Ordered Treemap Layouts

Ben Shneiderman
Department of Computer Science,
Human-Computer Interaction Lab,
Insitute for Advanced Computer Studies &
Institute for Systems Research
University of Maryland
ben@cs.umd.edu

Martin Wattenberg
Dow Jones / SmartMoney.com
and
Digital Media Center,
Columbia University
mmw111@columbia.edu

Abstract

Treemaps, a space-filling method of visualizing large hierarchical data sets, are receiving increasing attention. Several algorithms have been proposed to create more useful displays by controlling the aspect ratios of the rectangles that make up a treemap. While these algorithms do improve visibility of small items in a single layout, they introduce instability over time in the display of dynamically changing data, and fail to preserve an ordering of the underlying data. This paper introduces the ordered treemap, which addresses these two shortcomings. The ordered treemap algorithm ensures that items near each other in the given order will be near each other in the treemap layout. Using experimental evidence from Monte Carlo trials, we show that compared to other layout algorithms ordered treemaps are more stable while maintaining relatively favorable aspect ratios of the constituent rectangles. A second test set uses stock market data.

Keywords: treemaps, ordered treemaps, trees, hierarchies, information visualization

1. Introduction

Treemaps are a space-filling visualization method capable of representing large hierarchical collections of quantitative data [S92]. A treemap (Figure 1) works by dividing the display area into a nested sequence of rectangles whose areas correspond to an attribute of the data set, effectively combining aspects of a Venn diagram and a pie chart. Originally designed to visualize files on a hard drive, treemaps have been applied to a wide variety of domains ranging from financial analysis [JT92, W98] to sports reporting [JB97].

A key ingredient of a treemap is the algorithm used to create the nested rectangles that make up the map. (We refer to this set of rectangles as the layout of the treemap.) The slice and dice algorithm of the original treemap paper [S92] uses parallel lines to divide a rectangle representing an item into smaller rectangles representing its children. At each level of hierarchy the orientation of the lines - vertical or horizontal - is switched. Though simple to implement, the slice-and-dice layout often creates layouts that contain many rectangles with a high aspect ratio. (In this paper we define the aspect ratio of a rectangle to mean the maximum of width/height and height/width. Using this definition, the lower the aspect ratio of a rectangle, the more nearly square it is; a square has an aspect ratio of 1, which is the lowest possible value.) Such long skinny rectangles can be hard to see, select, compare in size, and label. [TJ92, BHW00]

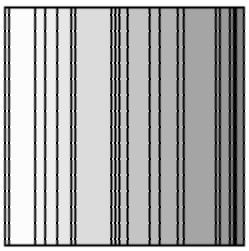

Figure 1. A slice-and-dice layout. Shading indicates order, which is preserved.

Several alternative layout algorithms have recently been proposed to address these concerns. The SmartMoney Map of the Market [W98] is an example of the cluster treemap method described in [W99] which uses a simple recursive algorithm that reduces overall aspect ratios. Bruls, Huizing, and van Wijk [BHW00] introduced the squarified treemap, which uses a different algorithm to achieve the same goal. Figure 2 shows examples of these two layouts.

Presented at IEEE Symposium on Information Visualization
October 22 - October 23, 2001
San Diego Paradise Point Hotel, San Diego

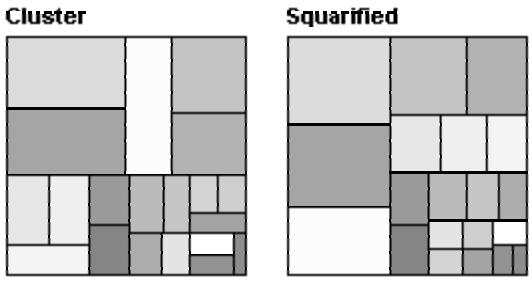

Figure 2. Low aspect ratio layouts. Shading indicates order, which is not preserved.

