

A Note on Space-Filling Visualizations and Space-Filling Curves

Martin Wattenberg, IBM Research

ABSTRACT

A recent line of treemap research has focused on layout algorithms that optimize properties such as stability, preservation of ordering information, and aspect ratio of rectangles. No ideal treemap layout algorithm has been found, and so it is natural to explore layouts that produce non-rectangular regions. This note describes a connection between space-filling visualizations and the mathematics of space-filling curves, and uses that connection to characterize a family of layout algorithms which produce nonrectangular regions but enjoy geometric continuity under changes to the data and legibility even for highly unbalanced trees.

CR Categories and Subject Descriptors: Hierarchy Visualization

1 INTRODUCTION

Space-filling visualizations, such as treemaps [12], are a widely used type of information display. These techniques scale well, with the capacity to show thousands of items legibly. The key difference among various space-filling techniques is the method by which they partition the screen.

At the core of a space-filling visualization is a *layout function* L that takes as an argument a list of k non-negative real numbers, (x_1, x_2, \dots, x_k) , and returns a corresponding partition of a rectangle into regions (r_1, r_2, \dots, r_k) , where $Area(r_i)$ is proportional to x_i . Typically the regions are assumed to be disjoint or have zero-area intersection. For hierarchical data, one typically then divides the regions into subregions, based on an identical or similar layout function.

Unfortunately, not all functions L make for good displays. In the case of treemaps, while many layout functions have been proposed [2,3,4,12,13,14,15], all suggested functions have drawbacks: [3] discusses various potential problems, such as lack of continuity of L or highly elongated shapes for the output regions. A natural question is whether these problems arise because treemaps are constrained to produce rectangular regions whose union is the unit square. For instance, in a treemap it is considered bad to have a rectangle that has a very high aspect ratio—yet to represent the list of numbers (999,1) a treemap must have an item whose aspect ratio is 1,000. While this may seem like a contrived example, it does arise in the real world; one case is in visualizing the highly unbalanced trees that are often produced by hierarchical agglomerative clustering.

Given the intrinsic limitations of rectangular layouts, one might ask what might be gained by giving up the condition that regions be rectangular. Indeed, several nonrectangular space-filling visualizations have been developed, including quantum bubblemaps [3], the polygonal maps of Nguyen and Huang [11], and Voronoi Treemaps [1]. While these maps enjoy helpful visual attributes, their continuity properties are unclear.

In this note we write down a list of natural but strong conditions for a layout function L , and ask whether it is possible to meet these goals by allowing nonrectangular—even nonconvex—regions. Following an insight of Keim [8], we show that it is indeed possible, using a type of layout we term a *jigsaw map* due to the resemblance of its regions to jigsaw puzzle pieces, and we

show examples of jigsaw maps applied to highly unbalanced trees that arise in bioinformatics.

Although their practical value still needs to be evaluated, jigsaw maps serve a theoretical purpose as well: We prove a theorem that any layout algorithm that meets every property in our list of desirable attributes is essentially a jigsaw map, thus providing a complete characterization of such layout functions in terms of discrete space-filling curves.

2 PERFECT LAYOUT FUNCTIONS

2.1 Desirable layout properties

The efficacy of a space-filling visualization depends on the layout function L . Experience with treemaps has uncovered a variety of desirable attributes for a layout function. In this section we describe these properties informally and discuss why they are important; in the next section we make them mathematically precise.

The first treemap layout [12] was known as the “slice and dice” method, and in practice led to layouts with rectangles with extremely high aspect ratios. Such long, skinny rectangles are difficult to see and select, and subsequent work on treemap layout algorithms discussed new algorithms which produced rectangles with a more compact shape, that is, aspect ratios closer to 1. See for example the squarified layouts of [4] or the recursive “cluster” layout of the SmartMoney Map of the Market [15].

These new algorithms turn out to share some new drawbacks: as the sizes of items in the layout changed, the structure of the layout itself changes discontinuously. Since many treemaps are designed to show time-varying data, such instability can be a significant distraction. In addition, since both the cluster and squarified algorithms involve a step which sorts elements by size, any underlying ordering of the items is destroyed.

As items are added or removed from the input list, the layout can change dramatically. Because the structure of treemap data often changes over time as well (with, for example, companies in the stock market splitting and merging) it would be nice if, when an item splits into two smaller items of the same total size as the original, that the rest of the layout shouldn't be affected.

