Tree visualization with Tree-maps: A 2-d space-filling approach

Ben Shneiderman
Department of Computer Science &
Human-Computer Interaction Laboratory
University of Maryland
College Park, MD 20742

June 18, 1991

(to appear in *ACM Transactions on Graphics*)

Introduction

The traditional approach to representing tree structures is as a rooted, directed graph with the root node at the top of the page and children nodes below the parent node with lines connecting them (Figure 1). Knuth (1968, p. 305-313) has a long discussion about this standard representation, especially why the root is at the top and he offers several alternatives including brief mention of a space-filling approach. However, the remainder of his presentation and most other discussions of trees focus on various node and edge representations. By contrast, this paper deals with a two-dimensional (2-d) space-filling approach in which each node is a rectangle whose area is proportional to some attribute such as node size.

Research on relationships between 2-d images and their representation in tree structures has focussed on node and link representations of 2-d images. This work includes quad-trees (Samet, 1989) and their variants which are important in image processing. The goal of quad trees is to provide a tree representation for storage compression and efficient operations on bit-mapped images. XY-trees (Nagy & Seth, 1984) are a traditional tree representation of two-dimensional layouts found in newspaper, magazine, or book pages. Related concepts include k-d trees (Bentley and Freidman, 1979), which are often explained with the help of a

2-d rectangular drawing, and hB-trees (Lomet and Salzberg, 1990) which are a more advanced multi-attribute indexing method that has a useful 2-d representation. None of these projects sought to provide human visualization aids for viewing large tree structures.

Tree-maps are a representation designed for human visualization of complex traditional tree structures: arbitary trees are shown with a 2-d space-filling representation. The original motivation for this work was to gain a better representation of the utilization of storage space on a hard disk as viewed from the perspective of a multiple level directory of subdirectories and files, as in Unix, Macintosh Finder, or MS-DOS. In this application the files are leaf nodes and the subdirectories are interior nodes. Most operating systems display the contents of one node at a time with names of the files and subdirectories or icons to represent them. The user can traverse the tree with a mouse click on a directory folder icon or by issuing a command (e.g. CD for change directory). A few systems attempt to show more than one node at at time, but very quickly the limited screen space is exceeded. Even clever attempts to show the full tree structure and allow rapid page turning soon exceed practical limits. For example, Norton Utilities from Symantec and Windows 3.0 from Microsoft show trees on their side with the root at the left of the display and indentation plus some lines to show the tree structure. In summary, even elegant tree-like layouts of the hierarchical structures, such as in Sun Microsystem's Open Look, soon overwhelm the available display space and users cannot grasp the entire picture.

This is an old problem. Even designers of family trees, animal species trees, or organization charts found that a large wall was necessary to give the whole picture. But even then only the structural relationship is shown; additional information, such as the size or importance of each node, was ignored or written in text form.

In the computer directory representation application, the goal was to show the entire set of files in a space-filling visualization that would allow users to recognize rapidly the larger files and consider them as candidates for deletion when the hard disk was filled. These large files might be at any level of the tree structure and the range in file sizes might be 5 or 6 orders of magnitude (from a few bytes to a few million bytes). There might be 2 to 8 levels with many thousands of files.

One approach to this problem is to choose a 1-d representation with the length of each file coded as the length of a portion of a multi-colored line. (Knuth, 1968, p. 435). This is impractical because the line would be too long to view. A 3-d or higher dimensional approach might be hard to draw and view. A 2-d space-filling approach has the potential to be drawable and comprehensible. If each file were represented as a small rectangle, the method would work, but it would be necessary to avoid a computationally intensive bin-packing algorithm. The following tree visualization approach, called *tree-maps* (Figure 1 and 2), appears to solve the practical problem and provide interesting opportunities for other applications.

