

An Interactive View for Hierarchical Clustering

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

This paper describes a visualization of a general hierarchical clustering algorithm that allows the user to manipulate the number of classes produced by the clustering method without requiring a radical re-drawing of the clustering tree. The visual method used, a space-filling recursive division of a rectangular area, keeps the items under consideration at the same screen position even while the number of classes is under interactive control. As well as presenting a compact representation of the clustering with different cluster numbers, this method is particularly useful in a linked views environment where additional information can be added to a display to encode other information, without this added level of detail being perturbed when changes are made to the number of clusters.

1. Introduction

A common problem in data analysis is forming groups of similar items based on a number of variables describing the items. A classic case is Fisher's analysis of iris data [3] in which measurements of sepal and petal length and width can be used to place the plants into groups that correspond to different species of iris plant. In business settings, it is often important to form customer groups for precision marketing.

Hierarchical clustering is one of the more common methods employed for categorizing cases based on available information on those cases. The overall goal of any clustering is clear: Using variables that describe the cases, divide the cases into a number of classes such that each class contains members that are similar to each other and dissimilar to members of other classes. This simple goal does not, however, translate to a simple algorithm, and there is a wide body of literature relating different techniques for creating such clusters [2]. Probably the most common technique is that of *hierarchical clustering* (ibid.)

This method of clustering does not require, as some methods do, that the number of resulting clusters be pre-set. Instead, a hierarchical clustering view builds a binary tree in which the original data items are the leaves, and interior nodes represent clusters of items. The interior

nodes are also marked with the measure of the dissimilarity between the two sets of child clusters. By cutting the tree at a given level of dissimilarity, the analyst can create clusterings with different numbers of groups without re-running the algorithm. This property is very important in the exploratory study of large data sets; it allows the analyst to run a potentially very slow algorithm on a large set of data and then examine the resulting artifact in an exploratory fashion. For the purposes of this paper, the method used to perform the hierarchical clustering is irrelevant; we are concerned only with visualizing the resulting tree.

Table 1 shows some sample data which we will use to demonstrate our approach; it consists of data on seven animals; their relative sizes and number of legs. In Figure 1 we show a tree resulting from the application of a common hierarchical clustering method, with interior

Animal	Legs	Size
cat	4	2
cow	4	13
dog	4	5
horse	4	12
kangaroo	2	7
man	2	9
snake	0	3

Table 1. Sample data items

(cluster) nodes annotated with the dissimilarity between their children. We can see that node *D* has dissimilarity 4.5, while node *B* has dissimilarity 2.0. This means that *man* is more similar to *kangaroo* than *snake* is to the *cat-dog* pair.

Table 2 shows the results of cutting this tree at different levels of dissimilarity. As we decrease the allowable dissimilarity, we cut the tree lower down and

so produce more clusters. With dissimilarity 10 there is only one cluster; with a dissimilarity of 2.5 there are five.

Figure 1 is the usual way of visualizing the results of a hierarchical clustering algorithm, but it becomes too cumbersome with even moderate sized data sets of over a thousand nodes. A common solution to this problem is not to display the full tree, but only to display the tree as

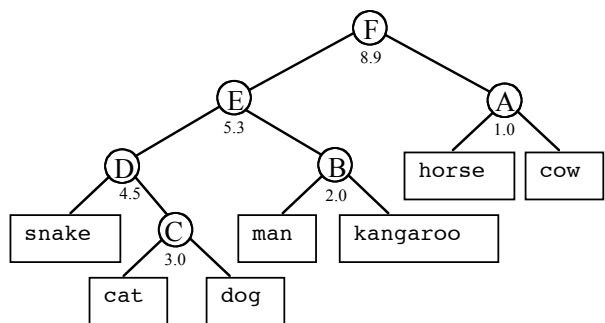


Figure 1. Tree created on sample data

far as the current cut level plus one step (for example, with dissimilarity 5, only the nodes A, B, D, E, F would be shown). This approach is suitable for many applications, but suffers from several problems. First, as the cut level is changed interactively, the tree displayed often changes dramatically, growing in some directions and remaining static in others. As well as being distracting, this has the associated problem that the cluster in which a specific data item lies changes as the tree is grown or shrunk and therefore it is hard to locate individual items. A second problem is that even moderately sized trees of 100 leaves are still hard to see, so that clustering is limited to fewer numbers of clusters. Third, cluster size is not obvious. Fourth, it offers no hint of what might happen if the number of clusters is increased. We address these problems by proposing a new method of viewing the results of a hierarchical clustering

