Cascaded Treemaps: Examining the Visibility and Stability of Structure in Treemaps

Hao Lü and James Fogarty

Computer Science & Engineering
DUB Group
University of Washington

{ hlv, jfogarty }@cs.washington.edu

ABSTRACT

Treemaps are an important and commonly-used approach to hierarchy visualization, but an important limitation of treemaps is the difficulty of discerning the *structure* of a hierarchy. This paper presents cascaded treemaps, a new approach to treemap presentation that is based in cascaded rectangles instead of the traditional nested rectangles. Cascading uses less space to present the same containment relationship, and the space savings enable a depth effect and natural padding between siblings in complex hierarchies. In addition, we discuss two general limitations of existing treemap layout algorithms: disparities between node weight and relative node size that are introduced by layout algorithms ignoring the space dedicated to presenting internal nodes, and a lack of stability when generating views of different levels of treemaps as a part of supporting interactive zooming. We finally present a two-stage layout process that addresses both concerns, computing a stable structure for the treemap and then using that structure to consider the presentation of internal nodes when arranging the treemap. All of this work is presented in the context of two large real-world hierarchies, the Java package hierarchy and the eBay auction hierarchy.

KEYWORDS: Cascaded treemaps, hierarchy visualization.

INDEX TERMS: H5.2. [Information Interfaces and Presentation]: User Interfaces – Interaction Styles.

1 INTRODUCTION AND MOTIVATION

Hierarchal structure is one of the most common approaches to organizing information, and so the effective visualization of hierarchical data has been the focus of significant research. Examples of previous work on hierarchy visualization include treemaps [16, 24], cone trees [23], hyperbolic trees [18], beamtrees [32], and radial visualizations [29].

Treemaps are a commonly-used approach based in dividing a display into nested rectangles, each with an area that corresponds to a weight associated with the node. In the original treemap work, for example, the contents of a hard disk were illustrated in a treemap with leaf nodes having areas corresponding to file sizes [16, 24]. Treemaps have since been widely adopted and applied to a variety of problems. For example, treemaps have been applied in Wattenberg's visualization of stock market data in the SmartMoney Map of the Market [35, 36], in Orso *et al.*'s visualization of program execution data [21], to present groups of related pictures in photo-browsing applications [3], to visualize Usenet activity [13, 26, 27], and in a visualization of currently popular items on the social bookmarking site del.icio.us [9]. Given the many approaches to hierarchy visualization, the widespread application of treemap techniques can be attributed to a combination of their effectiveness, their scalability, their aesthetic qualities, and the simplicity of their implementation.

A significant limitation of treemaps is the difficulty of discerning the *structure* of a hierarchy. Our review of related work discusses this limitation in greater detail, but it becomes most apparent when considering a large balanced tree. As noted by Van Wijk and Van de Wetering [33], such a case results in the original treemap degenerating to a regular grid, making it near impossible to determine the relative size of different portions of a hierarchy or to trace a path from a given node up to the root of a hierarchy. Because of this limitation, several common approaches are employed to better illustrate the internal nodes of a hierarchy and thus the structure of a treemap. *Nested treemaps* place a border around each internal node. Labels for internal nodes are typically placed either at the center of the internal node or at a location selected to minimize overlap with other labels. This approach minimizes the portion of a visualization that is dedicated to internal nodes, but can lead to overlapping or ambiguous label placement that makes it difficult to discern how a label relates to the treemap. *Labeled treemaps* therefore extend nested treemaps by both placing a border around internal nodes and further dedicating space to the consistent placement of labels, generally at the top of each node.

This paper discusses *cascaded treemaps*, an approach that illustrates the structure of hierarchies using a combination of layering and offsets. Cascaded presentations have appeared in prior work on browsing the National Science Foundation's funding hierarchy [19, 20] (under the name 2.5D treemaps, though we now use the term cascaded to avoid confusion with work by Turo and Johnson [30]). This paper goes beyond prior work by further contributing a detailed discussion of cascaded treemaps, a detailed discussion of node size distortion and interactive zooming in labeled treemaps, and an efficient two stage layout process that addresses both the distortion of leaf node sizes and support for stable interactive zooming.

We first discuss the relationship between cascaded treemaps and nested treemaps. Because cascading uses less space to present the same hierarchy information traditionally presented via nesting, cascaded treemaps create a depth effect and provide natural visual separation between siblings in a hierarchy. We next discuss distortions of leaf node size proportionality that are introduced by common approaches to treemap layout. Because existing layout algorithms generally do not explicitly consider the space that is dedicated to presenting internal nodes, large disparities can arise in the presentation of equally-weighted nodes. We then discuss the stability of treemap visualizations when supporting interactive zooming to examine portions of a hierarchy. Prior work has generally focused on generating a single view of a hierarchy, but large hierarchies inevitably introduce a need to support zooming to examine portions of the hierarchy that are rendered too small to

be useful. Finally, we present a two-stage layout process that addresses both the distortion of leaf node sizes and support for stable interactive zooming. Our approach is based in computing a layout for a hierarchy, then storing the structure of that layout together with the information necessary to efficiently and consistently present portions of the treemap.

2 Related Work

The initial treemap work used a slice-and-dice layout algorithm, alternating between horizontal and vertical arrangements of each level in a tree [16, 24]. Though trivial to implement, this often leads to nodes with very high aspect ratios, and the resulting long and skinny rectangles complicate labeling, comparisons, and interaction [30]. Cluster treemaps [36] and squarified treemaps [8] both minimize the occurrence of such high aspect ratio nodes by employing different strategies for choosing between horizontal or vertical arrangements within each node of a tree. Ordered treemaps add the additional constraint that nodes that are near each other in a provided ordering should be spatially near each other in the resulting treemap [5, 25]. This can prove useful, for example, in applying an alphabetical ordering to the nodes in a treemap for the benefit of a person searching for a node or to provide consistency in treemaps based on dynamically updating data, such as stock information. Quantum treemaps guarantee that every rectangle in a treemap will be a multiple of a given rectangle size, a property that is valuable when presenting images or other objects of indivisible size [3, 5]. While these variations on layout algorithms address important concerns, none of them are focused on the visibility of the structure of a hierarchy. Our discussion of cascaded treemaps is based in the squarified treemap layout algorithm, but our contributions are independent of the particular choice of layout algorithm.

