Regular TreeMap Layouts for Visual Analysis of Hierarchical Data

Tobias Schreck Daniel Keim Florian Mansmann
Databases and Visualization Group
University of Konstanz, Germany
{*schreck,keim,mansmann*}*@inf.uni-konstanz.de*

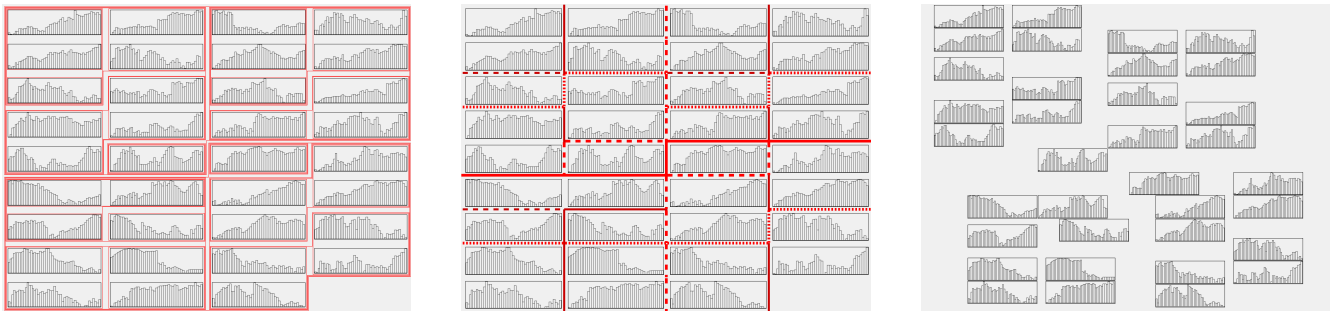


Figure 1: Three *Grid TreeMap* variants applied on a hierarchically structured data set of 35 time series objects.

Abstract

Hierarchical relationships play an utmost important role in many application domains. The appropriate visualization of hierarchically structured data sets can contribute towards supporting the data analyst in effectively analyzing hierarchic structures using visualization as a user friendly means to communicate information. Information Visualization has contributed a number of useful techniques for visualization of hierarchically structured data sets. Yet, the support for certain regularity requirements as arising from many data element types has to be improved. In this paper, we analyze an existing variant of the popular *TreeMap* family of hierarchical layout algorithms, and we introduce a novel TreeMap algorithm supporting space efficient layout of hierarchical data sets providing global regular layouts. We detail our algorithm, and we present applications on a real-world data set as well as experiments performed on a synthetic data set, showing its applicability and usefulness.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Display Algorithms; H.4 [Information Systems]: Information Interfaces and Presentation—General.

Keywords: Information Visualization; Hierarchic Data; Layout Generation; TreeMap Family of Layout Algorithms.

1 Introduction and Background

Growing volumes of information are produced by modern information systems, resulting in the need for appropriate tools for handling the *Information Overload* phenomenon [Thomas 2005]. Information Visualization aims at providing tools for visually supporting the processing of growing information loads by users, leveraging the strong human capabilities for visual information perception and processing [Keim 2002]. While information usually comes in a wealth of facets, one highly important data characteristic present in many application domains are hierarchical relationships among the elements of a data set. E.g., consider directory information, network structures, geo-related hierarchical relationships, or organizational structures, and so on.

Information Visualization to date has contributed with helpful ideas for displaying hierarchies. One of them is the Cone Tree technique [Robertson et al. 1993], a 3D visualization that orders child nodes on a circle below or next to their parent node. When the links between parent and child nodes are drawn, cone-like structures appear. Other research focused on using planar methods to display hierarchies, as any tree can be drawn in 2D without intersecting edges. One possibility to display a hierarchy is the Hyperbolic Tree [Lamping et al. 1995]. This visualization technique enables focus and context visualizations by taking advantage of a hyperbolic projection which scales nodes according to their proximity to the focal point. Another technique for visualizing hierarchical data sets is the so-called Interring [Yang et al. 2002] which displays nodes of a hierarchy by ring segments. The center of the Interring represents the root node of the hierarchy. All child nodes are arranged on concentric circles; the further they are away from the root node, the deeper their level within the hierarchy. For each node, all respective ancestor nodes can be found in between the ring segment representing the considered node, and the center of the Interring.

In this paper, we develop a novel visualization approach suited for the space-efficient layout of hierarchically structured data sets requiring a high *degree of regularity* in the display. Many important data types such as time series, images, or abstract and multi-dimensional data types require special consideration of regularity in the display in order to effectively compare many different elements along a given hierarchical structure imposed on the elements. Our work is inspired by the popular TreeMap [Shneiderman 1992] family of layout algorithms. One central advantage of TreeMaps is that they recursively subdivide the display in a space-filling manner, fully utilizing the available display space. Thereby, they produce space-efficient overviews over hierarchically structured data sets. Meanwhile, several applications of the TreeMap algorithm, such as *SequoiaView* [SequoiaView Homepage] based on [van Wijk and

van de Wetering 1999] or the *Map of the Market* based on [Wattenberg 1999], have found their way into practice. By design, the standard TreeMap algorithm does not consider display regularity, thus it is not directly suited for visualization of arbitrary data elements. Variants of the algorithm have been developed addressing the regularity problems present in the original approach (cf. Section 2). We here propose a TreeMap variant specifically addressing regularity requirements.

