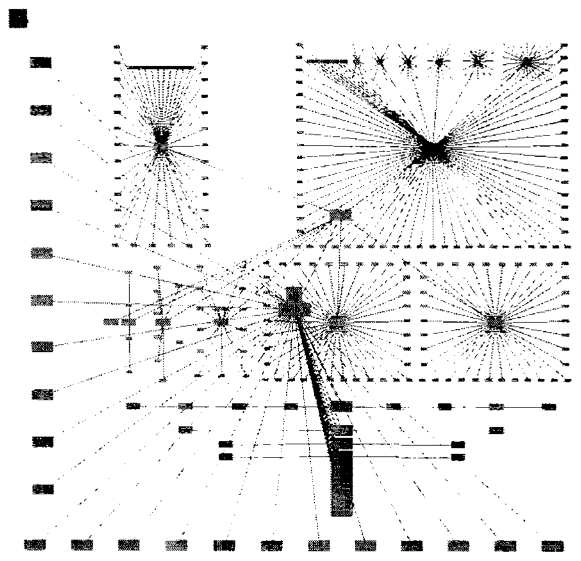


Figure 6c. An example of the layout after animated zooming out by clicking on a node.

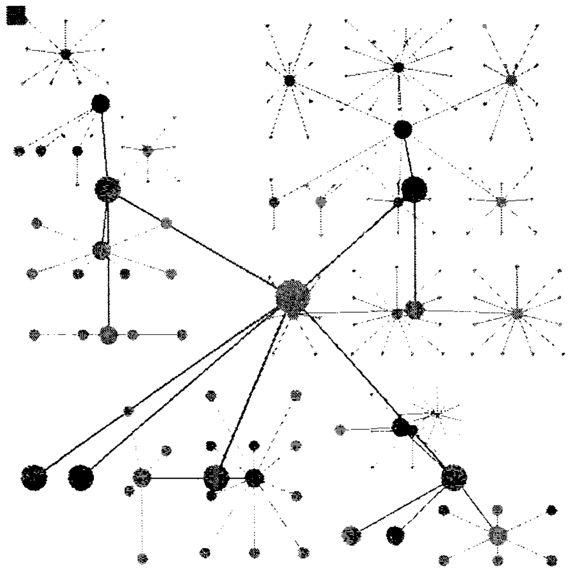


Figure 7a. An example of implementing our layout algorithm with a medium size data (approximately 190 nodes)

2.3. Navigation and Animation

We use semantic zooming for the viewing and navigation of the hierarchical structure in our ENCCON system. When a particular node *v* is selected by a left mouse-clicking, the selected node *v* moves smoothly forward to the center position of the root (i.e. center of the rectangular display area) and the surrounded area of *v* is zoomed in. The display region of node *v* now expands to the entire display area. In other words, we only visualize the sub-tree of selected node and the rest of the hierarchical structure is filtered out. This viewing technique requires the recalculation of positions of all vertexes of the sub-tree that is currently appearing after a corresponding left mouse-click occurs (see Figure 5c). The system implement zoom out (move back to the display of father's hierarchy) when the user clicks on the right button.

The animation is also accommodated with the navigation in order to preserve the mental map [2] during the navigation. The animation is achieved by smoothly moving the nodes to the new location, and fading in and fading out the nodes. Particularly, the animation process can be described below:

- Zoom in (corresponding to the left mouse-click on a node): the focused sub-tree expands smoothly to occupy the entirely display area. The size of nodes in this sub-tree also increases to reach their new size. Simultaneously, the color of nodes, which is visible at the previous state, is fading out to the background color (see figure 6b).

- Zoom out (corresponding to right mouse-click): the nodes of the sub-tree (from the previous state) move smoothly to their new location, and their size decreases to the new size. Simultaneously, the parent's hierarchy is smoothly fading in from the background color to the node color (see figure 6d).

2.4 Display Property

As same as the other area division approaches where a child's area is always smaller than its father, we also apply this rule to our viewing technique. The size of nodes and the width of edges we choose are proportional to their levels in the hierarchy. This means that, the closer to the root, the larger of nodes and the wider of edges. This rule, some how, improves the clarity of the presentation of tree hierarchies.

2.5 Examples

Figures 7a and 7b are examples of applying our technique on a medium large and a very large data set. A demo of ECCON system (implemented in Java Applet) is available at: http://www-staff.it.uts.edu.au/~quvnguye/eccon/

3. Conclusion and Future Works

We have presented our ECCON approach for visualizing and manipulating large hierarchies. This technique is extended from SO Tree [1] and is effective and efficient for visualizing large amounts of tree structured relational data. The layout algorithm can draw the entire tree structure of large data sets using enclosure manner. The system allows the viewer to navigate and view any part of the large tree structure using a semantic zooming technique. Animation is also accommodated in order to preserve the mental map and reduce the human cognitive loads during the navigation. Although this technique needs further optimization and improvement, we believe that it is a valuable tool for visualizing a variety of the large real tree-structural data sets in many application domains.

Next step, we will investigate new layout algorithms to improve the efficiency of using display space. We will also investigate new focus+context [18] viewing techniques that can keep the global view of the entire tree structures in a small region while a detailed view of a part of the tree can be displayed in the large region during the navigation. In addition, we will work on usability test to demonstrate the benefits of using our technique.