Dissimilarity	Nodes Cut	Clusters
10	- none -	cat-cow-dog-horse-kangaroo-man-snake
7.5	F	cat-dog-kangaroo-man-snake cow-horse
5	E, F	cat-dog-snake kangaroo-man cow-horse
2.5	C, D, E, F	cat dog snake kangaroo-man cow-horse

Table 2. Clusters at various levels of allowable dissimilarity

algorithm motivated by space-filling tree layout methods such as *tree-maps* [5].

2. Description of the Method

A tree-map, as described in [5] is a method for drawing a tree that makes maximal use of screen space. The basic version takes a specified rectangular area and recursively subdivides it up based on the tree structure. It looks at the first level of the tree and splits up the viewing area horizontally into n rectangles, where n is the number of children of the first node. Each rectangle is allocated an area proportional to the size of the subtree underneath each child node. The algorithm then looks at the next level of the tree and for each node here performs the same algorithm, except it recursively divides the area vertically. The algorithm continues doing this subdivision in alternating directions until either the maximum specified depth is reached or a leaf node is reached. In either case the rectangular area for that node is then drawn with user-specified characteristics such as color, shading and labeling.

The tree-map recursive subdivision model was looked at for implementing a view on hierarchical clustering, but was found to be lacking in the following areas:

- *The tree-map has no concept of tree dissimilarity; it assumes that the user wants to cut the tree at given depths.* This would mean that the clustering associated with dissimilarity 5 in our example could not be represented by the tree-map. The best it could do would be to cut at depth 2 and give the groups *cat-dog-snake*, *kangaroo-man*, *cow*, *horse*. This splits up *cow* and *horse*, which is wrong as they are very similar to each other. We therefore modify the splitting criteria so that it recursively splits until it reaches a leaf or the node has sufficiently small dissimilarity.
- *The tree-map shows accumulated groups of data items only.* We want to see not the accumulated groups, but also the individual data items (so that we can color individual data items differently, rather than just color groups all one color). Therefore we modify the algorithm so that when it decides not to split a node any longer it draws a rectangle for the bounding cluster, and then enters a second mode where it continues to recursively split until it reaches a leaf node, where it draws the node as a glyph in the center of the rectangle that the splitting algorithm has created.
- *The tree-map's alternating horizontal and vertical splits is good at showing the tree depth, but has a tendency to create very long skinny rectangles with unbalanced trees.* Since such trees are common in hierarchical clustering and since we have no interest in the depth of a given cluster, just its dissimilarity, we elect to choose to split along the longest axis of

the current rectangle, so that the resulting rectangles are more square than they would have been if split along the other axis. We do this without any consideration of the depth of the current node.

- *The tree-map typically shows the complete hierarchy of split nodes, inseting rectangles prior to splitting to do so.* Since our interest is not in seeing at what depth the clusters occur, but in seeing the clusters themselves as clearly as possible, we omit this inseting and display.

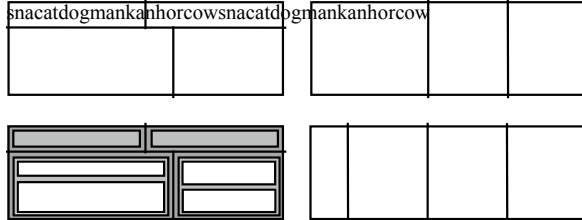


Figure 3. Tree-map(left) compared with the Hierarchical Clustering View (right). Tree-Map is cut at depths 2 and 3. The clustering view is cut at dissimilarities 5.0 and 2.5