The remainder of this paper is structured as follows. In Section 2, we discuss the original TreeMap algorithm as well as several extensions related to our work. In Section 3, we then develop a family of new TreeMap algorithms based on analysis of the TreeMap variants discussed in Section 2. In Section 4, we apply three instantiations of our layout algorithm family as well as one related TreeMap variant on a hierarchically structured time series data set, demonstrating the usefulness of our techniques. In Section 5, we go on to formally evaluate the space efficiency characteristics of our algorithms by performing a range of layout experiments based on synthetic data sets modeling different classes of hierarchic structures. Finally, Section 6 concludes and outlines future work in the area.

2 Existing TreeMap Layout Algorithms

In this Section, we review the TreeMap family of layout algorithms based on which we will develop the *Grid TreeMap* layout scheme. In Section 2.1, we first recall the standard continuous TreeMap algorithm by Shneiderman. In Section 2.2, we then discuss the Quantum TreeMap algorithm.

2.1 Continuous TreeMaps

The standard TreeMap algorithm [Shneiderman 1992] is a simple yet powerful layout technique supporting hierarchically structured data. To date it has triggered many applications, extensions, and several commercial products based on the algorithm. While in principle the technique is applicable to any data type, usually it is applied for layout of categorical, scalar-valued, or low-dimensional data collections. As input the algorithm accepts a hierarchy of data elements as well as a root display area. The layout is generated by recursively traversing the hierarchy top-down, splitting the display rectangle alternatingly horizontally and vertically on the fly. The final display is space–filling, overlap–free, and communicates two salient data properties: It visualizes (a) *hierarchical* relationships of the data elements via the splitting and nesting structure of the display, and (b) *quantitative* information regarding the distribution of data elements by the size of respective display rectangles. The latter property is guaranteed by determining split points for splitting a given input rectangle in linear proportion to sums of weights obtained from counts or other quantitative measures underlying the data elements to be laid out. The basic TreeMap algorithm is illustrated in Figure 2. In the remainder of this paper, we will refer to the original TreeMap algorithm as the *Continuous TreeMap* (CTM), as it continuously places split points in linear proportion to sums of underlying weights.

Due to splitting of rectangles in linear proportion to sums of underlying weights, the continuous TreeMap may produce tessellations consisting of many different rectangle aspect ratios, depending on input data characteristics. Then, it may be hard for the user to select, compare, and trace rectangles throughout the hierarchy [Vernier and Nigay 2000; Bruls et al. 2000; Bederson et al. 2002]. Several TreeMap variants have been proposed modifying the splitting

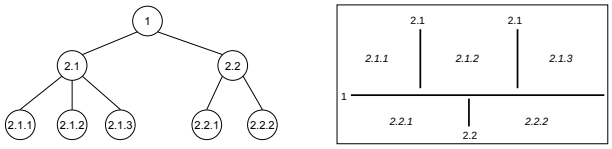


Figure 2: The standard Continuous TreeMap (CTM) algorithm alternatively splits display rectangles along horizontal and vertical lines while recursively traversing a hierarchically structured data set in top–down direction. In this illustration, a data set of 5 elements organized in 2 groups (left image) is laid out by using the number of leaf elements as the weights for split point determination (right image). While the display is space–filling and overlap–free, display regularity is low due to differing rectangle aspect ratios, and the non-alignment of rectangles resulting from continuous splitting.

process in order to produce more uniform rectangle aspect ratios. Optimizing rectangle aspect ratios incurs certain additional costs as compared to the original TreeMap algorithm, such as increased *computing complexity* due to optimization steps [Wattenberg 1999; Vernier and Nigay 2000], *reduced ordering* by switching of layout directions [Bruls et al. 2000], or reduced size to weight *proportionality* [Dayal et al. 2005].

2.2 Quantum TreeMaps

It has been recognized that besides *optimizing* aspect ratios, in many applications it is desirable to *guarantee* constant size and aspect ratios for all of the rectangles to be laid out. This is motivated by supporting visual comparability when displaying multi-dimensional or abstract data types such as images [Bederson et al. 2002] or time series data [Dayal et al. 2005], which call for regularity-providing layout generation algorithms. Consider for example the problem of laying out sets of hierarchically structured time series, or more generally, bar chart data. Then, in order to be able to compare time intervals and value magnitudes, there must not be too many different scales present in the display tessellation. Figure 3 shows an example of visualizing a set of bar charts using the CTM algorithm. Clearly, it is hard to compare periods in time and value magnitudes throughout the display, as practically every time series rectangle is assigned unique scales for its x and y axes.

In order to support regularity in TreeMap displays, in [Bederson et al. 2002] it was proposed to perform a quantization of the output of the TreeMap algorithm, where width and height of the resulting rectangles are allowed to assume only integer multiples of predefined height and width quanta. The so-called *Quantum TreeMap* (QTM) guarantees consistent rectangle aspect ratios, and by design places all data elements on unique positions on a global regular grid. The basic idea is to first perform continuous splitting, and then, based on this (intermediate) result, search for a good quantization of the split points allowing to lay out the number of data elements requested by the given data subset. It has to be defined what constitutes a good quantization, but usually, the amount of wasted space, or the deviation from the continuous rectangle in terms of aspect ratio or symmetric area difference are candidates for optimization. Figure 4 illustrates the QTM technique.

By design, QTM does not obey the space-filling property of the original algorithm. In [Bederson et al. 2002], the authors performed an experimental analysis of the *display overhead* metric, which is the amount of display space not occupied by data elements. The authors concluded that display overhead is not critical when laying out data sets consisting of many elements per hierarchical group.