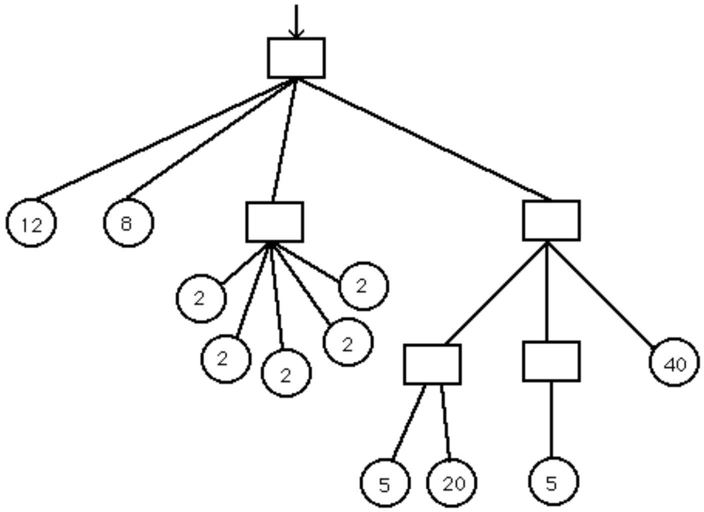


Figure 1: Typical 3-level tree structure with numbers indicating size of each leaf node

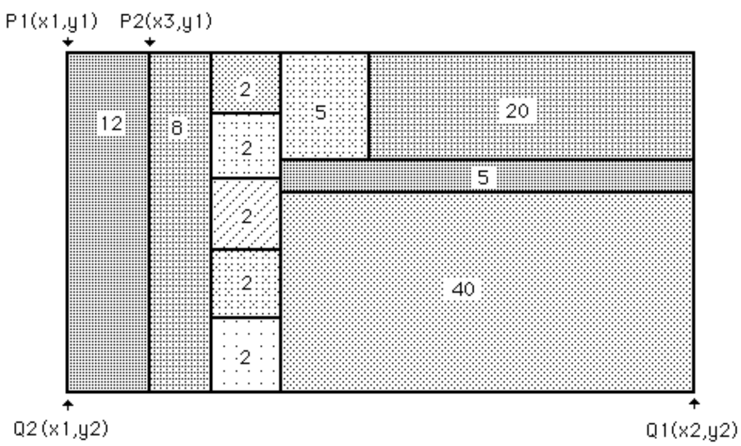


Figure 2: Tree-map of Figure 1

Tree-map algorithm

The algorithm takes a tree root (Figure 1) and a rectangular area defined by the upper left and lower right coordinates P1(x1, y1), Q1(x2, y2). The number of outgoing edges from the root node determines the number of partitions of the region [x1, x2]. Since the leftmost subtree contains a fraction (Size(child[1])/Size(root)) of the total number of bytes in the root, then the first vertical partitioning line is drawn at:

x3 = x1 + (Size(child[1])/Size(root)) *(x2 - x1)

The algorithm then recurs down the left tree using the 90-degree rotated rectangle P2(x3, y1), Q2(x1, y2) and splits on the y-axis direction, while the loop continues along the remaining subtrees making partitions on the remaining rectangle P2(x3, y1), Q1(x2, y2). Therefore nodes are partitioned vertically at even levels and horizontally at odd levels (Figure 2).

For visual clarity, different colors (or gray shading) must be used within each region. The effect of seeing thousands of small rectangles is like a checkerboard with varying sized spots. Color coding could represent different types of files (e.g. text, programs, binary, graphics, spreadsheets), owners of programs (each owner has a different color), frequency of use (brighter colors for more frequent use), or the age of a file (older files might be more yellow or grayer). If adjacent areas have the same color, then a boundary line will be necessary. Since users' needs will vary extensively, no specific solution can satisfy all situations, and the users should have a control panel for several parameters and also to indicate which colors are assigned to attribute values.

Since the files that take storage space on the hard disk are all leaf nodes (we ignore, for the moment, the usually modest storage overhead consumed by a directory), each interior node must have the total size of its subtree (if necessary, then propagate the sums of storage consumed for each file and subdirectory up through the levels of the tree to the root). If this data is not maintained by the system, then there must be a preliminary pass through the tree to collect this data and place it at each interior node. If the user desires to see the display as a percentage of total disk space utilization, then the root node must have one additional child which is a dummy record whose size is the entire unused portion of the disk.