Figure 1 presents an example of the need to illustrate the structure of a hierarchy. Originally presented by Van Wijk and Van de Wetering, this treemap illustrates a large balanced tree corresponding to an organizational chart. It is near impossible to compare the size of different units within the organization or to determine to which division a specific employee belongs. Van Wijk and Van de Wetering thus motivate cushion treemaps, a presentation based in shading a treemap as if it were a 3D surface illuminated from above [33]. Cushion treemaps do not require any additional pixels be dedicated to illustrating internal nodes, but they been found difficult to reliably interpret [32]. Nested treemaps are much more common, and Figure 2 gives an example of a nested treemap from Demian and Fruchter's work on Corporate Memory [10]. The hierarchy is presented using a combination of varying line thickness and padding between nodes. Although labels are given different colors at each level in the hierarchy and label placement is alternated in adjacent rectangles, this example illustrates a common difficulty of labels on one level of the hierarchy interfering with the visibility of labels on another level. Labeled treemaps therefore dedicate additional space to the consistent placement of labels. A common label placement is at the top of each internal node, as in Figure 3's illustration of the Java package structure [15] created using the University of Maryland reference Treemap implementation [24]. While this provides consistent label placement, we discuss several problems introduced by treemap layout algorithms that fail to consider the implications of dedicating large groups of pixels to presenting labels at many locations throughout a tree.

A number of other hierarchy visualization techniques have been developed [18, 23, 29, 32], including space-filling visualizations like step trees [6], Voronoi treemaps [2] and generalized treemaps [34]. Although relevant to hierarchy visualization, we pursue contributions that are sufficiently distinct from such work that we do not dwell on extensive comparisons.

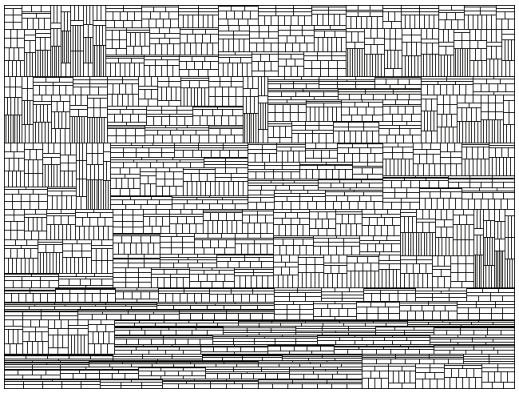

Figure 1. Van Wijk and Van de Wetering's example of treemaps degenerating to a uniform grid for a large balanced tree [33]. It is near impossible to compare the size of non-leaf nodes or to trace a path from a given node up to the root of the hierarchy.

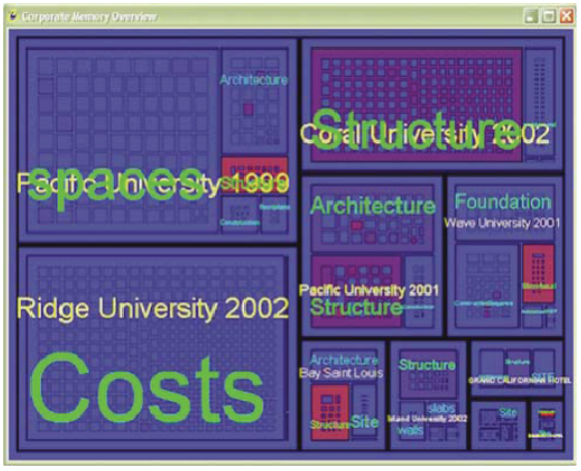

Figure 2. A nested treemap from Demian and Fruchter's exploration of effective nested treemap presentation in the Corporate Memory system [10]. Labels are given different colors at each level in the hierarchy and label placement is alternated in adjacent rectangles, but label overlap is a problem.

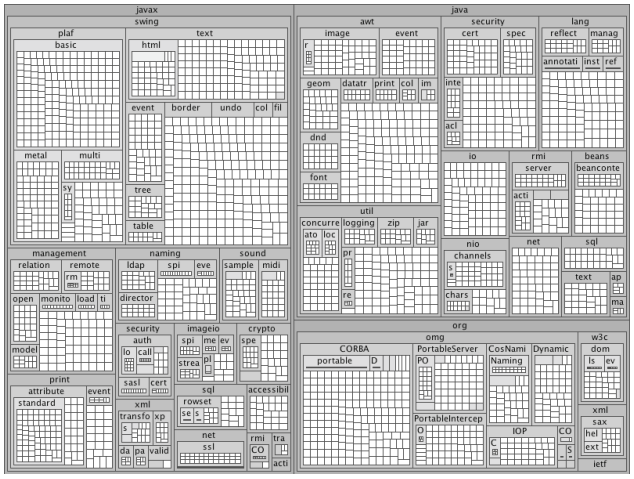

Figure 3. A labeled treemap generated by the University of Maryland's reference Treemap implementation [31], illustrating the Java package structure. Labels are consistently placed at the top of internal nodes, and shading is used to illustrate relative depth.