To sum up, four desirable properties of a layout function are nicely shaped regions, stability with regard to changing leaf values, stability with regard to changing tree structure, and preservation of ordering information. We now make these desiderata precise.

2.2 Discretizing the Problem

Layout functions are often defined in terms of real numbers. In what follows we consider instead a discrete version of the problem, defining layouts that partition a square mesh of pixels with integer coordinates. To be precise, we define a layout function L to be a function whose domain is sequences (x_1, x_2, \dots, x_k) , of integers whose sum is n^2 , and whose range is sequences of

disjoint connected subsets (r_1, r_2, \dots, r_k) of the mesh $(1, 2, \dots, n) \times (1, 2, \dots, n)$, such that $|r_i| = x_i$. For simplicity we have defined the layout function on a square mesh, but as discussed below we do not lose any generality by doing so. In practice it is unlikely that the item sizes x_i will just happen to add up to n^2 , so the values will need to be normalized appropriately, with care taken so that round-off errors do not accumulate.

We shift to the discrete perspective partly because it simplifies the exposition and partly to more accurately model a computer screen made from individual pixels. Note that [3] discussed a similar “quantum” layout; there, the goal was to pack pictures into a readable grid. Since every space-filling algorithm must at least implicitly work this way in practice, even if only during the rasterization process, this is a reasonable perspective to take.

2.3 Perfect layouts

Based on the considerations described in the introduction, we can list four generally desirable properties for a layout function. In what follows, let L be a layout function and $x = (x_1, x_2, \dots, x_k)$ be an arbitrary input vector with $L(x) = (r_1, r_2, \dots, r_n)$.

1. Stability: Let y be any input vector of the same length as x , differing only at two positions, i and j , with $x_i = y_i + 1$ and $x_j = y_j - 1$. Then corresponding regions in $L(x)$ and $L(y)$ should differ by at most two items. This may be viewed as a discrete version of a continuity condition, and says that the smallest possible change in inputs should yield the smallest possible change in outputs.

2. Split Neutrality: Suppose $y = (y_1, y_2, \dots, y_{k+1})$ is an input vector such that for some $j < n$ we have $x_i = y_i$ for $i < j$, $x_j = y_j + 1$ for $i > j + 1$, and $x_j = y_j + y_{j+1}$. Let $L(y) = (s_1, s_2, \dots, s_{n+1})$. Then we require $r_i = s_i$ for $i < j$, $r_j = s_j + 1$ for $i > j + 1$, and $r_j = y_j$ union y_{j+1} . Informally, this says that structural changes are handled smoothly: if we have a set of items, and one of those items is replaced by (“splits into”) two items whose sum is the original one, then in the corresponding layouts, the region for the original item splits into two subregions, with no other region affected.

3. Order Adjacency: Because order is often important, r_i should be adjacent to r_{i+1} for each $i < k$.

4. c -Locality: The diameter of a region r_i should be bounded by a small constant c times the square root of the area of r_i . In other words, layout regions should be relatively compact, rather than long and narrow, to aid in seeing and selecting regions. This is the analogue for nonrectangular shapes of the treemap criterion that aspect ratios should be close to 1. (Bederson et al, 2002)

Properties 1 and 2 ensure that small changes in the underlying data will lead to small changes in the corresponding layout. Property 1 has been discussed previously with regard to treemaps in [3]. Property 2 is necessary to discuss smooth change as items are inserted or deleted in the input lists. Property 3 requires some sense of sequence to be preserved. It is an interesting question to what extent these properties are redundant. It is fairly easy to see that split-neutrality and order-adjacency imply stability (using an argument similar to the proof of Theorem 2, below) but a more challenging question is whether stability and order adjacency imply some type of split-neutrality.

Property 4 essentially says that the regions cannot be too long and skinny—one might think of this as an anti-gerrymandering

provision. It is the analogue of the restrictions on aspect ratios that arise when discussing good treemap layouts.

If a layout function satisfies all four properties, we call it *perfect*. (Here “perfect” is used in a mathematical sense of completely satisfying the list of properties; we make no claim that a perfect layout is the best way to present any particular data set.) No layout algorithm in the literature satisfies all four properties. Most layout algorithms currently used in treemaps (squared, cluster, and strip [3]) are not continuous, for example. Squarified and cluster treemaps are not order-preserving and not split-neutral, since they rearrange items by size. The very simple algorithm used in each level of a slice-and-dice layout does satisfy the first three properties but does not enjoy the c -locality property.