The tree-map algorithm assumes a tree structure in which each node contains a record with its directory or file name (*name*), the number of children (*num_children*), and an array of pointers to the next level (*child* [1..num_children]). The arguments to the tree-map algorithm are:

root : a pointer to the root of the tree or subtree

P, Q : arrays of length 2 with (x,y) coordinate pairs of opposite corners of the current rectangle (assume that Q contains the higher coordinates and P the lower coordinates, but this does not affect the correctness of the algorithm, only the order in which rectangles are drawn)

axis : varies between 0 and 1 to indicate cuts to be made vertically and horizontally

color: indicates the color to be used for the current rectangle.

In addition we need:

Paint_rectangle : a procedure that paints within the rectangle using a given color, and resets the color variable.

Size : a function that returns the number of bytes in the node pointed to by the argument. Alternatively, the size could be pre-computed and stored in each node.

The initial call is:

```
Treemap(root, P, Q, 0, color)
```

where P and Q are the upper right and lower left corners of the display. By setting the *axis* argument to zero the initial partititions are made vertically. It is assumed that arguments P and Q are passed by value (since P, Q are modified within):

```
Treemap(root, P[0..1], Q[0..1], axis, color)
  Paint_rectangle(P, Q, color)                -- paint full area
  width := Q[axis] - P[axis]                  -- compute location of next slice
  for i := 1 to num_children do
      Q[axis] := P[axis] + (Size(child[i])/Size(root))*width
      Treemap(child[i], P, Q, 1 - axis, color)     -- recur on each slice, flipping axes
      P[axis] := Q[axis];
  endfor
```

Execution speed and pattern: This algorithm runs linearly with the number of nodes in the tree structure. This version will paint the rectangles from left to right and top to bottom, with deeper levels covering colored sections as previously drawn during the depth first traversal. Breadth first traversals are also possible, as are algorithms that do not cover already colored sections. We have also tried sequencing leaf nodes before subtrees with ordering by date or size (ascending or descending each have their advantages).
Obtaining file names: When used for directory displaying, the users need to examine the file name, extension, date, etc. This can be accomplished in many ways. For example, the users could move a cursor onto a candidate region, and then click to obtain the relevant information at the bottom line of the screen or just above the cursor itself. Allowing other operations (deletion, copying, marking) by way of pop-up menus is a natural next step. If directory names are desired then nested rectangles that show a containing frame could be used, although this would reduce the effective display space.

Display resolution: On a standard VGA display the resolution is 640 x 480 pixels giving 307,200 total pixels, which is quite adequate for displaying one to two thousand files (each file gets an average of 200 pixels). Of course, small files or zero byte files become too small to represent and are currently eliminated. With larger displays, still larger trees could be displayed. With smaller displays or larger trees, it might be necessary to select among the subdirectories to get adequate detail or to add zooming to reveal small files.

Applications: The applications seem quite broad, but here are a few. In an organization chart, the number of employees or budget in each division or department might be represented to gain an idea of the relative size of each and color might indicate closeness to planned levels. For a library that uses the Dewey decimal system, the number of books within each topic could be represented to see the relative strengths of the library's holdings. In a stock portfolio, the dollar values of each purchase might be represented by the size and the profit/loss might be color coded. In a traditional tree structured data structure, the probability or cost of access might be coded as the size to help find poorly balanced structures. If the tree structure represents a computer program, then the size could represent that amount of time spent in that segment of code, thus guiding an attempt to optimize performance.

Summary

This paper presents a novel approach to representing trees that have weights or sizes on the leaf nodes. The 2-d visualization is space filling and the recursive algorithm for generation runs rapidly. It depends on color coding (or shading) of regions and easily provides users with a quick overview that clearly indicates relative sizes of the the leaf nodes. Figures 3 & 4 show examples of tree-maps with size coding, as implemented by Brian Johnson on a Apple Macintosh II computer with a high resolution color display. Figure 3 shows fifteen files in four directories at three levels, with nested boxes to show the levels. Figure 4 represents actual disk directories encompassing 850 files at four levels with color coding by file type (text, graphics, applications, etc). We continue to explore refinements of tree-maps such as alternate layouts, better methods for coping with large ranges of file size, color coding schemes, and operations applied to files.