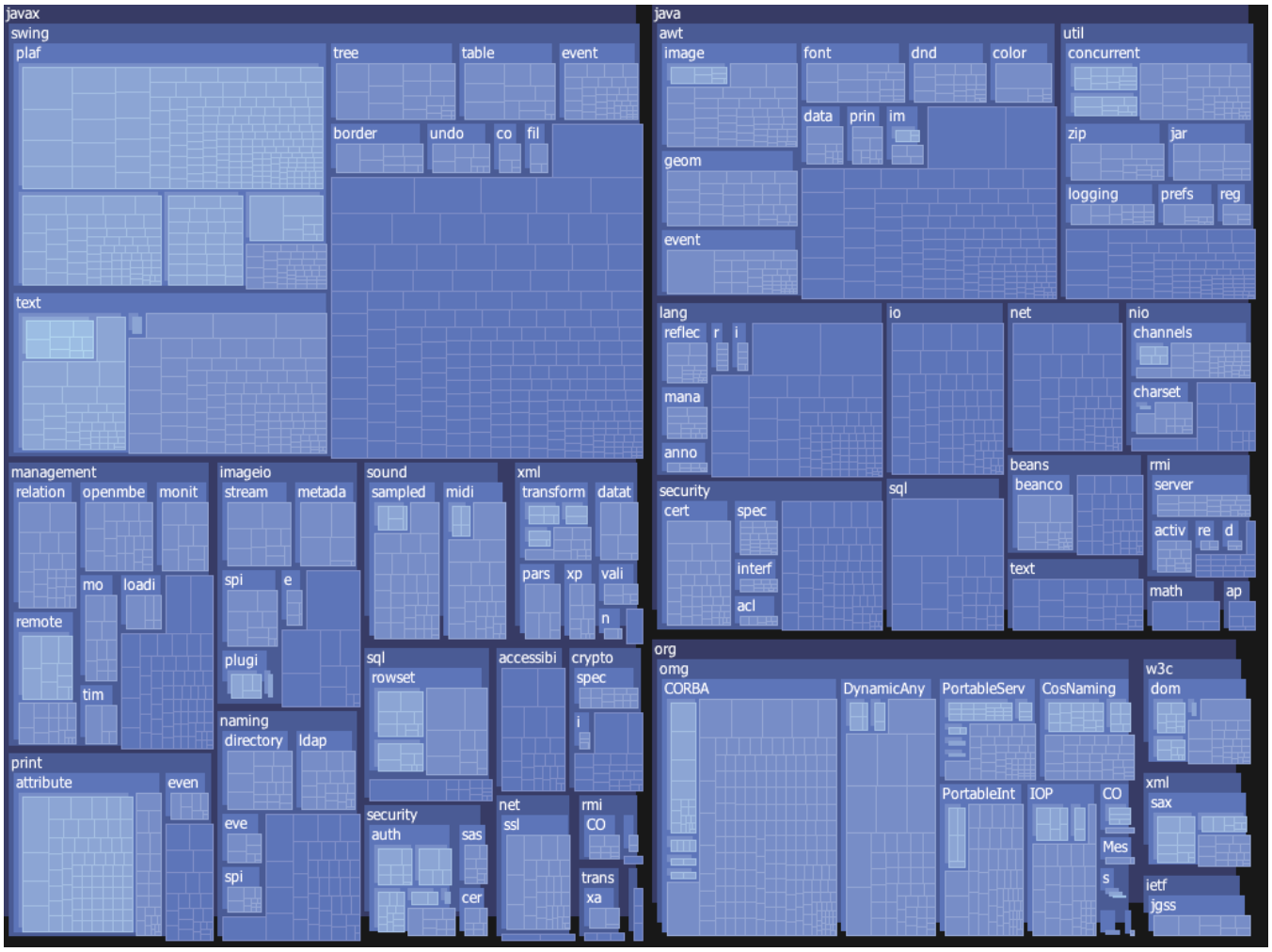

Figure 4. A cascaded treemap illustrating the Java package structure. As in Figure 3, labels are consistently placed and shading is used to illustrate relative depth. Our cascading effect uses less space to convey the same information typically conveyed via nesting, and the result is a clearer visualization that provides greater visual separation between siblings.

3 CASCADED TREEMAPS

Figure 4 presents an example cascaded treemap, implemented within the prefuse toolkit [14], illustrating the same Java package structure that was visualized in Figure 3. In contrast to the nesting strategy developed in the original treemap work [16, 24], our fundamental operation is a cascade, as illustrated in Figure 5. By shifting children down and to the right relative to their parent, cascading creates a depth effect resembling a stack of cards. As with nesting, this effect can be applied with or without the dedication of additional space for a label. In Figure 4, for example, the first three levels of the hierarchy are labeled. Below this, our cascade effect is applied but nodes are not labeled. Cascading therefore provides for the visibility of structure even in levels of the hierarchy below those that are labeled.

Cascading and nesting are different approaches to treemap *presentation*, while the related work discussed in the previous section has often focused on new approaches to treemap *layout*. A nested treemap created using any of the layout algorithms discussed in the previous section can be converted to a cascaded treemap via a straightforward bottom-up process given in Figure 6. Starting from the leaf nodes of the tree, each node is scaled down to the size of the bounding box of its children, then shifted to create the cascade effect. Depending on the relative sizes of the cascade offset and a node's children, it may then be

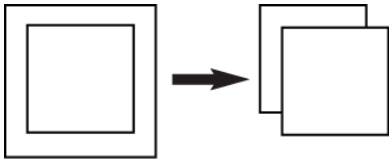

Figure 5. In contrast to nesting, our fundamental operation is a cascade to illustrate the structure of a hierarchy.

Converting a Nested Treemap to a Cascaded Treemap

NestedToCascaded(Rectangle **R**)
1. Apply *NestedToCascaded* to each child of **R**
2. If **R** has children then apply *Cascade* to **R**

Cascade(Rectangle **R**)
1. Calculate the bounding box of all children of **R**
2. Shrink **R** to the size of the bounding box
3. Shift **R** up and left by the cascading offsets
4. If needed, enlarge **R** to cover the center points of its children

Figure 6. A bottom-up procedure for converting a nested treemap presentation to a cascaded presentation.

necessary to grow the rectangle to cover the center points of the node's children. This final step ensures that a cascaded treemap has a unique interpretation, as every rectangle must overlap its parent. Even given this last step, it can be shown that this transformation process can only decrease the size of each rectangle in a nested treemap. A cascaded treemap can therefore always fit in the same amount of space used by a nested treemap.

In fact, it can be noted in Figure 5 that our cascading presentation uses less space to convey the same containment relationship typically conveyed by nesting. If we consider an offset of p pixels with a rectangle of size (w, h), using a nesting presentation yields an illustration of size ($w + 2p$, $h + 2p$) but a cascading presentation uses only size ($w + p$, $h + p$). Figure 7 takes the next step, illustrating the space savings obtained when using cascading to display multiple levels of containment.