In the introduction we discussed how the simple case of laying out two items with weights 1 and 999 with rectangles implies a high aspect ratio. Because of this simple obstacle (and others) to perfect rectangle-based layouts, one may ask whether any perfect layouts exist. The answer, as we now show, is yes.

2.4 Space-filling curves and mesh indices

To construct a perfect layout, we require the concept of a *space-filling curve*. A space-filling curve is a continuous function that, roughly speaking, maps a one-dimensional space onto a higher dimensional space. Such curves have been used in computer science for many applications due their good clustering properties; see for example [7] and [9]. They were introduced in the context of information visualization by Keim [8] who proposed using these curves to help place related items near to each other in a pixel-oriented layout. (Keim’s work differs from this paper in that he considers positioning large numbers of single-pixel items, rather than items of many different sizes.)

Keim uses a discrete version of a space-filling curve, which he terms a *screen-filling curve*. (Another term used in the literature is a *mesh index*.) We may define a screen-filling curve as a one-to-one function $h: (1, 2, \dots, n^2) \rightarrow (1, 2, \dots, n) \times (1, 2, \dots, n)$ with the “continuity” property that $h(i)$ and $h(i+1)$ are adjacent.

Many screen-filling curves are known that enjoy an additional strong *locality property*:

$$\text{Distance}(h(i), h(j)) < c|i-j|^{1/2}$$

for some small constant c . It is this locality condition which we use to construct layout functions that satisfy property 4. The classical Hilbert curve has $c = \sqrt{6}$ (see [10]) but better values are possible.

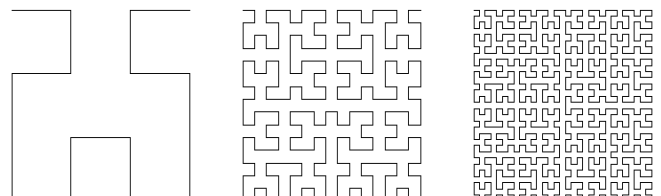
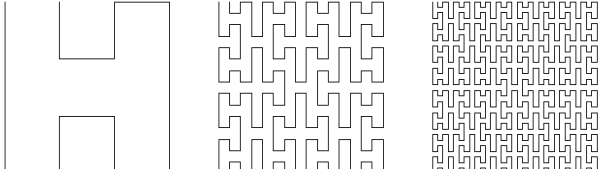


Figure 1. Some examples of Hilbert Curves.

The smallest known value of c , conjectured to be optimal, is 2, which holds for the so-called H-Curve [10]. (The H-Curve can be constructed for rectangles of arbitrary dimensions, as well as squares, which is why we have lost no generality in assuming the mesh is square.)

Figure 2. Some examples of H-Curves.



The diagram above shows three examples of H-Curves of different lengths on different meshes. Note that if there is a different number of items than grid elements—as must happen if there are a prime number of leaf nodes—there may be empty space resulting in a little empty “notch” in the on corner; see Figure 5 for an example of this discretization phenomenon.

2.5 The Jigsaw Map: A Perfect Layout Function

We now define the *jigsaw map* and prove it is a perfect layout function. The idea behind a jigsaw map is to solve a trivial one-dimensional layout problem and then map that one-dimensional layout into two dimensions using a screen-filling curve.

As before, let the input sequence be given by $x=(x_1, x_2, \dots, x_k)$, where $x_1 + x_2 + \dots + x_k = n^2$. We define the jigsaw layout function J using two other functions: a screen-filling curve satisfying c -locality, which we call H , and a trivial “one dimensional layout function,” g .

To define this second function, let $m_i = x_1 + x_2 + \dots + x_i$. Thus $m_1 = x_1$ and $m_k = n^2$. Let g be the function that maps x to the following sequences of subsets of $\{1, 2, \dots, n^2\}$:

$$g(x) = (\{1, 2, \dots, m_1\}, \{m_1+1, \dots, m_2\}, \dots, \{m_{k-1}+1, \dots, m_k\}).$$

Intuitively, we can think of g as a sort of “one dimensional” layout function that creates a perfect layout on a $1 \times n^2$ mesh. Now let

$$J(x) = H(g(x)),$$

where the function composition is taken to mean:

$$J(x) = (\{H(1), H(2), \dots, H(m_1)\}, \dots, \{H(m_{k-1}), H(m_{k-1}+1), \dots, H(m_k)\}).$$

THEOREM 1.