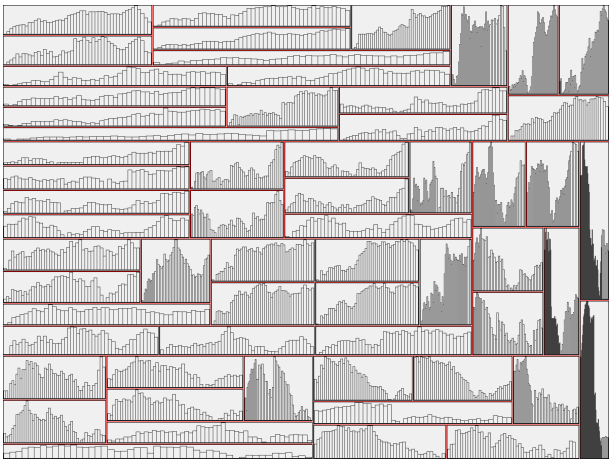


Figure 3: In practice, the CTM algorithm produces many different aspect ratios and *x*/*y*-scales in its layout. Thereby, it is difficult to present important data types such as time series inside the obtained rectangles in a useful way. The image shows the CTM layout of the data set used in Section 4. Clearly, it is hard to effectively compare the data set elements.

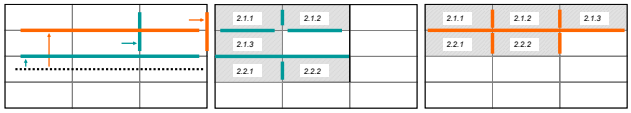


Figure 4: The Quantum TreeMap (QTM) quantizes the results of continuous splitting to a suitable amount of rows and columns on a global regular grid. Left is shown the global grid and the first split line from the example in Figure 2 (the dashed line). Based on the adopted searching heuristic, different allocation outcomes are possible (middle and right images).

In this paper, we will revisit the QTM algorithm in an experimental analysis, and will compare it against our *Grid TreeMap* algorithm developed in the next Section.

3 Grid TreeMap Algorithm

The Quantum TreeMap provides layouts with guaranteed constant size and aspect ratios of the data rectangles, and with consistent alignment of the elements on a global grid. There exist two sources *O* for potential display overhead (loss in display utilization efficiency):

O_1 QTM is not always able to quantize layout partitions to grid dimensions corresponding to the exact number of elements to be laid out (grid cells may be left unoccupied; first source of potential display overhead).

O_2 QTM performs on-the-fly quantization of split points. During processing of the algorithm, the resulting global layout may grow or shrink in width and height, deviating from the initial root display rectangle. The final result may have to be be (isotropically) scaled back into the original root display rectangle. Whenever the aspect ratios of the root display and the resulting grid differ, then display overhead due to scaling will occur (second source of potential display overhead).

We presume that based on the hierarchic characteristics of the input data set, significant display overhead may occur in QTM. We therefore researched an alternative quantization scheme we call the *Grid TreeMap* (GTM). Instead of first performing the continuous TreeMap on the root display rectangle and then quantizing the result to the grid, we go the other way round. We first decompose the root display rectangle to a grid of sufficient dimensionality, and then perform the TreeMap algorithm directly on the resulting grid. The grid dimensionality is found such that (a) the number of rows and columns is sufficient to hold all the data elements, (b) no more than one row or one column is only partially occupied with data elements, and (c) the aspect ratio of the resulting grid slots matches a predefined (targeted) aspect ratio as closely as possible. We then perform the TreeMap algorithm on this grid by alternatingly scanning rows or columns to assign data elements to slots on the grid (cf. Figure 5). We presume this approach to be more space efficient than QTM for certain data set characteristics, at the same time producing regular layouts. We recognize that by using scan based assignment of elements to grid slots, we will encounter "stair-step" effects, thus split boundaries (hierarchical separators) are not guaranteed to be straight lines anymore. It will be the responsibility of the following rendering techniques to compensate for the loss of the straight line property, which by design is provided in the CTM and QTM methods.

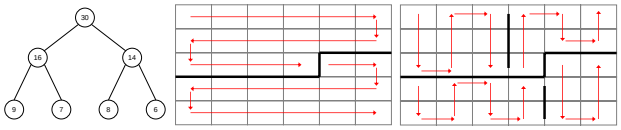


Figure 5: With the *Grid TreeMap* (GTM), first the display rectangle is decomposed into a grid of sufficient dimensionality to hold all data set elements. Then, on this grid GTM performs slicing and dicing by alternatingly scanning rows and columns. The right two images show the GTM algorithm allocating the first and the second hierarchical partitions of the hierarchy denoted on the left.

3.1 Rendering Grid TreeMaps

One property of the Quantum TreeMap not present in the Grid TreeMap is that in the former, groups of elements are separated by straight lines. With QTM, the TreeMap user can trace connections of the straight lines to quickly recognize hierarchical relationships present in the data. With GTM, which performs scan line based partitioning of grid slots, we do not have the straight line separator property, and it is expected that this negatively affects the ready perceivability of hierarchic relationships without providing appropriate visual support. Therefore, we have to pay close attention to the way by which we indicate the hierarchy separators. We have experimented with several different separating schemes, and found three of them effective for rendering GTM layouts of different data hierarchy patterns:

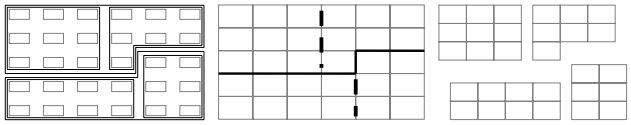
1. *GTM-N* (Nested): Draw enclosing boundaries around groups of elements. Free the required display space by simultaneously scaling down all grid rectangles such that the display can accommodate all boundaries without over plotting.