Prior work has illustrated the structure of nested treemaps by using thicker line widths for rectangles nearer the root of a hierarchy or by adding additional padding between rectangles nearer the root of a hierarchy [13, 16, 24, 30]. This approach is effective, but necessarily reduces the amount of space available for illustrating a hierarchy's leaf nodes [10]. In contrast, the smaller presentation of containment in a cascaded treemap implicitly provides free padding between siblings in a hierarchy. Figure 8 illustrates the origin of this free padding. When two nested siblings are placed side-by-side, any desired padding must be explicitly added. But when two cascaded siblings are placed side-by-side in the same amount of space that would be occupied by a nested presentation, the cascading within the left sibling provides free padding between the siblings. It is this free padding that makes the structure of a cascaded treemap visible.

Furthermore, the free padding created by a cascaded presentation naturally accumulates to create a depth effect and to provide larger gaps between siblings nearer the root of a hierarchy. Figure 9 presents an example hierarchy constructed by Bruls *et al.* to illustrate the difficulty of understanding hierarchy structure in a nested treemap [8]. When using a nested presentation, identifying the relationships among nodes requires visually tracing maze-like lines. In contrast, all of the leaf nodes in the cascaded presentation are in the same locations as those in the nested presentation, but the cascaded effect naturally creates additional padding between the siblings closer to the root. This is, of course, the same space occupied by maze-like lines in the nested presentation, but the cascaded presentation seem to provide a less cluttered and more easily interpreted visualization.

Finally, although existing work on nested presentations links the visibility of a treemap's structure to larger nesting offsets and thicker borders closer to the root of a hierarchy, our informal experimentation suggests that cascaded treemaps are relatively effective even with thin borders and small offsets. Figure 10, for example, presents side-by-side illustrations of the same treemap using nested and cascaded presentations (intentionally omitting the coloring used in most of this paper). Both treemaps are drawn using one pixel borders and two pixel offsets. It informally appears easier to use the cascaded presentation to determine that there are two separate high-level nodes on the extreme left of these treemaps. It also informally seems easier to see that there is a group of nodes in the bottom right that have only the root as a common parent (as opposed to a false interpretation, which seems easier to make with the nested treemap, that this hierarchy first branches into two subtrees corresponding to two columns in the treemap). Further evidence cascading is effective with small offsets can be seen in Figure 4's illustration of the Java package hierarchy and Figure 14's illustration of the eBay auction hierarchy. This effectiveness would seem to be due to cascading's natural creation of a depth effect and additional padding between siblings closers to the root of complex hierarchies.

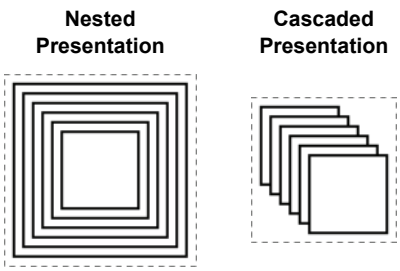

Figure 7. A cascaded presentation uses less space to show the same containment relationship conveyed by nesting.

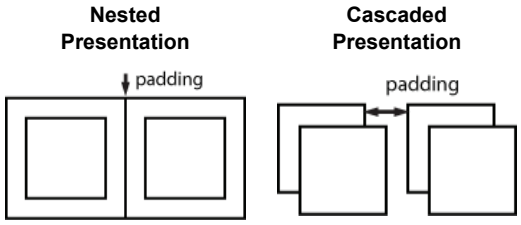

Figure 8. Although any desired padding must be explicitly added to a nested presentation, a cascaded presentation implicitly provides padding between siblings. This free padding is created by the cascading within the left sibling.

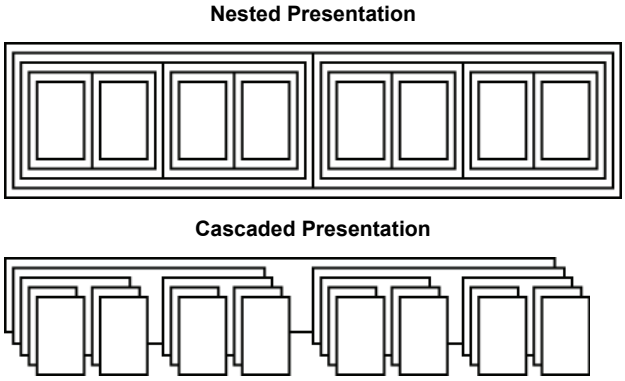

Figure 9. Bruls *et al.* construct this example to illustrate the difficulty of understanding structure in a nested presentation [8]. In contrast to the maze-like lines in the nested presentation, our cascaded presentation naturally creates additional padding between siblings closer to the root of complex hierarchies.

Figure 10. Cascaded presentation works well even when using small offsets. Note the relative ease of seeing the two separate high-level nodes on the extreme left of these treemaps, as well as the lack of a common parent for the nodes in the bottom right.

4 Distortions of Node Size and Missing Nodes Due to the Visualization of Structure in Treemaps

An intended advantage of treemaps is the ability to compare the area of leaf nodes across different portions of a treemap. This property should allow, for example, a person who is using a treemap visualization of the stock market to compare market capitalization of two companies in different sectors [35, 36]. Similarly, it should allow a person using a treemap visualization of Usenet to compare the level of activity in different areas of the Usenet hierarchy [13, 26, 27]. The original treemap presentation, without any visualization of the structure of internal nodes, does indeed provide this property [16, 24]. However, visualizations of structure (including even the use of single-pixel borders in nested treemaps) invalidate this property.

Figure 11 presents an example of a nested treemap where the size of leaf nodes cannot be reliably compared. This labeled treemap was generated using the University of Maryland's reference Treemap implementation [16], and it presents a portion of the package structure for a well-known Java toolkit. Node 1 has a weight of 151 units and occupies 286 pixels in the treemap. Node 2 is a similar size, occupying 210 pixels (73% of Node 1), but has a weight of only 18 units (12% of the weight of Node 1). In contrast, Node 3 has a weight of 17 units (94% of Node 2) but occupies only 36 pixels (17% of the space occupied by Node 2). Distortion of treemap node sizes is briefly mentioned by Demian and Fruchter [10], but they mention it only as a drawback that should be balanced against the benefits of increasing the nesting offset in a nested treemap. Demian and Fruchter neither probe the general cause nor offer a solution.