The new methods suffer from two drawbacks. First, changes in the data set can cause dramatic discontinuous changes in the layouts produced by both cluster treemaps and squarified treemaps. (By contrast, the output of the slice and dice algorithm varies continuously with the input data.) These abrupt layout changes are readily apparent to the eye; below we also describe quantitative measurements of the phenomenon. Large layout changes are undesirable for several reasons. If the treemap data is updated on a second-by-second basis-e.g., in a stock portfolio monitor-then frequent layout changes make it hard to track or select an individual item. Rapid layout changes also cause an unattractive flickering that draws attention away from other aspects of the visualization. Moreover, even occasional abrupt changes mean that it is hard to find items on the treemap by memory, decreasing efficacy for long-term users.

The second shortcoming of cluster and squarified treemap layouts is that many data sets contain ordering information that is helpful for seeing patterns or for locating particular objects in the map. For instance, the bond data described in [J94] is naturally ordered by date of maturity and interest rate. In many other cases the given order is alphabetical. The original slice-and-dice layout preserves the given ordering of the data, but cluster treemaps and squarified treemaps do not. Another recent algorithm [VN00] enables control over the aspect ratios but does not guarantee order.

This paper introduces ordered treemaps, which use layout algorithms that change relatively smoothly under dynamic updates and roughly preserve order, but also produce rectangles with low aspect ratios. We discuss two different algorithms to create ordered treemaps, each with slightly different properties. (Dynamic demonstrations of these algorithms have been posted on the Web, at http://www.columbia.edu/~mmw111/treemap.)

We then report the results of Monte Carlo experiments comparing the two ordered treemap algorithms to squarified treemaps, cluster treemaps, and the slice-and-dice algorithm, using natural metrics for smoothness of updates and overall aspect ratio. The results suggest that ordered treemaps steer a middle ground, producing layouts with aspect ratios that are far lower than slice-and-dice layouts, though not as quite as low as cluster or squarified treemaps; they update significantly more smoothly than clustered or squarified treemaps, though not as smoothly as slice-and-dice layouts. Thus ordered treemaps may be a good choice in situations where legibility, usability and smooth updating all are important concerns.

2. Algorithms for ordered treemaps

The key insight that leads to algorithms for ordered treemaps is that it is possible to create a layout in which items that are next to each other in the given order are adjacent in the treemap. Although such a layout does not follow the simple linear order of the slice-and-dice layout, it provides useful cues for locating objects and turns out to provide constraints on the layout that discourage large discontinuous changes with dynamic data.

We discuss two closely related algorithms for creating layouts that approximately preserve order. Both follow a similar recursive process, inspired in part by the idea of finding a two-dimensional analogue of the well-known Quicksort algorithm.

The inputs are a rectangle R to be subdivided and a list of items that are ordered by an index and have given areas. The first step is to choose a special item, the *pivot*, which is placed at the side of R. In the second step, the remaining items in the list are assigned to three large rectangles that make up the rest of the display area. Finally, the algorithm is then applied recursively to each of these rectangles.

In the first algorithm, pivot-by-size, the pivot is taken to be the item with the largest area. The motivation for this choice is that the largest item will be the most difficult to place, so it should be done first. The algorithm, as illustrated in Fig. 3, can be described as follows:

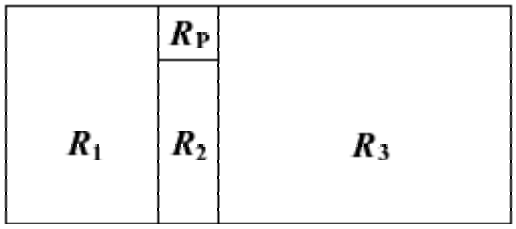

Figure 3. The pivot configuration.

1. Let P, the pivot, be the item with the largest area in the list of items.
2. If the width of R is greater than or equal to the height, divide R into four rectangles, R_1, R_P, R_2, and R_3 as shown in Fig. 2. (If the height is greater than the width, use the same basic arrangement but flipped along the line $y=x$.)