2. *GTM-S* (Split Lines): Keep the distance between adjacent grid slots constant, but indicate split line depth (hierarchy level) by appropriately setting the drawing attributes color, thickness, and shape of the respective split lines.

3. *GTM-B* (Burst): Indicate hierarchy by varying the distance between groups of elements with the splitting depth. Let

greater distance between groups of rectangles indicates higher (closer to the root) hierarchic separations.

GTM-Split keeps the amount of display space dedicated to hierarchic information constant, but is not expected to scale for arbitrary deep hierarchies due to the fixed amount of display space reserved for the split lines. GTM-Nest and GTM-Burst do not guarantee a bound for the fraction of display space allocated for indicating arbitrarily deep or wide hierarchical relationships, but we will see in the next Sections that the techniques work well in practice. The proposed rendering techniques are illustrated in Figure 6. In the next sections, we will apply the GTM variants on a real-world data set, and perform experiments on synthetic data assessing the quality of the layout algorithms.



(a) Nest (b) Split Lines (c) Burst

Figure 6: Three rendering methods for visualizing hierarchical relationships for Grid TreeMap layouts. Cf. also Figure 5.

4 Application

We here present the application of the QTM and GTM algorithms on a data set consisting of daily stock price time series obtained from [S&P500 stock price archive]. We like to render the data using the familiar bar chart drawing technique, which requires a high degree of regularity in the display to support effective comparative analysis. To allow the analyst to gain a quick overview over a set of bar charts, the corresponding layouts should produce regular tessellations. Note that a continuous layout as provided by the classical CTM (c.f. Figure 3) is clearly inappropriate for this task, and we recognize the need for quantized layouts.

In order to impose a meaningful hierarchical ordering on the data set, as a preprocessing step we apply the *Hierarchical Agglomerative Clustering Algorithm* (HAC) [Han and Kamber 2001] on the data set. The HAC iteratively merges pairs of sets of elements which exhibit highest similarity at the given iteration. The result is a binary tree which when properly visualized is a nice tool for analyzing similarity relationships in a set of objects on which a meaningful similarity scale can be defined. Note that binary-tree hierarchies are a stress test for the layout algorithms. The height h of a balanced tree with fanout f containing n leafs (data elements) logarithmically depends on f, as $h = log_f n$ holds. This in turn means that binary trees give deepest hierarchies (more hierarchy levels) as compared to trees with fanout larger than 2. Thereby, the display has to communicate more information and consequentially, has to allocate more display space resources for hierarchy visualization.

For generating the layouts, we used a subset of the time series database consisting of 60 series with 48 values each. We configured the HAC algorithm to use the Euclidean distance between time series normalized for offset and scale. Such normalization is a useful preprocessing step when calculating time series similarity [Keogh 2004]. The resulting tree is of height 11, which leads to a high degree of data partitioning, stressing the layout algorithms. We set the targeted aspect ratio (as width : height) to 4 : 1, which seems reasonable for rendering bar charts of the considered length. We set the root display to 1200×900 pixels. Where needed, we set indention of grid rectangles and thickness of splitting / nesting lines to reasonable parameters which try to be space efficient, and at the same time support perception of hierarchical separation relationships.

In the following, we consider as *display overhead* the fraction of the root display rectangle which is not occupied by time series (bar chart) rectangles. Specifically, empty space as well as space occupied by split lines or used for element separation contributes to display overhead. In case of QTM and GTM-B which may return layouts violating the aspect ratio of the root display rectangle , we isotropically scale back the layout to fit the root display rectangle prior to measuring display overhead.

4.1 Quantum TreeMap

Figure 7 (a) gives the result of the QTM layout of the hierarchically clustered data set. The display aligns all time series rectangles on a global grid, and guarantees that all rectangles have the same size and aspect ratio. Due to the binary branching in the HAC tree, the tree height is significant and results in a complex set of hierarchical relationships among the time series. This in turn stresses the layout in terms of display overhead. When allocating inner tree nodes, QTM searches for grid layouts which locally minimize display overhead. Therefore, the total display overhead of a given data partition not only depends on the current quantization decision made for the given tree node. It also depends on the quantization steps performed when subsequently laying out the subtrees rooted at the given node, as well as the layout of the siblings of the considered tree node. By design, QTM is a greedy layout algorithm which does not consider global effects in the local layout decisions it makes, and consequently, the display overhead is expected to be high if the tree structure is complex and deep. The overall overhead in the QTM display of this data set amounts to 71%, which is significant. On the other hand, the QTM retains the straight line property when separating groups of elements in the display. This makes it rather easy for the analyst to trace partition borders in order to understand the hierarchical relationships in the data set.

A more subtle problem coming along with the uneven distribution of elements to embedding rectangles is that of potentially reduced perceivability of *balancing relationships*. Note that in the QTM display, the size of rectangular display partitions does not have to be proportional to the number of contained data elements. E.g., consider in Figure 7 (a) the first top-level partition of the data set indicated along the straight horizontal line at about $\frac{1}{3}$ height. Within this partition, the data is subsequently separated into two groups containing 15 and 3 elements, respectively (the left and the right partition in the topmost display partition). As a result of the final quantization layout, the second-level (vertical line) separator between the two groups is placed at $\frac{1}{2}$ display width. At first, this might mislead to assuming both partitions to be balanced, as the display areas in the left and right hand side partitions are equal. Yet, the population of these areas with data elements is quite uneven (15 and 3 elements, respectively).