Figure 3 shows how the hierarchical clustering view of Figure 1 would differ from a tree-map version. Note that in the tree-map version it is impossible to split up the group *cat-dog-snake* without also splitting up *cow-horse*, so a perfect comparison cannot be made. The first tree-map is shown without inseting; the second is shown with inseting (the default setting). For both views continuing to split deeper does not alter the locations of existing groups, nor does it change the size of the view, but where the tree-map shows the details of the hierarchy more accurately, it does so by presenting a more cluttered view that focuses attention on the tree depths, not the clusters. The tree view shows us that the leaf nodes *horse* and *cow* are at a different depth than the others, but that is an unimportant detail. The alternating horizontal-vertical split pattern has also created several long thin bars. Since we know these are leaf nodes, we know that these subtend the same areas, but if we did not

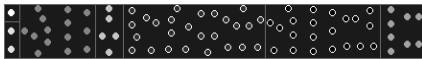


Figure 4. Clustering view with elongated area

know that it would be hard to tell that the cluster 'sna' is the same size as the cluster 'kan'.

3. Detailed Example

In this section we show a simple example of the Hierarchical Clustering View being used to examine some data on breakfast cereals. This data set is freely available from the net at statlib (<http://lib.stat.cmu.edu>) and

consists of nutritional information on 76 different cereals. In this section we use two of them, *sugar* and *fiber* to cluster the data and we use a third, *shelf*, to color individual data points. Figure 2 shows the clustering view for various different dissimilarity values. The line graph at the bottom of each view is a plot of number of clusters (x-axis) against dissimilarity needed to do next split (y-axis). The dissimilarity values were chosen subjectively by dragging the mouse over the line graph and watching the plot change.

Looking down the series of screen shots, the property that the configuration of points does not change is obvious. All that is changing is the enclosing boxes that define the groups.

The steep descent of the line graph shows us that we do not need many clusters in this data set; probably the seven of the first set are sufficient.

The colors of the data items encode the shelf that the

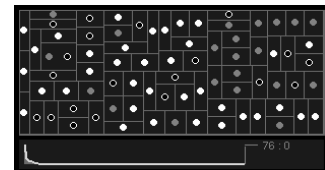
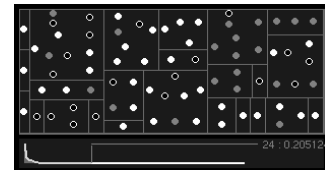
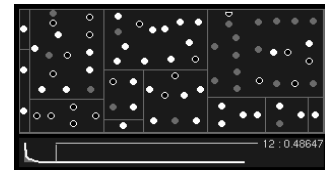
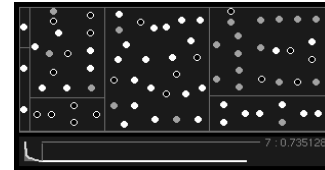


Figure 2. Clustering Cereals

product was found on (black lowest, gray middle and white highest). Notice that one cluster is found only on the bottom shelf. These consist of low sugar, medium fiber cereals; *oatmeal*, *cheerios* and several varieties of *shredded wheat*. Along the left of the view are two groups of one and two items. These are all extremely high fiber *bran* cereals, placed at adult eye level.

At the top right is a group where most of the cereals are in the middle shelf; at children's eye level. Looking at

these we see, without much surprise, that these are the high sugar, low fiber group. The supermarket's plan is clear: Get the children to grab the sweet cereals and guilt the adults into buying good-for-you high-fiber cereal.

Figure 4 and Figure 5 show the effect of changing the aspect ratio of the view both by making it more square and more elongated. In either case the algorithm produces good results. Note that in this figure we have set the grayscale intensity of the items to indicate their cluster group. Figure 6 shows a scatterplot of the variables used to construct the clustering, with data items colored the same as in the cluster view. The white outlier (*100% bran*) and the two other *bran* outliers in off-white are clear. There is not much separation in this data set; the