3. Place P in the rectangle R_P, whose exact dimensions and position will be determined in Step 4.
4. Divide the items in the list, other than P, into three lists, L_1, L_2, and L_3, to be laid out in R_1, R_2, and R_3. L_1 and L_3 all may be empty lists. (Note that the contents of these three lists completely determine the placement of the rectangles in Figure 3.) Let L_1 consist of all items whose index is less than P in the ordering. Let L_2 and L_3 be such that all items in L_2 have an index less than those in L_3, and the aspect ratio of P is as close to 1 as possible. We add the proviso, to avoid degenerate layouts, that L_3 cannot contain exactly one item.
5. Recursively lay out L_1, L_2, and L_3 (if any are non-empty) in R_1, R_2, and R_3 according to this algorithm.

The second ordered treemap algorithm, pivot-by-middle, is almost identical except that the pivot is taken to be the middle item of the list - that is, if the list has n items, the pivot is item number n/2, rounded down. The motivation behind this choice is that it is likely to create a balanced layout. In addition, because the choice of pivot does not depend on the size of the items, the layouts created by this algorithm may not be as sensitive to changes in the data as pivot by size. Figure 4 shows examples of the layouts created by the two algorithms.

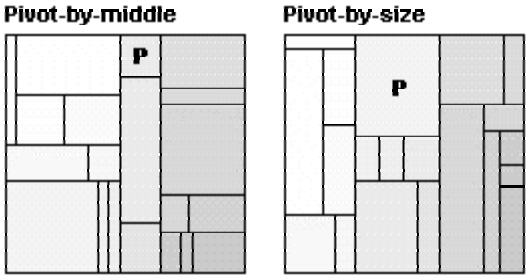

Figure 4. Pivot layouts. Shading indicates order, which is roughly preserved. The "P" indicates the first pivot rectangle in each layout.

Both algorithms have the property that they create layouts that roughly preserve the ordering of the index of the items, which will fall in a left-to-right and top-to-bottom direction in the layout. The two algorithms are also reasonably efficient: pivot-by-size has performance characteristics similar to QuickSort (order n log n average case and n^2 worst case) while pivot-by-middle has order $n$log n performance in the worst case.

Although the two algorithms produce layouts with relatively low aspect ratios (as described in the following sections) they are not optimal in this regard. The stipulations in step 4 of the algorithm avoid some but not all degenerate layouts with high aspect ratios, so we experimented with post-processing strategies designed to improve the layout aspect ratio. For example we tried adding a last step to the algorithm in which any rectangle that is divided by a segment parallel to its longest side is changed so that it is divided by a segment parallel to its shortest side. Because this step gave only a small improvement in layout aspect ratio while dramatically decreasing layout stability, we did not include it in the final algorithm.

3. Metrics for treemap layouts: aspect ratio & change

In order to compare treemap algorithms we define two measures: the average aspect ratio of a treemap layout, and the layout distance change function, which quantify the visual problems created by poor layouts. The goal is to have a low average aspect ratio and a low distance change as data is updated.

We define the average aspect ratio of a treemap layout as the unweighted arithmetic average of the aspect ratios of all leaf-node rectangles. This is a natural measure, although certainly not the only possibility. One alternative would be a weighted average that places greater emphasis on larger items, since they contribute more to the overall visual impression. We choose an unweighted average since the chief problems with high aspect ratio rectangles—poor visibility and awkward labeling—are at least as acute for small rectangles as large ones.

The layout distance change function is a metric on the space of treemap layouts that allows us to measure how much two layouts differ, and thus how quickly or slowly the layout produced by a given algorithm changes in response to changes in the data. To define the distance change function, we begin by defining a simple metric on the space of rectangles. Let a rectangle R be defined by a 4-tuple (x,y,w,h) where x and y are the coordinates of the upper left corner and w and h are its width and height. We use the Euclidean metric on this space, i.e. if rectangles R_1 and R_2 are given by (x_1, y_1, w_1, h_1) and (x_2, y_2, w_2, h_2) respectively, then the distance between R_1 and R_2 is given by

$$d(r_1, r_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (w_1 - w_2)^2 + (h_1 - h_2)^2}$$

We use this metric since it takes into account the visual importance of the shape of a rectangle, but there are several plausible alternatives to this definition. Two other natural metrics are the Hausdorff metric for compact sets in the plane or a Euclidean metric based on the coordinates of the lower right corner instead of height and width. These metrics differ from the one we chose by a small bounded factor, and hence would not lead to significantly different results.