A more extreme version of the same problem can be found in Node 4. When presenting large hierarchies, it is often the case that not all of a hierarchy is visible. The most fundamentally unavoidable example is when a treemap layout algorithm reaches a point in the layout process where a subtree must be drawn within a single pixel. Given our interest in supporting such large hierarchies, the next section discusses issues of treemap stability during interactive zooming. In Figure 11, however, the problem is not simply that portions of the hierarchy are too small to be drawn. Instead, the problem is that Node 4 is not visible without first zooming into Node 1, but Node 4 has a weight almost twice that of Node 2 or Node 3. We refer to such a node as *missing*, as the node should be visible in a fair division of space and nodes smaller than it are actually visible. Missing nodes are not uncommon, and we later discuss treemaps for a real-world dataset that have hundreds of missing nodes. Despite this, missing nodes are not addressed in the existing treemap literature.

The general problem that causes both these distortions of node size and the problem of missing nodes is that existing treemap layout algorithms pay no attention to how much space within the treemap is dedicated to conveying structure. Consider, for example, the recursive layout algorithm used by squarified treemaps [8]. The layout function is given a set of weighted nodes and a rectangle in which to arrange them. Examining the node weights and the rectangle, the algorithm makes a horizontal or vertical division of the rectangle, divides the nodes amongst the resulting sub-rectangles, and recurses upon each.

What is missing from this process is a consideration of how much of each sub-rectangle will be dedicated to labels and borders within the subtree. If the current node is not a leaf, space will be needed for borders (exactly how much space will depend both on the presentation and the offset size) as well as for labels. Knowledge of the space required for labels is complicated by the fact that the decision to make a vertical or a horizontal division of the rectangle affects how much space is needed. This is because a vertical division means that labels can be placed in a row across the top of the resulting sub-rectangles, requiring only the height of a single label, while a horizontal division means that labels will be placed in a column at the top of each sub-rectangle, requiring space corresponding to the number of sub-rectangles times the height of a label. Attempting to fully resolve the implications of a horizontal or vertical division quickly leads to an exponential explosion in computation time [8], so existing layout algorithms ignore the dedication of space to labels. But this gives divisions based in over-estimating how much space is available for the display of a subtree, so labels push aside space intended to display a subtree and problems with distorted node sizes and missing nodes result. Section 6 will therefore present a two-stage layout algorithm that first decides whether to split rectangles horizontally or vertically, and then considers the need to dedicate space to structure while deciding how to size each sub-rectangle.

5 Stability and Interactive Zooming in Treemaps

A related concern is the stability of treemap layouts when used with interactive zooming. Zooming is a fundamental interaction [1, 4, 22], especially as large hierarchies inevitably result in portions of the hierarchy being rendered too small to be useful. Recent work by Blanch and Lecolinet has begun to address the shortage of work examining interaction and zooming within treemaps [7], but they do not address layout stability.

Simple pixel-level zooming is obviously inappropriate, as it results in a magnification not only of a treemap's content but also of the offsets between nodes and the space dedicated to labels.

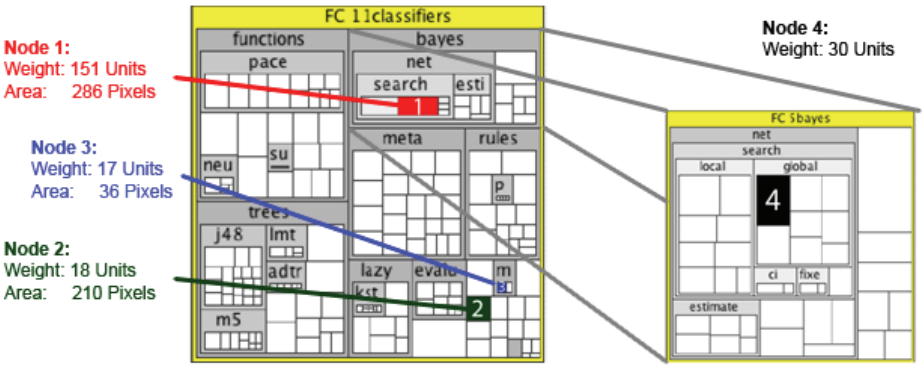

Figure 11. Existing treemap layout algorithms ignore the space dedicated to presenting structure. This leads to a distortion of node sizes and the notable complete absence of nodes. In this example, generated by the University of Maryland's reference Treemap implementation [31], there are gross disparities between the weights and the space allocated to Node 1, Node 2, and Node 3. Even worse, Node 4 has almost twice the weight of Node 2 or Node 3, but is not visible in the treemap with first zooming into Node 1.

The original treemap. Due in part to our cascaded presentation, much of the structure of the hierarchy under the node named “bayes” is already clearly visible.

Upon zooming to “bayes”, generating a new treemap re-arranges several nodes. This change in the structure may be jarring to a person and it cannot be easily animated.

By holding the structure of the tree constant, we provide a stable zoom into “bayes”. Nodes are in the same relative locations, and changes in size can be smoothly animated.

Figure 12. Our two-stage layout process also provides for stable interactive zooming in a treemap.

The most straightforward approach to zooming within a treemap is therefore to draw a new treemap rooted at the node into which a person zooms. The problem with this is illustrated in Figure 12 by considering a zoom into the node labeled “bayes.” In the leftmost image, much of the structure of the hierarchy under the “bayes” node is already visible. Generating a new treemap after zooming yields the middle image, in which the relative placement of several nodes has been re-arranged. This change in structure may be jarring, and it cannot be easily animated. We are therefore interested in a zoom that preserves the relative placement of nodes within a hierarchy, and so we refer to this as a stable zoom.

The difficulty with stable zooming is in many ways similar to difficulties with node size distortion and missing nodes. A zoom may change both the size and the aspect ratio at which a node should be rendered, as well as the level of detail at which each level in the hierarchy below it should be rendered. A label that once occupied half of the height dedicated to a node may now only occupy one quarter of the node (see “local” in Figure 12). Conversely, a node that previously displayed only a border may now also need to display a label (see “ci” and “fix” in Figure 12). Depending on the layout algorithm, such changes may result in different choices regarding whether to horizontally or vertically divide a rectangle within the treemap.