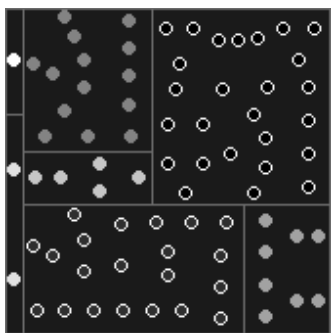


Figure 5. Clustering view with square area

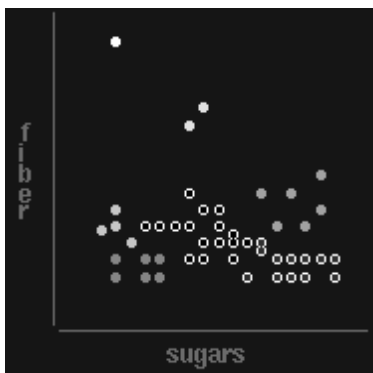


Figure 6. Scatterplot of cluster variables

clusters are not very distant from each other. We might have guessed this from the line graph associated with the cluster view; it drops very steeply then flattens out when the fourth split is made; there are really only three very different groups; the two outlier groups and the body of the data.

4. Large Example

We illustrate the use of this method for large data sets on a set of telecommunications data summarizing call traffic by geographical locations throughout the USA. Six variables have been collected each of which counts a type

of call. By clustering the data and interactively choosing a splitting level that creates a reasonably large split we arrive at figure 7. In the author's experience of large data set clustering, most initial splits create small groups of outliers, requiring the user to cut quite far into the tree to see important structural clusters.

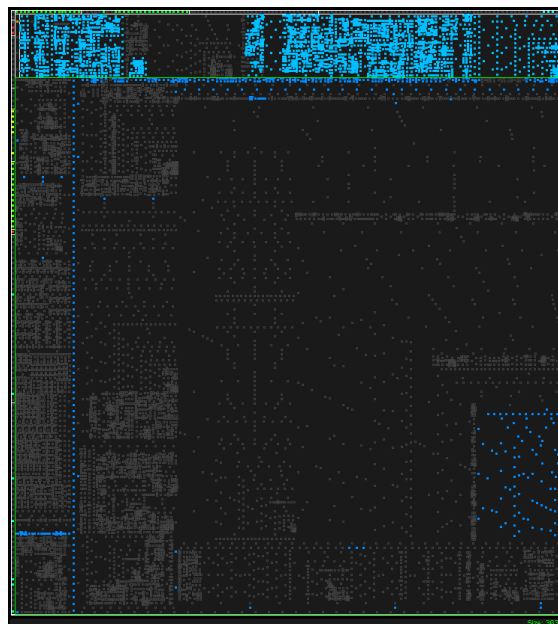


Figure 7. Cluster view for 40,875 zip codes

The second-largest cluster has been selected and other linked multivariate views (parallel axes plots, boxplots and scatterplot matrices) showed that this cluster was high in three variables, but low in the other two. To investigate which of these three variables was affecting the clustering algorithm the most, the user repeatedly selected high values of each of the three variables (the upper quartiles) and examined the results in the linked cluster view. Figure 7 shows one of these selections. It is clear that the second cluster is comprised to a large extent of areas with high levels of this variable, and that deeper splits will split this group into sub-clusters almost entirely composed of areas with high values.

Figure 8 shows part of a plot of the zip code centroids, with the color of the points indicating the cluster group (unselected areas, shown in gray in figure 7, have been hidden in this view). Note the geographic clustering of areas that have been clustered by their data into the green group. The linking both to and from the cluster view allows the user to explore the clusters with respect to other variables and see if there are additional interesting dependencies.

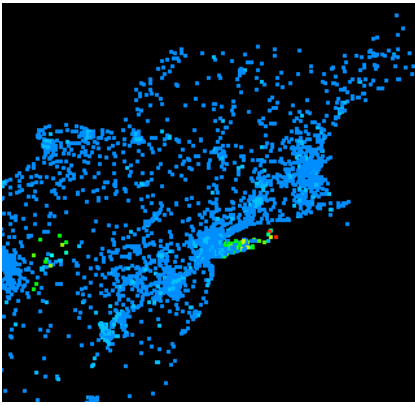


Figure 8. Linking high value points to the map, color-coded by group

5. Summary

The hierarchical clustering view proposed here solves several important criteria:

- It displays individual data items as well as data clusters, making it very useful in an exploratory environment, especially when used in a linked views environment (Eick and Wills, 1995). Furthermore, the locations of the data items do not change when the degree of clustering is changed. Regardless of the groups, the data items remain stationary, aiding navigation and data interrogation.
- It allows the user to try new critical values rapidly and easily and thereby cut the tree into different clusters.
- It is compact; trees such as in Figure 5 with 76 leaf nodes and 75 interior nodes would be very expansive in traditional tree drawings, whereas our method allows trees with tens of thousands of leaves to be shown on a standard display.
- It is adaptable to a wide variety of aspect ratios; even extremely elongated areas pose no problems for it.