Acknowledgements: I gratefully acknowledge the thoughtful comments of the reviewers, editor Dan Olsen's constructive suggestions, and David Mount's help in revising the tree-map algorithm. We appreciate the continuing financial support for the Human-Computer Interaction Laboratory's research from Apple, NCR Corporation, and Sun Microsystems.

References

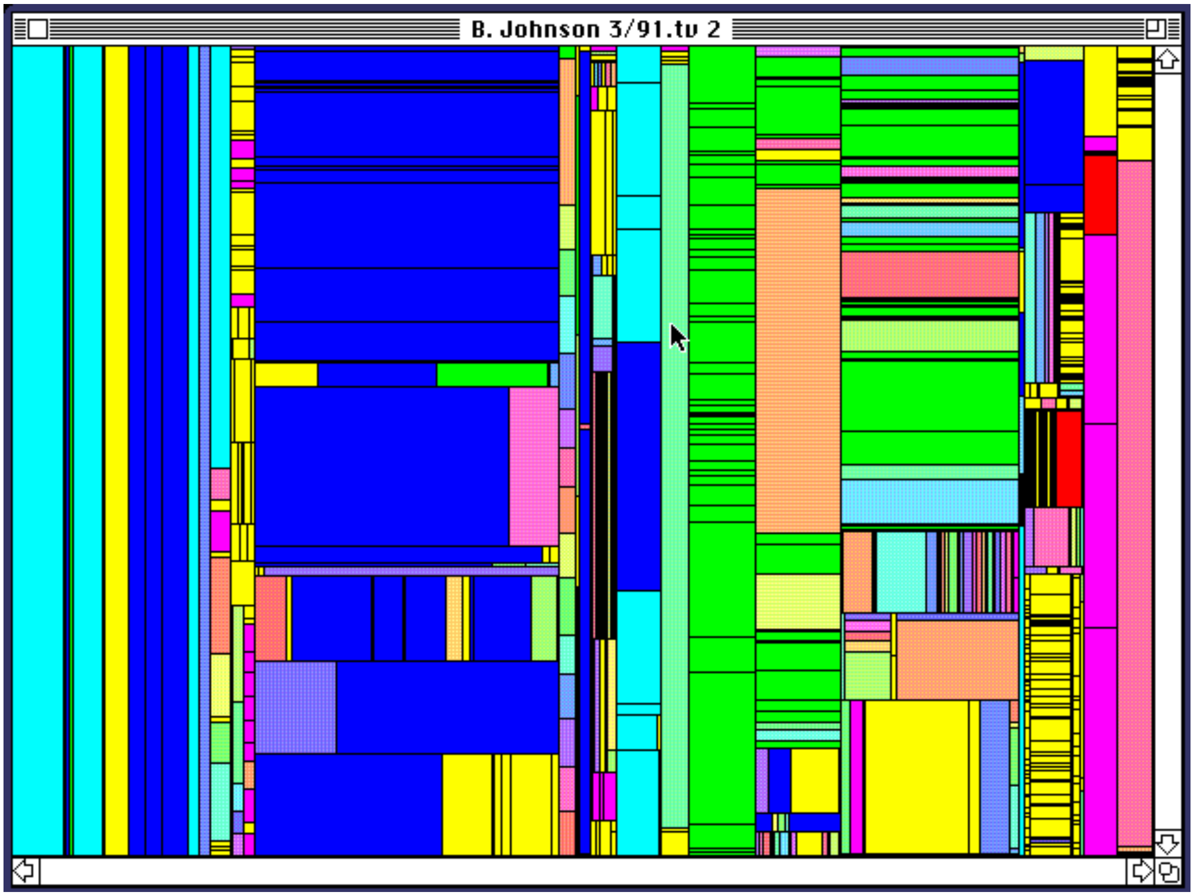
Bentley, J. L. and Friedman, J. H., Data structures for range searching, *ACM Computing Surveys, 11*, 4, (1979), 397-409.