Our same two-stage layout process addresses the need for stable interactive zooming. Our process holds constant the decision to split rectangles horizontally or vertically, then sizes each sub-rectangle according to the current view of the hierarchy. This approach is able to account for changes in the size and aspect ratio at which a node should be rendered, as well as changes in how children are shown. This yields the zoom seen in the right image of Figure 12. Nodes are in the same relative locations as can be seen in the structure of the hierarchy before zooming, and so the resulting zoom can easily be smoothly animated.

6 Two-Stage Treemap Layout

We developed a two-stage treemap layout algorithm to address the concerns we have raised regarding treemap layout. As noted earlier, existing recursive algorithms simultaneously decide along what orientation and at what points to split the rectangle corresponding to each node in a hierarchy, recursing upon the resulting sub-rectangles. Our two-stage approach separates the decision regarding orientation and number of splits in each rectangle from the decision regarding where to place those splits. The orientation and number of splits in each node are computed once, and the placement of those splits is adjusted each time the treemap is arranged. The resulting process is analogous to initially placing a set of bars dividing the space allocated to the treemap, then sliding those bars around to create different presentations.

In more detail, we first apply the squarified treemap algorithm, parameterized by the size of the display area and using temporary cascading offsets and label sizes of zero [8]. This gives us the structure of the tree upon which we base our layouts. For each node in the tree, we save the orientation of the split and how the children of the node were partitioned by the split (which children go to the left and right or the top and bottom, depending on the orientation of the split). As in most treemap implementations, internal nodes also store the sum of the weights of their children.

Because the split orientations and the partitioning of children are now known, a recursive layout can appropriately account for the space needed to illustrate structure at each internal node. To arrange a node, the layout function first checks the stored split orientation. We suppose the split is horizontal, noting that vertical is similar. The function next computes how much vertical space is needed for offsets and labels above the split. Our prototype computes this by walking the subtree, but necessary information about the subtree could be cached based on the labeling policy for the treemap. The function then computes how much vertical space is needed for offsets and labels below the split. The remaining space is for the content of the treemap, and so the layout function gives each side of the split the space computed as necessary for labels and offsets as well as a portion of the remaining content space based on the relative weights of nodes on each side of the split. The layout procedure is then ready to recurse on both sides of the split, as it knows how much space will be used by labels and offsets and has ensured that the remaining space is appropriately divided by node weight. This knowledge of how space will be used ensures that no missing nodes will be created.

The only exception to this is when the labels and offsets needed by a treemap consume more space than is available. Note that the layout procedure already knows whether this is true before it recurses on a split. Missing nodes will be created if the sum of the space needed for labels and offsets above the split plus the space needed below the split exceeds the space available for a node. Computing this at the root of the hierarchy lets appropriate adjustments be made before layout begins. For example, the layout algorithm could change the labeling policy to reduce how many layers of the hierarchy are labeled. Similar insight could be used to automatically vary the level of labeling within a hierarchy to control how much of the treemap is dedicated to labels versus the illustration of leaf nodes.

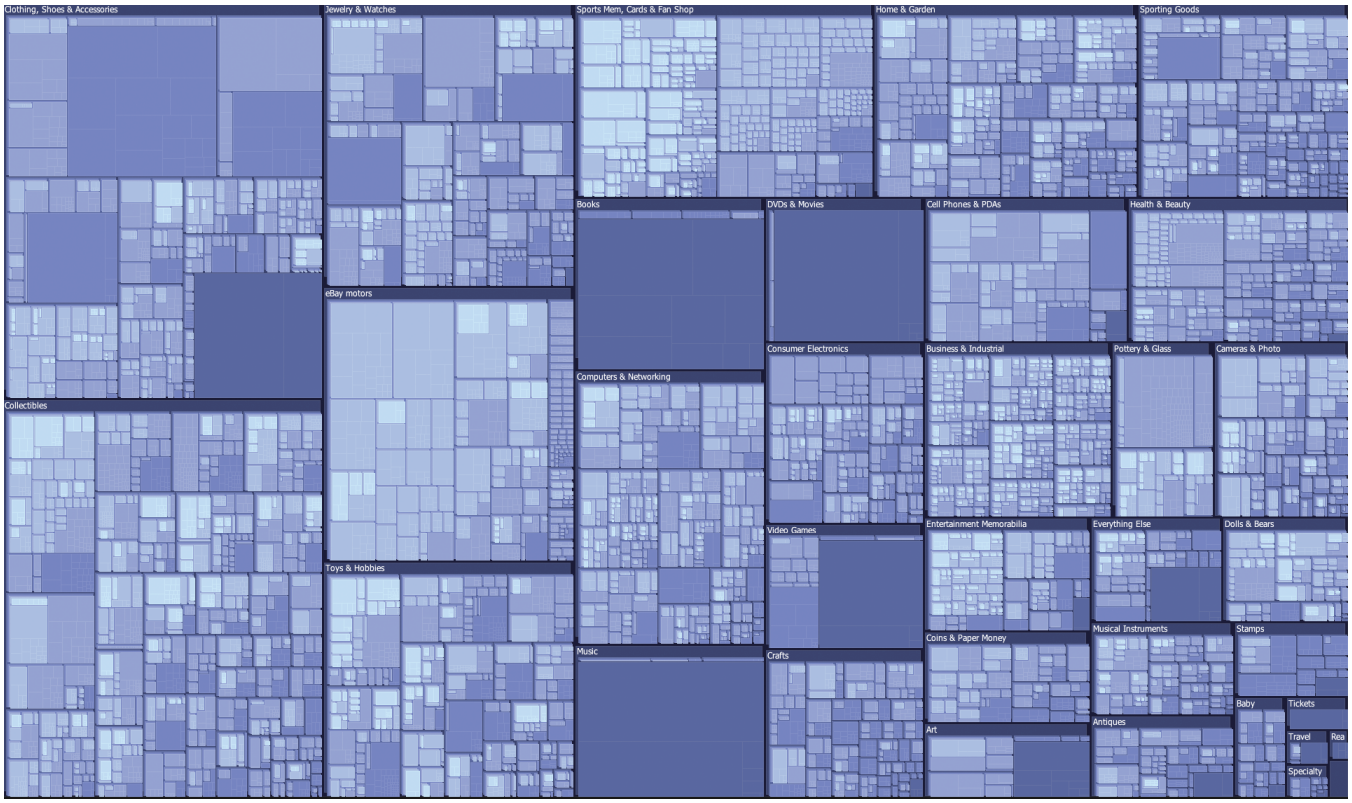

Figure 14. A cascaded treemap visualizing the entire eBay auction hierarchy. The weight of each leaf node corresponds to the number of active auctions.