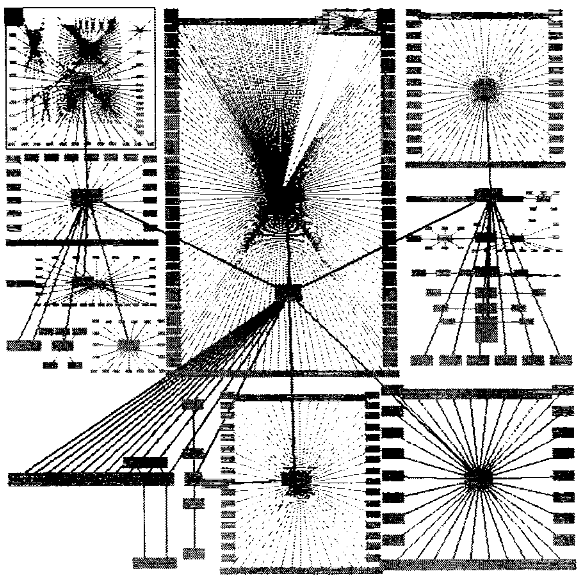


Figure 6a. An example of displaying the entire hierarchy before the navigation.

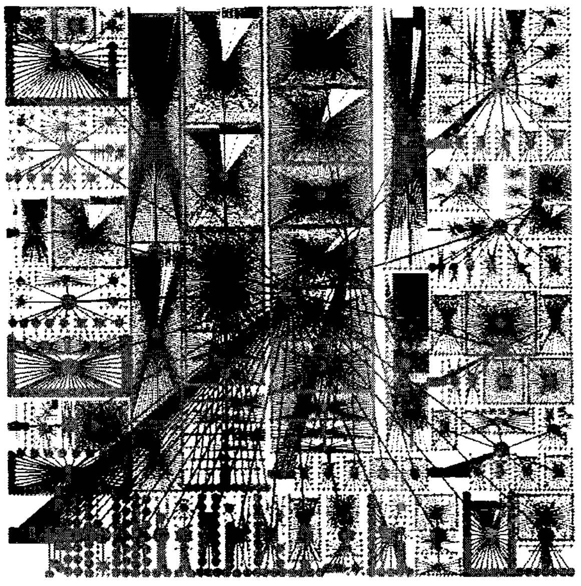


Figure 7b. An example of implementing our layout algorithm on a very large data set of a file system (with approximately 11100 nodes)

References

[1] Q.V. Nguyen, M.L. Huang, Space-optimized tree: a connection + enclosure approach for the visualization of large hierarchies, *Information Visualization Journal*, Palgrave, **2**(1), 2003, 3-15.

[2] L. Bartram, Perceptual and interpretative properties of motion for information visualization, *Technical Report CMPT-TR-1997-15*, School of Computing Science, Simon Fraser University, 1997.

[3] G.G. Robertson, J.D. Mackinlay, S.K. Card, Cone trees: animated 3D visualizations of hierarchical Information, *Proc. CHI'91 on Human Factors in Computing Systems*, ACM Press, New Orleans, Louisiana, USA, 1991, 189-194.

[4] J. Lamping, R. Rao, The hyperbolic browser: a focus+context technique for visualizing large hierarchies", *Journal of Visual Languages and Computing*, **7**, 1995, 33-55.

[5] T. Munzner, Exploring large graphs in 3D hyperbolic space, *IEEE Comp. Graphics & Appl*ications, **18**(4), 1997, 18-23.

[6] P. Eades, Drawing free trees, *Bulleting of the Institute of Combinatorics and its Applications*, 1992, 10-36.

[7] G. Melancon, I. Herman, Circular drawings of rooted trees, *Reports of the Centre for Mathematics and Computer Sciences*, INS-9817, ISSN 1386-3681, 1998.

[8] Ed H. Chi, J. Pitkow, J. Mackinlay, O. Pirolli, R. Gossweiler, S.K. Card, Visualizing the evolution of web ecologies, *Proc. ACM CHI'98 Conference on Human Factors in Computing Systems*, ACM Press, Los Angeles, California, 1998, 400-407, 644-645.

[9] E.M. Reingold, J.S. Tilford, Tidier drawing of trees, *IEEE Transactions on Software Engineering*, **7** (2), 1981, 223-228.

[10] Y. Shiloach, Arrangements of planar graphs on the planar lattices, *Ph.D Thesis*, Weizmann Institute of Science, Rehovot, Israel, 1976.

[11] E. Kleiberg, H.V. Wetering, J.J. Wijk, Botanical visualization of huge hierarchies, *Proc. IEEE Symposium on Information Visualization (InfoVis'01)*, San Diego, CA, 2001, 87-94.

[12] G.J. Wills, NichesWorks – interactive visualization of very large graphs, *Journal of Computational and Graphical Statistics*, **8**, 1999, 190-212.

[13] S.T. Teoh, K.L. Ma, Rings: A technique for visualizing large hierarchies, *Proc. Graph Drawing (GD 2002)*, 2002, California, 268-275.

[14] R.J. Hendley, N. Drew, A. Wood, R. Beale, Narcissus: visualizing information, *Proc. '95 Information Visualization*, Atlanta, 1995, 90-96.

[15] B. Johnson, B. Shneiderman, Tree-maps: a space-filling approach to the visualization of hierarchical information structures, *Proc. The 1991 IEEE Visualization*, Piscataway, NJ, 1991, 284-291.

[16] J.J. van Wijk, H. van de Wetering, Cushion treemaps: visualization of hierarchical information, *Proc. IEEE InfoVis '99*, San Francisco, California, 1999, pp. 73-78.

[17] M. Bruls, K. Huizing, J.J. van Wijk, Squarified Treemaps, *Proc. VisSym '00*, Amsterdam, the Netherlands, 2000, 33-42.

[18] I. Herman, G. Melançon, & M.S. Marshall, Graph visualization in information visualization: a survey, *IEEE Transactions on Visualization and Computer Graphics*, **6**, 2000, 24-44.