We then define the layout distance change function as the average distance between each pair of corresponding rectangles in the layouts. We use an unweighted average

for the same reasons as we use an unweighted average for aspect ratios.

4. Experimental design and results

To compare the performance of ordered treemap layout algorithms to squarified, cluster and slice-and-dice layouts, we ran two sets of experiments. The first consisted of a sequence of Monte Carlo trials to simulate continuously updating data. Our goal was to measure the average aspect ratio and average layout distance change produced by each of five algorithms. In the second experiment we measured the average aspect ratio produced by each of the algorithms for a static set of stock market data.

4a. Monte Carlo trials

To simulate the performance of the five layout algorithms under a variety of conditions, we performed experiments on two types of hierarchies with two different statistical distributions of item sizes. The first hierarchy ("20x1") was a collection of 20 items with one level of hierarchy. The second ("8x3") was a balanced tree with three levels of hierarchy and eight items at each level for a total of 512 items.

For each experiment we ran 100 trials of 100 steps each. In one experiment we began with data drawn from a log-normal distribution created by exponentiating a normal distribution with mean 0 and variance 1. In a second version, we used data drawn from a Zipf distribution [R97] with power parameter equal to 1. Both distributions are representative of naturally occurring positive-valued data [R97]. In each step of a trial the data was modified by multiplying each data item by a random variable e^x, where x was drawn from a normal distribution with variance 0.05 and mean 0, thus creating a log-normal random walk. All layouts were created for a square with side 100.

The results are shown in tables 1 through 4.

Table 1: 20x1, Log-normal initial distribution.

Algorithm	Aspect Ratio	Change
Slice-and-dice	56.54	0.52
Pivot-by-middle	3.47	3.06
Pivot-by-size	3.15	7.17
Cluster	1.72	11.00
Squarified	1.75	10.10

Table 2: 8x3, Log-normal initial distribution.

Algorithm	Aspect Ratio	Change
Slice-and-dice	26.10	0.46
Pivot-by-middle	3.97	1.08
Pivot-by-size	3.14	4.07
Cluster	1.79	7.67
Squarified	1.74	8.27

Table 3: 20x1, Zipf initial distribution.

Algorithm	Aspect Ratio	Change
Slice-and-dice	36.85	0.51
Pivot-by-middle	2.70	2.91
Pivot-by-size	2.58	6.86
Cluster	1.75	12.57
Squarified	1.38	11.71

Table 4: 8x3, Zipf initial distribution.

Algorithm	Aspect Ratio	Change
Slice-and-dice	44.58	0.61
Pivot-by-middle	4.54	1.57
Pivot-by-size	3.85	4.10
Cluster	1.78	6.19
Squarified	1.67	7.18

The results strongly suggest a tradeoff between low aspect ratios and smooth updates. As expected, the slice-and-dice method produces layouts with high aspect ratios, but which change very little as the data changes. The squarified and cluster treemaps are at the opposite end of the spectrum, with low aspect ratios and large changes in layouts. The two ordered treemaps fall in the middle of the spectrum. Neither produces the lowest aspect ratios, but they are a clear improvement over the slice-and-dice method, with the pivot-by-largest algorithm producing slightly better aspect ratios. At the same time, they update more smoothly than cluster or squarified treemaps, with the pivot-by-middle algorithm having a slight advantage over pivot-by-largest.

4b. Static stock market data

Our second set of experiments consisted of applying each of the five algorithms to a set of 535 publicly traded companies used in the SmartMoney Map of the Market [W98] with market capitalization as the size attribute. For each algorithm we measured the aspect ratio of the layout it produced. The results are shown in the first column of Table 5, and the layouts produced are shown in Figures 5-9 at the end of this paper. (The gray scale indicates ordering within each industry group which is the last level of hierarchy in this data set.) Note that although aspect ratios are higher than in the statistical trials-partly due to outliers in the data set-the broad pattern of results is similar.