7 Discussion and Conclusion

As a final example, Figure 14 presents a cascaded treemap illustrating the entire eBay auction hierarchy [11]. This hierarchy contains 30,380 nodes, including 25,964 leaf nodes. Only the first level of the hierarchy is labeled, and the remainder of the hierarchy is illustrated via cascading. Each leaf node corresponds to an auction category, and the weight of each leaf node is the number of active auctions in the category. Despite the large number of nodes and the small offsets used, much of the structure of this hierarchy is easily visible. For example, one can easily see several categories, including "Music" and "Books", where the bulk of auctions are grouped in just a few categories that are very shallow within the hierarchy. In other portions of the hierarchy, such as "Collectibles" and "Business & Industrial", auctions are relatively evenly divided across a deep hierarchy. Although nodes in the treemap are much too small for labels, the visibility of the structure of the hierarchy gives appropriate insight into what will be found by zooming into a portion of the hierarchy.

As an informal experiment, we separately generated treemaps for the subtrees corresponding to the twenty largest top-level categories in this dataset. We labeled the top three levels in each treemap, and the treemaps were arranged to be shown in an area of size 1280 x 1024. We compared the number of missing nodes introduced by a standard layout algorithm versus our two-stage layout, as well as the correlation between node weight and node area in the resulting treemaps. Using a standard layout algorithm, we found an average of 91.7 ($\sigma = 124.4$) missing nodes (recall that missing nodes are not simply too small to be seen, rather they have been explicitly pushed out of the space allocated to them by offsets and space for labels not considered by the layout algorithm), representing 12.0% ($\sigma = 18.1\%$) of the nodes in the hierarchy and 0.8% ($\sigma = 0.7\%$) of the total weight of the hierarchy. As expected, our two-stage layout process did not create any missing nodes. While correcting the problem of missing nodes is an important contribution, examining the correlation between node weight and node area in the resulting treemap yielded more ambiguous results. Overall our two-stage layout process had no consistent positive or negative impact on node size distortion, but it did appear to have a consistently positive impact on the correlation for relatively small nodes in the hierarchy. This may be due to a ceiling effect if larger nodes already have good correlations that cannot be significantly improved, due to a relationship between label or offset size and the level in a treemap at which our two-stage process significantly affects node size, or because our combination of offsets, labeling depth, and layout size limited the improvement that could be obtained by explicitly considering the space dedicated to presenting the structure of the treemap. This suggests future work to carefully probe the different variables that affect a treemap layout to determine which lead to the greatest distortions and how those distortions can be corrected. Our current informal expectation, consistent with Figure 11, is that node size distortion is especially problematic for deep paths within relatively small hierarchies. One would not expect such deep paths to occur within the eBay hierarchy, which has been manually designed to support effective Web-based browsing and will therefore favor broad trees that allow people to quickly scan lists of auction categories.

Several avenues of future work are suggested by this paper. Our two-stage layout has addressed the problem of missing nodes, but we have noted that it would be interesting to further study what types of trees lead to the largest distortions of node size. One might then be able to modify our two-stage process to better address such situations, perhaps by modifying the standard squarified treemap layout algorithm that we currently use in our first layout phase. After such issues have been more completely

addressed, it would be appropriate to conduct formal experiments examining cascaded treemaps. Such experiments should build upon methods applied in previous work [17, 28, 32], but should also explicitly address stability and zooming in the interactive exploration of treemaps. We note however that it seems inappropriate to overly emphasize task performance measures, as several other aspects of treemaps have been important to their adoption, including their ease of implementation, their visual simplicity, and their ability to scale to large datasets [12]. It is therefore important that cascaded treemaps provide a natural depth effect and padding between siblings in complex hierarchies while preserving the aesthetic qualities, scalability, and ease of implementation commonly associated with treemaps.

In summary, this paper discusses four aspects of cascaded treemaps. We first discuss the cascaded presentation's ability to provide a depth effect and natural padding between siblings in complex hierarchies while using less space to illustrate the same containment relationship typically illustrated through nesting. We then discuss general concerns regarding treemaps, the distortion of node size, and the creation of missing nodes. We next discuss a related issue, support for stable treemap layouts in support of interactive zooming. Both concerns with existing treemap layout algorithms are addressed in our efficient two-stage layout process. We present all of this in the context of large real-world datasets, the Java package hierarchy and the eBay auction hierarchy.

ACKNOWLEDGEMENTS

We thank Shi Xia Liu for her early contributions to this work. We also thank anonymous reviewers and members of the Graphics Interface 2008 Program Committee for identifying key improvements to this paper. This work was supported in part by SRI CALO grant 03-000225.

REFERENCES

[1] Ahlberg, C. and Shneiderman, B. (1994). Visual Information Seeking: Tight Coupling of Dynamic Query Filters with Starfield Displays. *Proceedings of the ACM Conference on Human Factors in Computing Systems* (CHI 1994), 313-317.

[2] Balzer, M., Deussen, O. and Lewerentz, C. (2005). Voronoi Treemaps for the Visualization of Software Metrics. *Proceedings of the ACM Symposium on Software Visualization* (SoftVis 2005), 165-172.

[3] Bederson, B.B. (2001). PhotoMesa: A Zoomable Image Browser Using Quantum Treemaps and Bubblemaps. *Proceedings of the ACM Symposium on User Interface Software and Technology* (UIST 2001), 71-80.