4.2 Nested Grid TreeMap (GTM-N)

Figure 7 (right) gives the result of the GTM-N (nested) layout of the data set. The algorithm first generates the Grid TreeMap layout, and then draws thin boundaries around groups of elements. Here, we drew boundaries of 1 pixel width around groups of elements which belong together along the hierarchy. We also indent adjacent lines by 1 pixel. The deeper a given hierarchy, the more grouping lines have to be drawn. Consequently, the bar chart boxes have to

(a) Quantum TreeMap (QTM)

(b) Nested Grid TreeMap (GTM-N)

Figure 7: Quantum TreeMap (a) and nested Grid TreeMap (b) layouts of 60 time series, organized by similarity using the hierarchic agglomerative clustering algorithm (HAC) for preprocessing.

be scaled down to free up the space required for border drawing. We chose to simultaneously scale down all rectangles by the same amount as required by the hierarchy, to keep the data elements at the same size and consistently positioned on the global grid.

The display overhead metric amounts to roughly 70%, which is comparable to the QTM result. The reason is the depth of the hierarchy. As the HAC tree is of height 11, and considering we have to allocate at least 2 pixel per hierarchic boundary on each side of the respective rectangles, it is not surprising that the rectangles are scaled down significantly. On the other hand, as compared to QTM most of the space not occupied with bar charts is occupied by nesting boundaries, indicating hierarchic information. We note that the topmost separation levels are clearly perceivable, as there is a visual cumulative effect when many lines are drawn in parallel. Also, the number of elements contained in subtree partitions is better perceivable, as there is no "dead space" like in the QTM. It is somewhat more difficult to trace the lower separation borders. Eventually, we have to rely on the pixel-level to completely assess the full hierarchical structure.

4.3 Hierarchic Split Lines Grid TreeMap (GTM-S)

The Grid TreeMap with hierarchic split lines (GTM-S) uses visual attributes of split lines of fixed maximal width in order to communicate hierarchical separations throughout the data tree. Designing split lines which are visually discriminating and at the same time are capable to encode ordinal relationships (tree levels) is not an easy task. We experimented with several different settings, and found the scheme given in Figure 9 a good compromise, although other schemes are possible. We employ color, split line width, and dashing as visual attributes. Figure 8 (a) shows the resulting Grid TreeMap layout using hierarchic split lines. We have indented the rectangles by 10 Pixels each to free space for drawing the hierarchical split lines. We notice that the top separation levels in the hierarchy (i.e., the bright most split lines) are best perceived. Tracing the lower-level split lines is somewhat harder, as the lines get increasingly thinner, but still it is possible to do so using the legend. The overall display overhead of 31% is reasonably smaller than the one resulting from QTM and GTM-N.

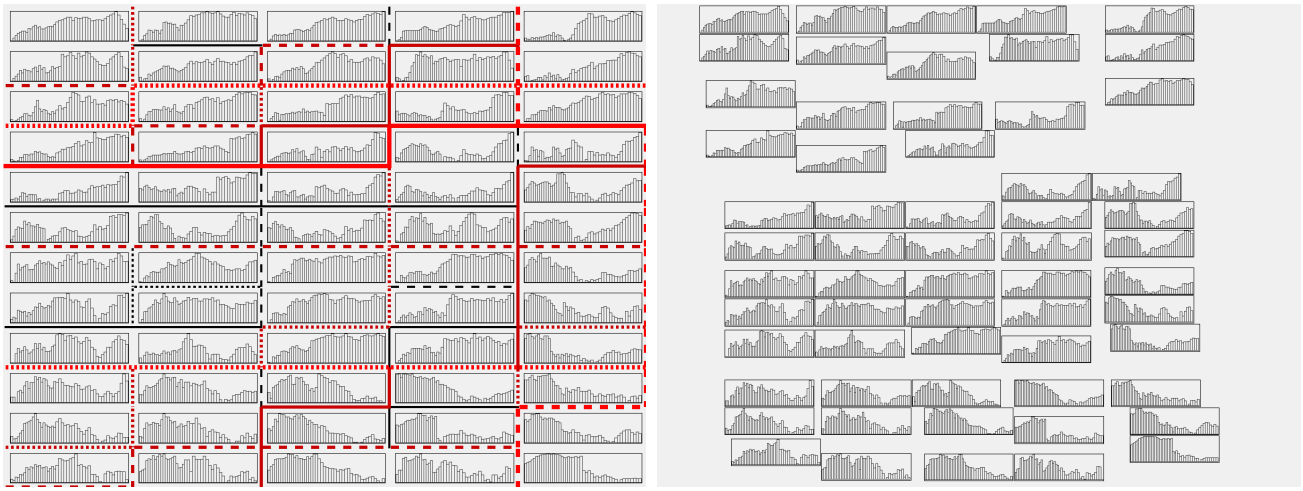
4.4 Burst Grid TreeMap (GTM-B)

GTM-N and GTM-S draw enclosing boundaries and hierarchic split lines to communicate hierarchic relationships among subsets of data elements. Another alternative we tested is to visualize the separation relationships by depth-dependent horizontal and vertical spacing between the individual data element groups. In our strategy, groups of objects get "burst apart" such that groups separated on higher hierarchy levels get separated by more horizontal or vertical space between them. We implemented a policy which during the layout generation performs the bursting by translating rectangle partitions: In a sequence of scanning rows (columns), the partitions get burst apart along vertical (horizontal) directions. We model the "bursting distance" $d(l)$ separating the sibling element groups rooted at an inner tree node of level l in a tree of height h as linearly depending on the respective hierarchy level l:

$$d(l) = \frac{h-l}{h} \times D^{max}$$

It results that the topmost partitions (rooted at hierarchy level $l = 0$) get allocated the distance $d(l = 0) = 1 \times D^{max}$ for separation, while the consecutive split levels are allocated a linearly decreasing amount of maximal bursting distance. In practice, we get good results by setting $D^{max} = min(width, height)$ of the grid cells.