Table 5: Stock market data for 535 companies.

Algorithm	Aspect Ratio
Slice-and-dice	369.83
Pivot-by-middle	19.30
Pivot-by-size	22.04
Cluster	3.74
Squarified	3.21

5. Conclusion and future directions

Treemaps are a popular visualization method for large hierarchical data sets. Although researchers have recently created several algorithms that produce create clear, legible treemap layouts with low aspect ratios, these new algorithms have two drawbacks: they are unstable under updates to the data, and they scramble any natural order on the items being mapped.

We introduced ordered treemaps, which alleviate these problems by creating layouts that preserve order and that update cleanly for dynamically changing data. Experimental results show that they offer a useful compromise between the smooth updates of the slice-and-dice method and the low aspect ratios of cluster treemaps and squarified treemaps.

There are several directions for future research. First, there is doubtless room to optimize the ordered treemap algorithms discussed in this paper, especially to improve the overall aspect ratios they produce. It would also be useful to optimize the algorithms used by cluster treemaps and squarified treemaps to improve stability under dynamic updates. Another practical area to explore would be matching layout algorithms to particular statistical distributions of data and changes in the data, since different algorithms may be appropriate in different situations.

More speculatively, since experimental results suggest a tradeoff between aspect ratios and smoothness of layout changes, it would be worthwhile to look for a mathematical theorem that makes this tradeoff precise. It might also be fruitful to explore variants of treemap layouts that can update smoothly by using past layouts as a guide to current ones, or by using tiles that can have nonrectangular shapes.

Figure 5. Stock portfolio with slice-and-dice layout.

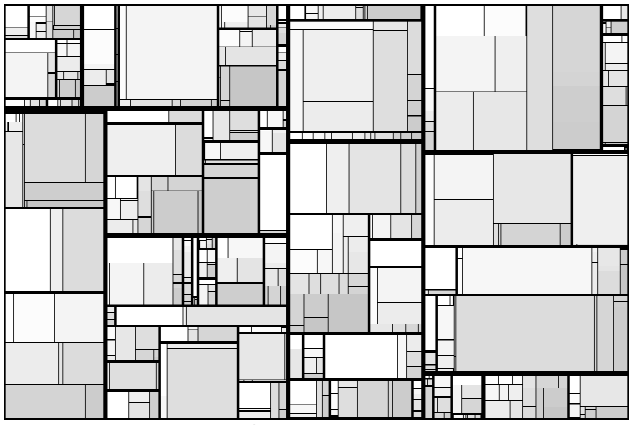

Figure 6. Stock portfolio with pivot-by-middle layout.

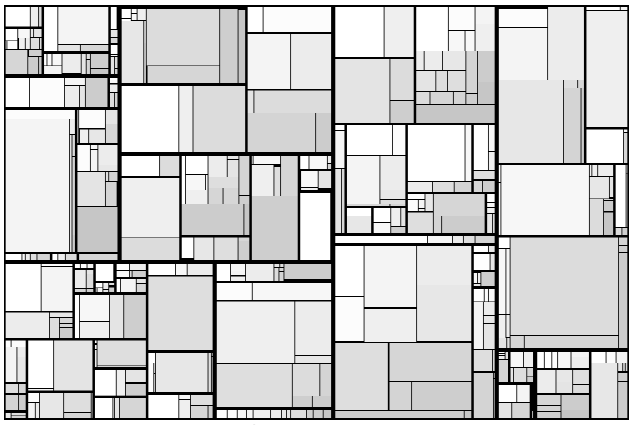

Figure 7. Stock portfolio with pivot-by-largest layout.