[4] Bederson, B.B. and Hollan, J.D. (1994). Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics. *Proceedings of the ACM Symposium on User Interface Software and Technology* (UIST 1994), 17-26.

[5] Bederson, B.B., Shneiderman, B. and Wattenberg, M. (2001). Ordered and Quantum Treemaps: Making Effective use of 2D Space to Display Hierarchies. *ACM Transactions on Graphics* (TOG), **21**(4). 833-854.

[6] Bladh, T., Carr, D.A. and Scholl, J. (2004). Extending Tree-Maps to Three Dimensions: A Comparative Study. *Proceedings of the Asia Pacific Conference on Computer-Human Interaction* (APCHI 2004), 50-59.

[7] Blanch, R. and Lecolinet, E. (2007). Browsing Zoomable Treemaps: Structure-Aware Multi-Scale Navigation Techniques. *IEEE Transactions on Visualization and Computer Graphics* **14**(6). 1248-1253.

[8] Bruls, M., Huizing, K. and Van Wijk, J.J. (2000). Squarified Treemaps. *Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization* (TCVG 2000), 33-42.

[9] *del.icio.us Most Popular Treemap*. (2007). http://codecubed.com/map.html.

[10] Demian, P. and Fruchter, R. (2006). Finding and Understanding Reusable Designs from Large Hierarchical Repositories. *Information Visualization* **5**(1). 28-46.

[11] *eBay*. (2007). http://listings.ebay.com/.

[12] Fekete, J.-D. and Plaisant, C. (2002). Interactive Information Visualization of a Million Items. *Proceedings of the IEEE Symposium on Information Visualization* (InfoVis 2002), 117-124.

[13] Fiore, A. and Smith, M. (2001). Treemap Visualizations of Newsgroups. *Technical Report, Microsoft Research, Microsoft Corporation: Redmond, WA*

[14] Heer, J., Card, S.K. and Landay, J.A. (2005). prefuse: A Toolkit for Interactive Information Visualization. *Proceedings of the ACM Conference on Human Factors in Computing Systems* (CHI 2005), 421-430.

[15] *Java SE APIs & Documentation*. (2007). http://java.sun.com/javase/reference/api.jsp.

[16] Johnson, B. and Shneiderman, B. (1991). Tree-Maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures. *Proceedings of IEEE Conference on Visualization* (Vis 1991), 284-291.

[17] Kobsa, A. (2004). User Experiments with Tree Visualization Systems. *Proceedings of the IEEE Symposium on Information Visualization* (InfoVis 2004), 9-16.

[18] Lamping, J., Rao, R. and Pirolli, P. (1995). A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies. *Proceedings of the ACM Conference on Human Factors in Computing Systems* (CHI 1995), 401-408.

[19] Liu, S., Cao, N. and Lv, H. (2008). Interactive Visual Analysis of the NSF Funding Information. *Proceedings of the IEEE Pacific Visualization Symposium* (PacVis 2008), 151-158.

[20] Liu, S., Cao, N., Lv, H. and Su, H. (2006). The Visual Funding Navigator: Analysis of the NSF Funding Information. *Proceedings of the ACM Conference on Information and Knowledge Management* (CIKM 2006), 882-883.

[21] Orso, A., Jones, J. and Harrold, M.J. (2003). Visualization of Program-Execution Data for Deployed Software. *Proceedings of the ACM Symposium on Software Visualization* (SoftVis 2003), 67-76.

[22] Perlin, K. and Fox, D. (1993). Pad: An Alternative Approach to the Computer Interface. *Proceedings of the ACM Conference on Computer Graphics and Interactive Techniques* (SIGGRAPH 1993), 57-64.

[23] Robertson, G.G., Mackinlay, J.D. and Card, S.K. (1991). Cone Trees: Animated 3D Visualizations of Hierarchical Information. *Proceedings of the ACM Conference on Human Factors in Computing Systems* (CHI 1991), 189-194.

[24] Shneiderman, B. (1992). Tree Visualization with Tree-Maps: 2-D Space-Filling Approach. *ACM Transactions on Graphics* (TOG), **11**(1). 92-99.

[25] Shneiderman, B. and Wattenberg, M. (2001). Ordered Treemap Layouts. *Proceedings of the IEEE Symposium on Information Visualization* (InfoVis 2001), 73-78.

[26] Smith, M. *Netscan: Usenet Hierarchies - Treemap*. (2001). http://netscan.research.microsoft.com/treemap/.

[27] Smith, M. (2002). Tools for Navigating Large Social Cyberspaces. *Communications of the ACM* **45**(4). 51-55.

[28] Stasko, J., Catrambone, R., Guzdial, M. and McDonald, K. (2000). An Evaluation of Space-Filling Information Visualizations for Depicting Hierarchical Structures. *International Journal of Human-Computer Studies* (IJHCS), **53**(5). 663-694.

[29] Stasko, J. and Zhang, E. (2000). Focus+Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualizations. *Proceedings of the IEEE Symposium on Information Visualization* (InfoVis 2000), 57-65.

[30] Turo, D. and Johnson, B. (1992). Improving the Visualization of Hierarchies with Treemaps: Design Issues and Experimentation. *Proceedings of IEEE Visualization Conference* (Vis 1992), 124-131.

[31] University of Maryland. *Treemap: Home Page*. (2007). http://www.cs.umd.edu/hcil/treemap/.

[32] Van Ham, F. and Van Wijk, J.J. (2003). Beamtrees: Compact Visualization of Large Hierarchies. *Information Visualization* **2**(1). 31-39.

[33] Van Wijk, J.J. and Van de Wetering, H. (1999). Cushion Treemaps: Visualization of Hierarchical Information. *Proceedings of the IEEE Symposium on Information Visualization* (InfoVis 1999), 73-78.

[34] Vliegen, R., Van Wijk, J.J. and Van der Linden, E.-J. (2006). Visualizing Business Data with Generalized Treemaps. *IEEE Transactions on Visualization and Computer Graphics* **12**(5). 789-796.

[35] Wattenberg, M. *Map of the Market*. (1998). http://www.smartmoney.com/marketmap/.

[36] Wattenberg, M. (1999). Visualizing the Stock Market. *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems* (CHI 1999), 188-189.