This cluster view has been implemented in a linked views environment (Wills, 1996) as one of the linked views components. The clustering algorithm used is a combination of a fast mutual nearest neighbors method (Frakes and Baeza-Yates, 1995) and a complete linkage method (Everitt, 1993), used when the number of clusters is/becomes small), and it is designed to work on a data tables with a variety of different types of variables (categorical and numerical). Within this framework it is feasible to use this view to examine trees on data sets of size up to hundreds of thousands of items.

6. Appendix: Pseudo-code algorithm for drawing the view

The following section of pseudo code shows one way in which the hierarchical clustering visualization can be constructed. It would be used by calling the `split` routine with the tree root node, an initial area into which the view should be drawn, with `cluster_drawn` set

to be true, and with a suitable `critical_value` as set by the user or some other criteria.

```

STRUCTURE node
{
    real dissimilarity // between the child nodes
    real num_leaves // data items in subtree
    node child_a, child_b // next tree level
}

STRUCTURE rectangle
{
    real top, left, right, bottom // bounds
}

SUBROUTINE Draw_Group_Rectangle(rectangle r)
{
    // code to draw the rectangle defining the group
    // A typical example would fill the rectangle
    // with a color and shading
    // encoding information about the group
    // In this paper, we simply draw the
    // frame of the rectangle

    Frame_Rectangle(r)
}

SUBROUTINE Draw_Node_Glyph(node n, rectangle r)
{
    // code to draw the individual data item
    // A typical example would draw a glyph
    // centered in the middle of r
    // with color and shape encoding
    // other variables associated with n
    // In this paper, we choose a color
    // and draw a filled circle with radius 3

    Set_Color( .... )
    Fill_Circle( (r.left + r.right)/2, (r.top + r.bottom)/2, 3);
}

SUBROUTINE split (node n, rectangle r, boolean
cluster_drawn, real critical_value)
{
    IF cluster_drawn = FALSE AND n.dissimilarity <
critical_value THEN
        Draw_Group_Rectangle(r)
        // Draw the group information
        cluster_drawn = TRUE
        // It won't be drawn by deeper calls
    END IF

    IF node.num_leaves > 1 THEN
        rectangle r1 = r // For the left child
        rectangle r2 = r // For the right child
        split_ratio = n.child_a.num_leaves / n.num_leaves;

```

```

IF r.right - r.left > r.bottom - r.top THEN
  // If r is wider than tall
  r1.right = r1.left + (r.right - r.left) * split_ratio
  r2.left = r1.right
ELSE
  // If it's taller than wide
  r1.bottom = r1.top + (r.bottom - r.top) * split_ratio
  r2.top = r1.bottom
END IF

// Recursive calls
split(n.child_a, r1, cluster_drawn, critical_value)
split(n.child_b, r2, cluster_drawn, critical_value)
END IF

IF node.num_leaves = 1 THEN
  Draw_Node_Glyph(n, r) // Draw node itself
END IF
}

```

7. References

- [1] Eick S.J. and Wills G.J. (1995) High Interaction Graphics. *European Journal of Operational Research* **1995**, 445-459.
- [2] Everitt, B. S. (1993) *Cluster Analysis*. 3rd ed. Halsted Press NY
- [3] Fisher, R.A. (1936) The use of multiple measures in taxonomic problems. *Annals of Eugenics* **7**, 179-188
- [4] Frakes W.B. and Baeza-Yates, R. (1995) *Information retrieval : data structures and algorithms*
- [5] Schniederman (1992) Tree visualization with tree-maps; A 2D space-filling approach. *ACM Transactions on Graphics*. January 1992
- [6] Wills G.J. (1996) Selection: 524, 288 Ways to say 'This is Interesting'. *Information Visualization '96 Proceedings*

Captions for color pictures are identical to the ones I
the paper, which will be rendered in black and white