Figure 8 (b) shows the GTM-B layout of the clustered data set. Translating the groups proportional to the splitting level supports perceivability of the splitting hierarchies. There is a tradeoff between the space-efficiency and the usage of bursting: Larger burst distances (implemented by higher $D^{max} settings$) consume more display space leading to higher display overhead rates for a given data set. Still, display utilization is reasonable. In this example, the space used for bursting was 53% of display space. We note that other than in the previous variants, the individual rectangles are not aligned on a global grid anymore due to the continuous calculation of the amount of translation. Note also that the resulting layout may need to be scaled back into the original root display area (like QTM), possibly resulting in scaling overhead O_2 (c.f. Section 3).



(a) Split Line Grid TreeMap (GTM-S) (b) Burst Grid TreeMap (GTM-B)

Figure 8: Splitline-based (a) and Burst Grid TreeMap (b) layouts of the same data as in Figure 7.

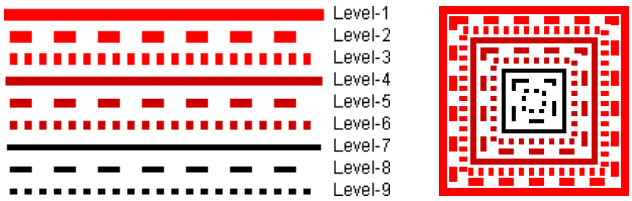


Figure 9: Split line legend for the GTM-S layout in Figure 8 (a).

5 Evaluation of Space Efficiency

We recognize two important criteria for evaluating the quality of the proposed layout algorithms. One criteria is the *effective perceivability of hierarchical relationships* by the user. Evaluating this metric formally is difficult as it would require a carefully designed user study which at present is beyond our resources. We note that we regard the discussion in Section 4 as supporting the effectiveness of the GTM techniques in visualizing hierarchical relationships in regular displays. A second obvious evaluation criterion is the *display utilization* which contrary to the hierarchy perceivability can be automatically evaluated using the display overhead metric. We experimentally measured the display overhead during batch experiments to layout synthetically generated data sets of different size and hierarchy characteristics. Specifically, we considered different balanced and unbalanced hierarchy models for laying out 5 up to 200 time series objects inside a 1200×900 root display area.

5.1 Balanced Hierarchies

We first evaluate the display overhead modeling the hierarchy as a *balanced* tree with constant fanout f at each inner tree node. Figure 10 gives the display overhead results for fanout settings $f = 2$ (left chart) and $f = 8$ (right chart), respectively.

GTM-Nest and GTM-Split clearly show decreasing space efficiency for growing numbers of elements. As the tree height grows with the number of elements, GTM-Nest has to allocate more space for boundary drawing, which is freed by scaling down the grid cells. In case of the binary tree ($f = 2$), already at 100 elements, space required for boundaries consumes more than 60% of display resources. Practically, rendering more that 100 elements in the given tree structure and root display (1200×900) using GTM-N thereby is problematic. Again, as noted in the preceding Section, using binary trees is a stress test for the Grid TreeMap. In case of trees of higher fanout this problem diminishes, as then tree height grows less aggressive. In case of $f = 8$, GTM-N's space efficiency improves to at most 50% of display overhead for 200 elements.

GTM-Split allocates constant inter-cell distances irrespective of the hierarchy structure (note the identical display overhead curves in Figures 10 and 11). As the number of cells grows, so does the amount of inter-cell spacing required for drawing attributed split lines. The amount of space allocated for hierarchy visualization amounts to 20% for few elements up to a maximum of 60% for 200 elements.

GTM-Burst indicates hierarchical separations by translating partitions of elements apart along opposite directions. The amount of translation negatively depends of the hierarchy level. Therefore, trees with high fanout at higher tree levels (close to the root) consume the most space for bursting. For $f = 2$ we observe reasonable display overhead numbers around 50% for most data sizes. With $f = 8$ the amount of space required is rather high (up to 80%) for most data sizes.

The Quantum TreeMap results show an oscillating display usage pattern. In our implementation, we let QTM search for a good local quantization by either extending the number of columns *or* the number of rows to accommodate the number of elements to lay out in each iteration, and using the *area* of the alternatives as the selection criteria. In the hierarchy model considered here, a large part of the display overhead in QTM is due to scaling overhead (overhead source O_2 in Section 3). In our experiments, many resulting QTM layouts are larger in width than in height, so isotropically scaling back the result into the 1200×900 display sacrifices vertical space. Due to the balanced structure of the hierarchy, overhead source O_1 makes up only a smaller part of the loss in display space. The latter finding is in accordance with the results from [Bederson et al. 2002]. With the data considered here, QTM overhead oscillates around 60% (50%) at $f = 2$ ($f = 8$), with significant variance.

