Eurographics/ IEEE-VGTC Symposium on Visualization (2006)
Thomas Ertl, Ken Joy, and Beatriz Santos (Editors)

Combining Extended Table Lens and Treemap Techniques for Visualizing Tabular Data

Alexandru Telea

Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, the Netherlands
alext@win.tue.nl

Abstract

We present a framework for visualizing large tabular data that combines two views: the table view and the treemap view. The table view extends the known table lens as follows: We cluster related elements to reduce subsampling artifacts and achieve table size independent rendering time; we use multiple-column sorting to create scenario-specific data hierarchies on the fly; and we use shaded cushions to show data structure and variation. Hierarchies built in the table view are shown in a customizable treemap view. One can choose both layout and rendering by a few clicks, effectively creating visual scenarios on-the-fly. We illustrate our framework on real-life stock data.

Categories and Subject Descriptors (according to ACM CCS): H.2.8 [Database applications]: I.3.3 [Picture generation]:

1. Introduction

Tabular data is one of the most used information organization forms nowadays, present in many domains, e.g. finance, engineering, statistics, and medicine, and is at the core of many software tools ranging from end-user systems such as Excel and SPSS to full databases such as SQL and Oracle. A data table is usually a regular matrix-like structure of rows and columns containing data cells. Usually, cells of a single column have the same meaning and data type. Cells on a row are attributes of the same object, or tuple. Common tasks performed on table data involve assessing the properties of some attribute (column), finding relations between several attributes, of subranges thereof, e.g. data trends, averages, and outliers, and spotting correlations or other interesting similarities between attribute subranges [Tuk77, PR96].

Performing the above exploratory tasks visually using graphical table depictions is a powerful, quick, and easy-to-learn way to get insight in such datasets. The most common depiction is the spreadsheet metaphor, which draws textual cell value representations, but can show only a few tens of rows on a screen at a given time. The table lens technique [RC94, PR96, TR97] displays more rows by drawing every table row as a pixel row colored and scaled to convey cell values. Both techniques preserve the familiar tabular layout. However, if the data are structured along some hierarchy, the tabular layout is not the best, as it cannot reflect the hierarchy. Treemaps may be an effective alternative [Shn92], as they show both hierarchy and attribute values and support exploratory tasks which explicitly consider the hierarchy and/or its relationships with the attributes. However, how to combine the strengths of treemaps and tables?

In this paper, we present TableVision, a framework for visual analysis of tabular data, that tightly integrates two views: an enhanced table view and a treemap view. The table view extends the table lens technique in several ways. First, we use multiple column sorting to discover structure in the data, by creating hierarchies of related rows in a few mouse clicks. Second, we use shaded cushions to show the user-constructed hierarchy directly on the table. In our table view, we use the same visual metaphors for various data types, e.g. numbers, text, categories, time, and date. Third, we use automatic clustering to alleviate subsampling problems and render arbitrary size tables at interactive rates. Our second view is a treemap view which displays the hierarchy built in the table view. Users can choose the layout and renderer for every treemap level, thus creating flexible visualizations on the fly. We demonstrate TableVision with several scenarios using stock market data.

We proceed as follows. Section 2 overviews related work. Section 3 describes TableVision: the table view and our ad-

ditions to the standard table lens (Sec. 3.1) and how we visualize the hierarchies built in the table view using the custmoizable treemap view (Sec. 3.2). Section 4 discusses our results. Section 5 concludes the paper and outlines future work directions.

2. Related Work

Related work can be structured in the categories of table and treemap visualization. Rao *et al.* have introduced the table lens, a technique that combines textual (focal) and colored bar-graph (contextual) information in a table layout [RC94, PR96] to visualize hundreds of rows on a single screen. The technique was extended to use multiple text focus points [TR97] and also commercially implemented [Inx05]. However, the original design [RC94, PR96] may have performance and subsampling problems when rendering tens of thousands of rows. Moreover, since a column is basically drawn as a narrow 2D graph, it is often hard to spot small-scale variations. Treemaps display both hierarchical structure (encoded as layout) and data value distribution (encoded as size and/or color) in a screen-compact way [Shn92]. Various treemap layouts exist, e.g. squarified [BHvW00], space-filling [BE95], and strip [BSW02]. However, treemaps usually visualize a predefined, existing hierarchy, e.. the 'map of the market' [Wat99,Mar05] or hard disk structure [vWvdW99]. In our case, both visualizing *and* constructing the hierarchy are intrinsic, coupled parts of the interactive exploratory process.