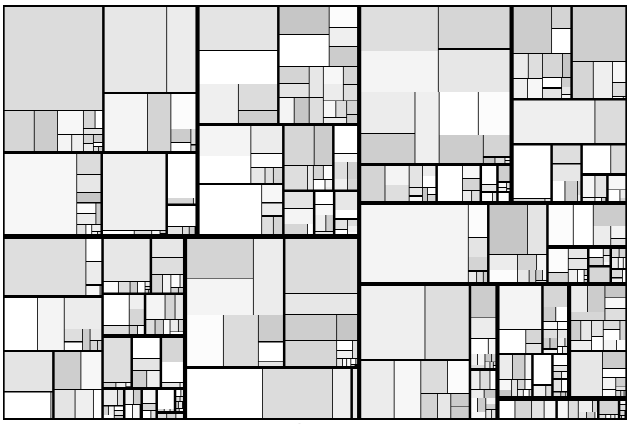

Figure 8. Stock portfolio with cluster layout.

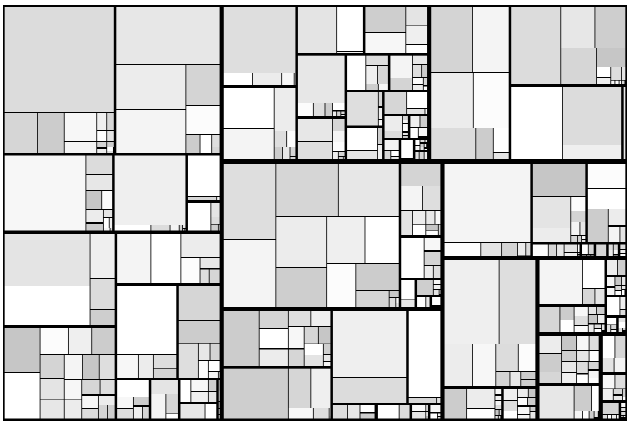

Figure 9. Stock portfolio with squarified layout.

Acknowledgements: Thanks to Ben Bederson and the reviewers for thoughtful suggestions on the draft.

6. References

[BHW00] Bruls, D.M., C. Huizing, J.J. van Wijk. "Squarified Treemaps". In: W. de Leeuw, R. van Liere (eds.), *Data Visualization 2000, Proceedings of the joint Eurographics and IEEE TCVG Symposium on Visualization*, 2000, pp. 33-42.

[JB97] Jin, L. and Banks, D.C. "TennisViewer: A Browser for Competition Trees." *IEEE Computer Graphics and Applications*, July/August, pp. 63-65, 1997.

[JS91] Johnson, B. and Shneiderman, B. "Tree-maps: A Space-filling Approach to the Visualization of Hierarchical Information Structures." *Proceedings of the IEEE Visualization '91*, pp. 284-291, October 1991.

[J94] Johnson, B. "Treemaps: Visualizing Hierarchical and Categorical Data" Unpublished PhD dissertation, Dept. of Computer Science, University of Maryland, College Park, MD (UMI-94-25057), 1994.

[JT92] Jungmeister, W-A. and Turo, D. "Adapting Treemaps to Stock Portfolio Visualization," University of Maryland Technical Report CS-TR-2996, 1992. Available at http://www.cs.umd.edu/hcil/pubs/tech-reports.shtml

[R97] Sheldon, R. *A First Course in Probability*, Prentice Hall 1997.

[S92] Shneiderman, B. "Tree Visualization with Tree-Maps: 2-d Space-filling Approach." *ACM Transactions on Graphics*, 11(1), pp. 92-99, 1992.

[TJ92] Turo, D. and Johnson. B. "Improving the Visualization of Hierarchies with Treemaps: Design Issues and Experimentation." *Proceedings of the IEEE Visualization* 92, pp. 124-131, October 1992.

[VN00] Vernier, F. and Nigay, L. "Modifiable Treemaps Containing Variable-Shaped Units", *Extended Abstracts of the IEEE Information Visualization* 2000. Available at http://iihm.imag.fr/publs/2000/Visu2K_Vernier.pdf

[W98] Wattenberg, M. "Map of the Market. "*SmartMoney.com*, http://smartmoney.com/marketmap, 1998.

[W99] Wattenberg, M. "Visualizing the Stock Market," *Proceedings of ACM CHI 99, Extended Abstracts*, pp.188-189, 1999.