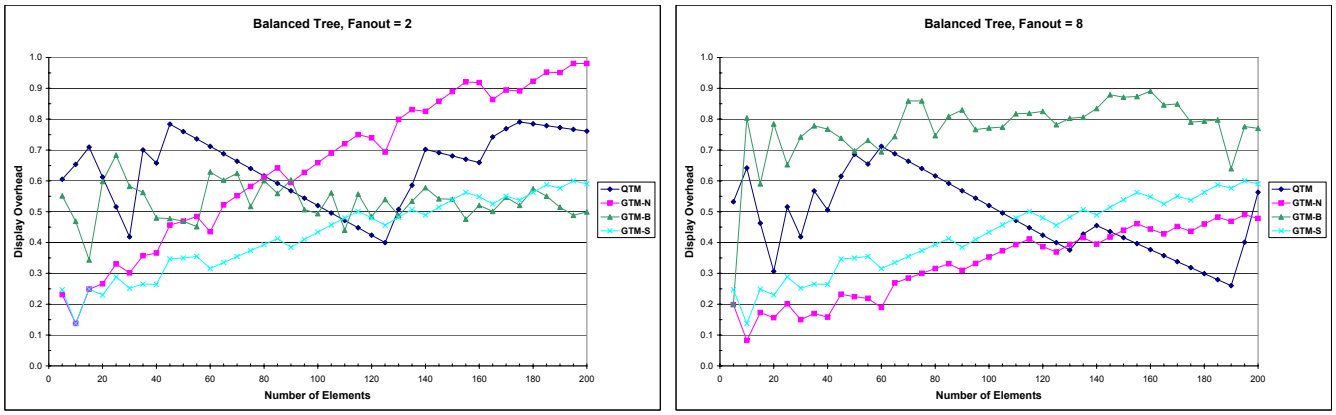


Figure 10: Display overhead for GTM-{N,B,S} and QTM, for balanced hierarchies with tree fanouts of 2 and 8, respectively.

5.2 Unbalanced Hierarchies

We also evaluated the display overhead when modeling the hierarchy as an *unbalanced* tree. We keep fanout f at each inner tree node constant but populate the leftmost child of each inner tree node with the triple number of leaf elements rooted at the respective node, as compared to its right hand side siblings. Figure 10 gives the results for fanout settings $f = 3$ (left chart) and $f = 4$ (right chart), respectively, for this tree model. We chose these fanout settings to get tree structures of sufficient depth for our experiments. We obtain unbalanced trees where data is unevenly distributed along the tree paths.

We observe the same metric readings for GTM-Split, as by design its space occupancy is invariant with respect to tree structure. GTM-Nest, due to the unbalanced hierarchy structure, has to dedicate less display space to the group boundaries as compared to the low-fanout tree results in Figure 10 (left). We observe maximal hierarchy overhead of about 80% (70%) for 200 elements using a fanout of $f = 3$ ($f = 4$). We note that for element counts up to about 140, in both hierarchies we still have around $100\% - 60\% = 40\%$ space left for element rendering. GTM-Burst is significantly impacted by the deeper hierarchies resulting from the unbalanced tree structure, as it has to bust apart more element partitions, and $d(l)$ is diminishing at a slower rate (cf. Section 4.4). Also, O_2 overhead occurs for GTM-Burst. Maximum overhead is in the range of 80% for both fanout settings, and for most data sizes. An exception is the interval between 40 and 100 elements with $f = 4$, where overhead is around 60%.

5.3 Experiments Summary

We can summarize the experimental findings as follows. GTM-Burst delivers efficient layouts in terms of the hierarchy overhead metric for hierarchical structures of comparably low fanout, as then, less space is allocated for bursting apart partitions (cf. Section 4.4). GTM-Nest delivers efficient layouts in the opposite case, namely, when fanout is high and resulting tree depth is comparably low. The limiting factor in GTM-Nest is the tree depth, as the maximal hierarchic level in the data set dictates the amount of down scaling that has to be applied on all grid cell rectangles (recall that we target regular layouts, so we demand uniform scales for all rectangles). GTM-Burst and GTM-Nest therefore are recommended for complementary use depending on the hierarchical structure to be visualized. GTM-Split provides a middle ground and is promising in case the number of elements in the data set is not too large, irrespective of the hierarchical structure present, up to a limit dictated by the number of levels we can visually discriminate using split line drawing attributes. We estimate the latter limit to be around 10 levels, which should be sufficient to support many real-world applications.

We also evaluated the Quantum TreeMap algorithm as a base line competitor for our algorithms. As the charts show, there are data constellations where either one of the Grid TreeMap algorithms outperforms the QTM, in term of the display overhead metric. We conclude that Grid TreeMaps are a practical option for visualizing regularity-requiring, hierarchical data sets. Depending on the nature of the data (hierarchical structure, data set size), the most appropriate GTM variant may be selected interactively by the user or automatically by the visualization system as based on the display overhead metric.

Regarding the rather high degree of display overhead in QTM, we note that the numbers may be due to our chosen implementation (cf. Section 5.1). Other quantization heuristics are possible, and we have not attempted to optimize the searching strategy to the considered data set characteristics. We note that traditionally, TreeMap algorithms perform greedy optimization, and this is rather a feature than a drawback of the techniques. The TreeMap philosophy is to provide fast algorithms suited for online layout generation. We therefore have not attempted to further optimize the quantization in our QTM implementation in a back-tracking manner, as we feel this would not be in accordance with the tradition of TreeMap algorithms. We rather take QTM as a reference for assessing the performance of our Grid TreeMap variants, which are designed to give compact representations based on the linear nature of the TreeMap algorithm family.