J is a perfect layout function.

PROOF.

Because of the “continuity” property of the screen-filling curve H , J must have the stability property and order-adjacency property. The c -locality property of H implies c -locality of J . It is easy to see that g is “split-neutral,” in the sense that splitting one of the x_i will not affect the partitions of the others, and therefore J will also be split-neutral. Hence J is a perfect layout function.

The diagram below is a schematic version of the process of creating a jigsaw layout. At left is a one-dimensional layout corresponding to the function g , at center is the same layout extended to a screen-filling curve, and at right is the actual space-filling result. Note that there is one point that we have skipped past in the construction, and that is how to use the function J to layout a hierarchy of objects, rather than a flat sequence. To do so is simple, however: simply create a one-dimensional layout using a depth-first ordering of leaf nodes, and then apply H .

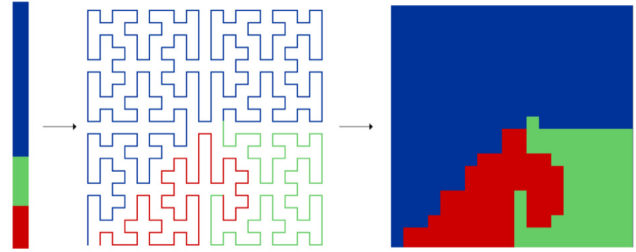


Figure 3. Using an H-Curve to Create a Jigsaw Map.

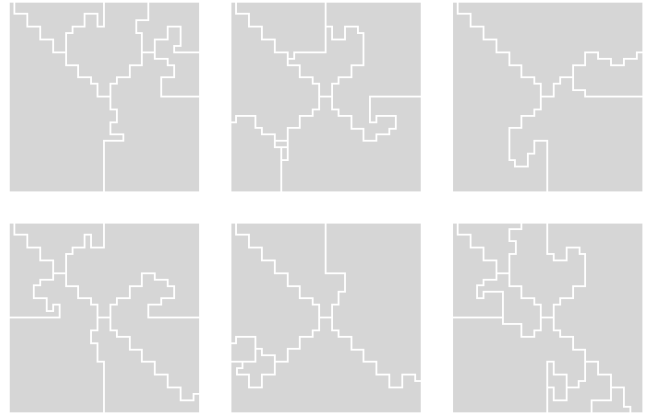


Figure 4. Sample Jigsaw Map Layouts.

2.6 All Perfect Layout Functions are Jigsaw Maps

Are there other perfect layouts besides the jigsaw construction described above? It turns out the answer is no: a converse to Theorem 1 holds.

THEOREM 2.

Any perfect layout function L can be written as $L = H(g)$, where H is a screen-filling curve and g is the one-dimensional layout function defined in the previous section. If L satisfies the c -locality property, the screen-filling curve H has the locality property that $\text{distance}(H(i), H(j)) < c|i-j|^{1/2}$.

PROOF.

Consider the input sequence S of n^2 ones. That is, $S = (1, 1, \dots, 1)$ with n^2 entries. Then for some integers $\{a_i\}$,

$$L(S) = (\{a_1\}, \{a_2\}, \dots).$$

Now we can “read off” the function H , and define:

$$H(i) = a_i.$$

We first show that $L = H(g)$. This follows from split-neutrality: for an arbitrary input $x = (x_1, x_2, \dots, x_k)$ we may first consider a sequence of “splits” of the input into $y = (1, 1, \dots, 1, x_2, \dots, x_k)$ where there are x_1 1’s at the beginning. We may then consider a second set of splits of y into the all-ones vector S . By definition, L maps the first x_1 1’s in S to the sequence $A = (\{a_1\}, \{a_2\}, \dots, \{a_{x_1}\})$. By split-neutrality, L must also map the first x_1 1’s in y to A . Finally, again by split-neutrality, the first element of $L(x)$ must be

$$\{a_1, a_2, \dots, a_{x_1}\} = \{H(1), H(2), \dots, H(x_1)\}.$$

In a similar fashion one can see the same is true for each x_i , and thus $L = H(g)$.