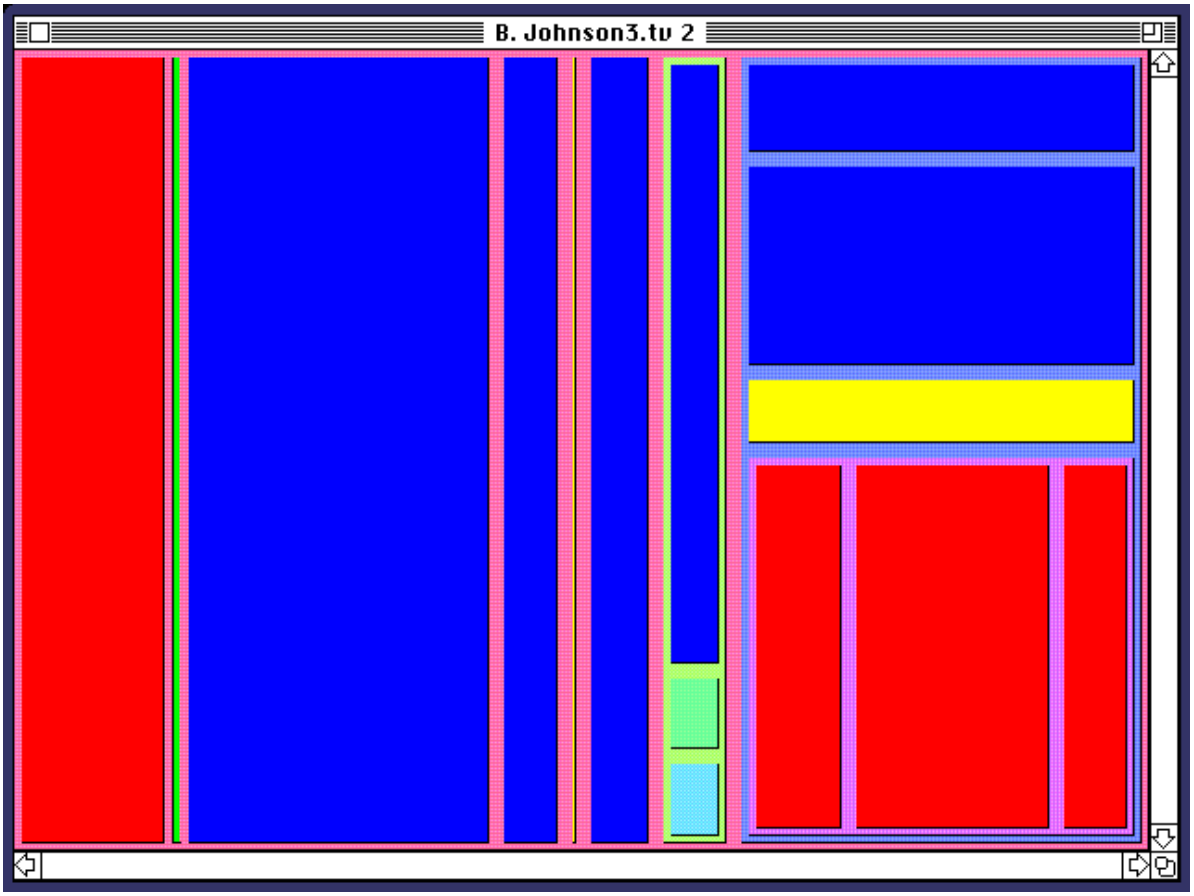
Knuth, Donald E., *The Art of Computer Programming: Volume 1 / Fundamental Algorithms*, Addison-Wesley Publishing Co., Reading, MA, (1968).

Lomet, David B. and Salzberg, Betty, The hB-tree: A multiattribute indexing method with good guaranteed performance, *ACM Transactions on Database Systems 15*, 4, (December 1990), 625-658.

Nagy, G. and Seth, S., Hierarchical representation of optically scanned documents, *Proc. of the IEEE 7th International Confeerence on Pattern Recognition*, Montreal Canada, (1984), 347-349.

Samet, Hanan, *Design and Analysis of Spatial Data Structures*, Addison-Wesley Publishing Co., Reading, MA, (1989).


B. Johnson 3/91.tv 2


B. Johnson3.tv 2