3. TableVision Framework

Our TableVision framework consists of multiple, tightly correlated, table and treemap views. Both views can be used independently. One scenario is to start with the more familiar table view (Sec. 3.1) to build data hierarchies and visualize these in the treemap view (Sec. 3.2). Both views access tables from an SQL database, which allows easily computing various statistics and derived data from the primary values. As examples, we use stock datasets containing tuples (rows) of industry categories, company names, and open, high, low courses of their shares taken at various dates and day-times. Each attribute is a separate table column.

3.1. Enhanced Table View

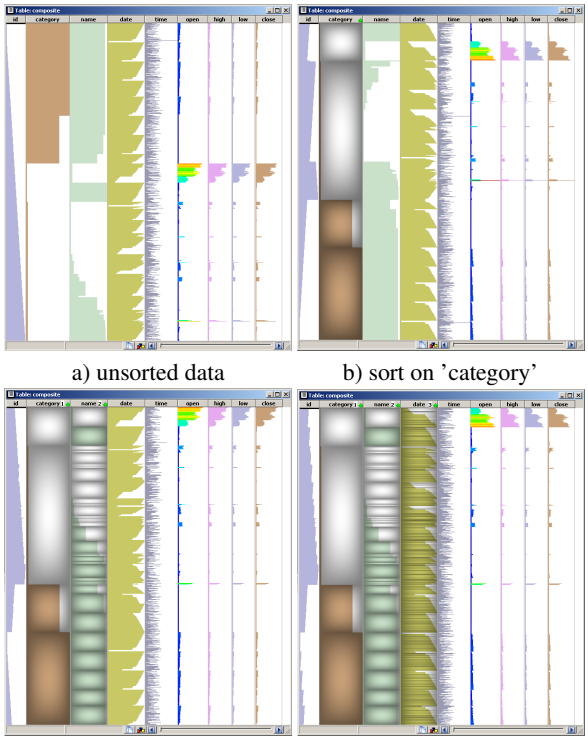
The enhanced table view builds on the basic table lens [RC94]: cell rows become pixel rows, drawn as colored bar graphs. We extend and adapt this idea as follows.

First, we treat all data attributes similarly, whereas [RC94] inferred the visual mapping from the attribute type: numeric attributes were drawn as bar graphs, categorical ones were drawn as colored 'swatches', whose position and/or color indicate the category. We use the bar graph to represent any data type. The bar length and/or color maps the attribute value. We support any data type having 'less-than' and distance operations. The less-than operation is needed to compute the data range (minimum and maximum). The distance operation, which yields a real value given two data elements, maps data values to bar lengths and/or colors. Providing these operations for numeric, date, and time types is trivial. For text data, the less-than operator is the lexicographic order. We define the distance $d(s,t)$ between two strings $s = \{s_i\}_{i=[1,S]}$ and $t = \{t_i\}_{i=[1,T]}$ as $d(s,t) = \sum_{i=1}^{L} A^{-i}|s_i - t_i|$, where A is the alphabet size and s,t are right-padded with nulls to the length $L = max(S,T)$. This distance maintains lexicographic order, whereas other known string distances, e.g. the Levenshtein (edit) distance and variants do not [Lev66]. Categorical data are treated as string data. We can replicate all presentation methods from [RC94] with our colored bar graph: categorical data maps to a sequence of distinctly colored rectangles. By varying the graph scale, all rectangles get the same size, yielding the color-by-category metaphor of [RC94]. Using a thick line graph instead of a bar graph yields the 'swatches' presentation of [RC94].

We next introduce the multiple sort technique. The original table lens described in [RC94, PR96, TR97] and the Inxight tool [Inx05] allows sorting one column at a time by clicking on it. This shows the column values' distribution, e.g. if there are many small, large, or average values, and also reveals correlations among different columns. However, scenarios such as 'show some stock data sorted on industry, then on company, then on date' are not directly possible. *Multiple sorts* support such scenarios. By shift-clicking on several column labels $c_1..c_k$, we sort the rows first on c_1, then sort the row ranges of equal c_1 values, called a cluster, on c_2, and so on, just as one would do in a spreadsheet. This partitions the table into row ranges (clusters) which share more column values as we go from the first (c_1) to the last (c_k) sorted column, similar in concept to the icicle plot [BN01] and permutation matrices [Ber81]. A second click on a sorted column c_i changes the sort order, a third click cancels the sorting of that column and all subsequently sorted columns $c_j, j > i$. This simple but powerful technique visualizes the distribution and/or variation of several attributes (the unsorted columns) grouped by desired properties (the row ranges on the sorted columns).