The remaining properties follow quickly. Because L is a layout function, h is one-to-one. By order-adjacency, H has the “continuity” property and is a screen-filling curve.

Finally, to see that H has the c -locality property, consider $i < j$. Let $d = j - i$. Consider the input (i, d, n^{2-j}) . L will map the middle item onto a region of size d that contains $H(i)$ and $H(j)$, which by c -locality of L implies $\text{distance}(H(i), H(j)) < c|j - i|^{1/2}$. This completes the proof of the theorem.

DISCUSSION. Theorem 1 and Theorem 2 give a fairly complete characterization of perfect layout functions, and allow us to phrase questions about perfect layouts (a relatively unstudied concept) in terms of the better-investigated area of space-filling curves. For example, the H-Curve is conjectured to have the best possible value of the locality constant c . If this conjecture is proven, it means that 2 is the best compactness constant for a perfect layout.

3 APPLICATIONS

3.1 An example using microarray data

The preceding discussion has been abstract, and the reader may be wondering whether the strong properties of a perfect layout function are more than just a mathematical curiosity. In this section we describe an application that was the inspiration for the jigsaw map method: creating a visualization of microarray data.

A detailed description of gene chip microarrays is beyond the scope of this paper, but generally speaking it is a technology for measuring the expression levels of many different genes under many different conditions. Microarray data typically takes the form of thousands of different vectors (one per gene) in a space with tens or even hundreds of dimensions (one per sample or experimental condition). Making sense of such data is difficult, but one standard technique is to organize the thousands of vectors using hierarchical clustering. The output of a hierarchical clustering is a binary tree, whose leaves correspond to particular vectors; such trees are usually highly unbalanced.

Visualizations of such trees have been standard among biologists since [5] introduced a combination heatmap/dendrogram designed to create a “map” of an entire set of hierarchically clustered vectors. Because such diagrams attempt to show all the data on one screen, they quickly become unwieldy. It is natural to ask whether the tree structure produced by the clustering can be used as part of a space-filling layout to show expression levels of genes in one particular sample. To see how jigsaw maps might work in this context, we apply the technique to data from [6], consisting of statistics for approximately 6000 yeast genes.

3.2 Treemap / Jigsaw Map Comparison

It is natural to use the tree structure as an input to a treemap, assigning each gene a weight of one. Unfortunately, the highly unbalanced binary trees produced by the hierarchical clustering represent a sort of perfect storm for traditional treemaps. Because the trees are binary, the traditional means of reducing aspect ratios—cluster, squarified, and strip layouts—all reduce to the slice-and-dice layout. The top row of Figure 5 (at end of paper) shows a treemap version of the data from [6]. At left is the “skeleton” of the layout, with higher-level branches outlined in darker colors; at right is a heatmap version of the same tree, showing expression level data with a red-black-green color scheme. It is clear that the treemap slice-and-dice layout produces

many high-aspect ratio rectangles. What is not visible is even worse: almost 10% of the rectangles are so thin that they are not even able to be drawn on the screen (and obviously would also be impossible to select with the mouse).

By contrast, the jigsaw map is easily able to show all the data without any distortion of leaf nodes. The bottom row of Figure 5 shows a jigsaw layout of the same data. The jigsaw layout allows every leaf node to be roughly square, and easily visible and selectable. In the heatmap version, the various regions with many genes that have been up- or down-regulated together are obvious.

At the same time, the irregular puzzle-piece shapes certainly look odd, and seem likely to make it more difficult to compare areas and understand the tree topology than a treemap does. Without further testing, it is hard to evaluate the tradeoffs. At the very least, due to Theorem 2, the jigsaw map can be viewed as an upper bound for visualizations based on perfect layouts.

4 CONCLUSION AND FUTURE DIRECTIONS

This paper has defined the notion of a “perfect” layout function: a layout method for a space-filling visualization that satisfies four desirable criteria. We constructed an example of such a layout, the jigsaw map, by using the notion of a screen-filling curve. We then proved that any perfect layout function is actually an instance of a jigsaw map, thus providing a complete characterization of perfect layouts in terms of screen-filling curves.

We finished by providing an example of the jigsaw map in action, displaying real-world data with a highly unbalanced tree structure derived from hierarchical agglomerative clustering. Such trees are common in the bioinformatics literature, and provide a natural application for jigsaw layouts. The example shows how jigsaw layouts do a significantly better job than a treemap at maintaining leaf nodes with decent aspect ratios.