6 Conclusions

We have presented novel layout techniques based on the idea of applying the recursive TreeMap algorithm on a regular grid of display cells. The introduced Grid TreeMap layouts provide a high degree of regularity, and thereby are suited for layout of hierarchically structured, complex data sets requiring regularity such as time series data, images, and multidimensional data, among others. Depending on the nature of the data set in terms of hierarchy fanout, degree of tree balance, and data set size, different Grid TreeMap rendering variants using boundaries, split lines, or translation-based separation, can be recommended. We applied the techniques for

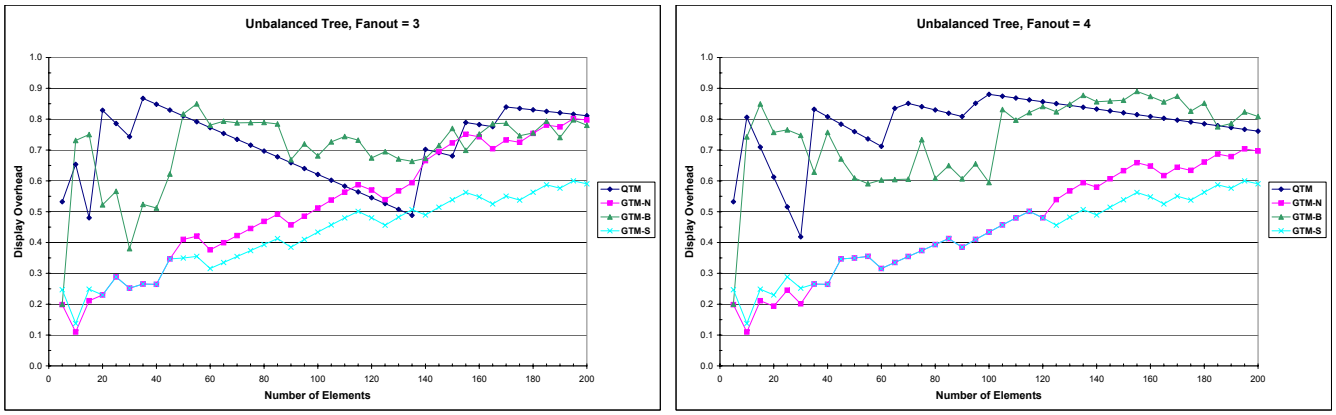


Figure 11: Display overhead for GTM-{N,B,S} and QTM, for unbalanced hierarchies with tree fanouts of 3 and 4, respectively.

visualizing hierarchically clustered time series data, and we performed synthetic experiments allowing to evaluate the algorithm performance in terms of perceivability of hierarchical relationships and display overhead.

In this paper we have focussed on designing the basic layout generators, and will address improved rendering for the different Grid TreeMap variants in the future. Several options for visually supporting the perceivability of hierarchical structures using cell connectors, indication of bursting directions, usage of shading and cushions and so on are possible. Also, interaction techniques such as mouse-over-based highlighting functionality like implemented in the SequoiaView system [SequoiaView Homepage] seem a promising extension for supporting hierarchical perception and navigation by the user. Evaluation of the effectiveness of hierarchical structures perception should be addressed by a formal user study. In this paper we assumed all the data elements to require the same layout (size and aspect ratio) for their enclosing element rectangles. Another interesting problem is to design regularity-providing layout generators for data where the elements require differing element layouts.

Acknowledgments

We thank Jack van Wijk for valuable discussion on regular layout generation. We thank our colleagues Christian Panse, Joern Schneidewind, Mike Sips, and Benjamin Bustos for ongoing discussion on layout techniques in general. This work was partially funded by the German Research Foundation (DFG) under Project No. KE 740/8-1, and under grant GK-1042 'Explorative Analysis and Visualization of Large Information Spaces'.

References

BEDERSON, B., SHNEIDERMAN, B., AND WATTENBERG, M. 2002. Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies. *ACM Transactions on Graphics 21*, 4, 833–854.

BRULS, M., HUIZING, K., AND VAN WIJK, J. 2000. Squarified treemaps. In *Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization (VisSym)*, 33–42.

DAYAL, U., HAO, M., KEIM, D., AND SCHRECK, T. 2005. Importance driven visualization layouts for large time-series data. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis)*, IEEE Computer Society, 203–210.

HAN, J., AND KAMBER, M. 2001. *Data Mining: Concepts and Techniques*. Morgan Kauffman.

KEIM, D. 2002. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics 8*, 1, 1–8.

KEOGH, E. 2004. Tutorial on data mining and machine learning in time series databases. In *Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM)*.

LAMPING, J., RAO, R., AND PIROLLI, P. 1995. A focus+context technique based on hyperbolic geometry for viewing large hierarchies. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, ACM, 401–408.

ROBERTSON, G., MACKINLAY, J., AND CARD, S. 1993. Information visualiyation using 3D interactive animation. *Communications of the ACM 36*, 4, 57–71.

SEQUOIAVIEW HOMEPAGE. http://www.win.tue.nl/sequoiaview/.

SHNEIDERMAN, B. 1992. Tree visualization with tree-maps: 2D space-filling approach. *ACM Transactions on Graphics 11*, 1, 92–99.

S&P500 STOCK PRICE ARCHIVE. http://kumo.swcp.com/stocks/.

THOMAS, J. 2005. Visual analytics: a grand challenge in science - turning information overload into the opportunity of the decade. In *Proceedings IEEE Symposium on Information Visualization (InfoVis)*. Keynote address.

VAN WIJK, J., AND VAN DE WETERING, H. 1999. Cushion treemaps. In *Proceedings IEEE Symposium on Information Visualization (InfoVis)*, 73–78.

VERNIER, F., AND NIGAY, L. 2000. Modifiable treemaps containing variable shaped units. In *Work in Progress Proceedings of the IEEE Visualization*.

WATTENBERG, M. 1999. Visualizing the stock market. In *Extended Abstracts on Human Factors in Computing Systems (CHI)*, ACM Press, 188–189.

YANG, J., WARD, M., AND RUNDENSTEINER, E. 2002. Interring: An interactive tool for visually navigating and manipulating hierarchical structures. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis)*.