There are several interesting future directions suggested by this work. One is to further explore the mathematics of perfect layouts. The criteria that define a perfect layout are extremely strict, and relaxing the criteria may make sense in certain cases. For example, we have adopted the strictest possible definition of stability, but it could easily be made less stringent. It is also unclear whether perfect layouts could be characterized by a simpler set of criteria than the ones given above. In particular, the split-neutrality condition seems very strong, but it may be unnecessary given stability and order-adjacency.

Another question is whether, if we look in the continuous domain, there is a related notion of perfect layout that allows for a layout function that creates convex polygons. It is likely that the split neutrality condition would need to be relaxed somewhat in this case.

More generally, it would be good to know more about how various layout criteria interact. For example, suppose that we relax the condition that all the space on the screen be used for data, but instead ask require that a fixed percentage be used? What if we allow slight overlaps between regions? Are there then good layouts with extremely regular regions such as circles and squares? A systematic investigation of the interplay between criteria for space-filling layouts promises to be fruitful.

5 ACKNOWLEDGMENTS

Thanks to R. Niedermeyer for use of Java code to produce an H-Curve, and to the anonymous referees for useful comments.

REFERENCES

- [1] Balzer M., Deussen O., 2005. Voronoi Treemaps for the Visualization of Software Metrics. *Proceedings of the 2005 ACM Symposium on Software Visualization*, pp. 165–172.
- [2] Bederson, B. (2001) PhotoMesa: a zoomable image browser using quantum treemaps and bubblemaps, *Proceedings of the 14th annual ACM symposium on User interface software and technology*,
- [3] Bederson, B., Shneiderman, B., Wattenberg, M. (2002) Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies. *ACM Transactions on Graphics* 21:4.
- [4] Bruls, M., Huizing, K., and van Wijk, J. J. (2000). Squarified treemaps. *Proceedings of Joint Eurographics and IEEE TCVG*
- [5] Eisen, M., Spellman, P., Brown, P. and Botstein, B. (1998) Cluster analysis and display of genome-wide expression patterns, *Proceedings of the National Academy of Sciences*.
- [6] Gasch, A., Spellman, P., Kao, C., Carmel-Harel, O., Eisen, M., Storz, G., Botstein D., Brown, P. (2000) Genomic expression programs in the response of yeast cells to environmental changes. *Molecular Biology of the Cell*: 11(12): 4241-4257.
- [7] Gotsman, C. and Lindenbaum, M.. (1996) On the Metric Properties of Discrete Space-Filling Curves. *IEEE Transactions on Image Processing*.
- [8] Keim, D. (1995) Enhancing the Visual Clustering of Query-Dependent Database Visualization Techniques Using Screen-Filling Curves, *Lecture Notes In Computer Science*; Vol. 1183, pp. 101 - 110. Springer-Verlag, London.
- [9] Moon, B., Jagadish, H. V., Faloutsos, C., and Saltz, J.H.. Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. *IEEE Transactions on Knowledge and Data Engineering*, 1996.
- [10] Niedermeier, R., Reinhardt, K., and Sanders, P. Towards Optimal Locality in Mesh-Indexings. *LNCS* 1279, 1997.
- [11] Nguyen, Q., Huang, M. (2002) A Space-Optimized Tree Visualization. *Proceedings of the IEEE Symposium on Information Visualization 2002 Symposium on Visualization (TCVG 2000)* IEEE Press, pp. 33–42.
- [12] Shneiderman, B., Tree visualization with tree-maps: 2-d space-filling approach, (1992) *ACM Transactions on Graphics*, 11:1.
- [13] Shneiderman, B., Wattenberg, M., Ordered Treemap Layouts, *IEEE Symposium on Information Visualization 2001*.
- [14] Vernier, F. and Nigay, L. 2000. Modifiable treemaps containing variable-shaped units *Proceedings of Extended Abstracts of IEEE Information Visualization 2000*.
- [15] Wattenberg, M., (1999) Visualizing the stock market, *CHI '99 extended abstracts on Human factors in computing systems*.

Figure 5. Top left: treemap skeleton layout (necessarily slice-and-dice, since the tree is binary). Top right: scalar data overlaid on this map. Bottom left: jigsaw skeleton of same tree. Bottom right: jigsaw map with scalar data.