However useful, multiple sorting can generate hard-to-interpret bar graphs, since the created clusters are not directly visible. We solve this by drawing the clusters as luminance cushions over the bar graphs. Luminance cushions have been previously used to show structure over existing data plots [TVvW04, VTvW05]. We use here the same idea. A cushion is a 2D luminance function $c(u,v) = (1-s) + s(1-4(u-0.5)^2) * (1-4(v-0.5)^2)$, where the u,v parameters span the [0..1] range and $s \in [0..1]$ is the cushion's visual strength. We blend cushions multiplicatively over the bar graphs, which modulates their luminance, as shown by Fig. 1. We start from the raw, unsorted stock data (Fig. 1 a), then multiply sort on industry category (Fig. 1 b, second col-

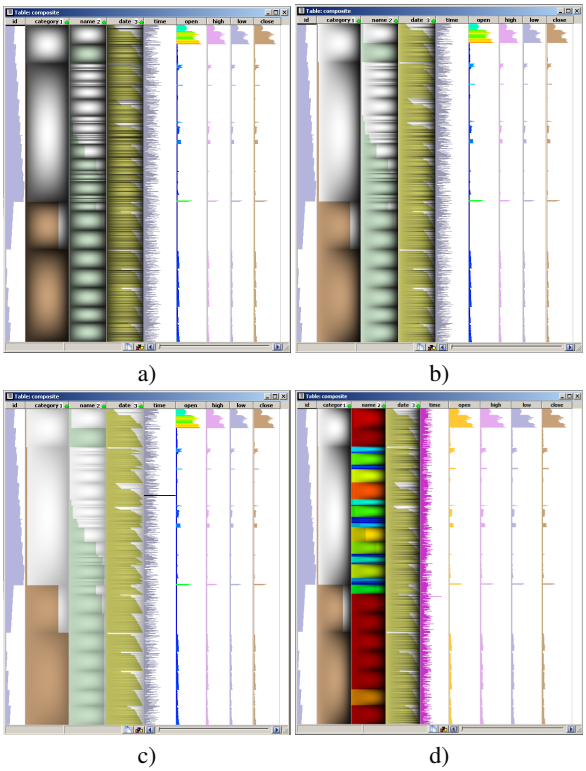
umn from left), then on company name within each category (Fig. 1 c, third column form left), and finally on date within each company (Fig. 1 d, fourth column form left). The sorting order is shown by small labels (1,2,3,...) on the column labels. Clearly, the clusters are now visible (Fig. 1).



a) unsorted data b) sort on 'category' c) sort on 'name' d) sort on 'date'

Figure 1: *Multiple sorting in the table view*

We can further refine the cushion design. Lowering the cushion saturation s towards zero increases the contrast, making the clusters more visible: Compare Fig. 2 a ($s = 0.05$) with Fig. 1 b ($s = 0.7$). Still, too saturated cushions may obscure the bar graphs. To solve this, we set $s(u) = s_0(1-k+ku)$, i.e. vary the saturation s linearly from 1 (invisible cushions) to some user-specified s_0 in the horizontal direction u. The constant k controls the horizontal fading: $k = 0$ gives the basic design $s = s_0$ (Fig. 1 b,Fig. 2 a), whereas $k = 1$ fades cushions from totally transparent at the left ($u = 0$) to the desired strength s_0 at the right ($u = 1$) (Fig. 2 b). The bar graphs are *left*-aligned, while the cushions are more visible in the *right* column half - we thus minimize their overlap, yet maintain their correlation by blending. Setting $k = 0.95$, $s_0 = 0.35$ is a nice compromise showing fainted cushions that do not obscure the bar graphs (Fig. 2 c), yet convey the clusters. Instead of letting the user vary two parameters $k, s \in [0,1]$, we use a single parameter $q \in [0,1]$ and set $k = [5q]/5$, $s = 5(q-k)$. This effectively combines five different k values with the full s range in a single user parameter. Finally, we can color cushions by their size using a blue (low) to red (high) rainbow colormap (Fig. 2 d, 'name' column, third from left). This shows the size distribution of equal-value row ranges. In our case, this supports the query 'find most traded companies' *without* having to sort the 'name' column on the size of its clusters. These companies are shown by the red cushions - since they have the most table rows, they are the most traded ones.

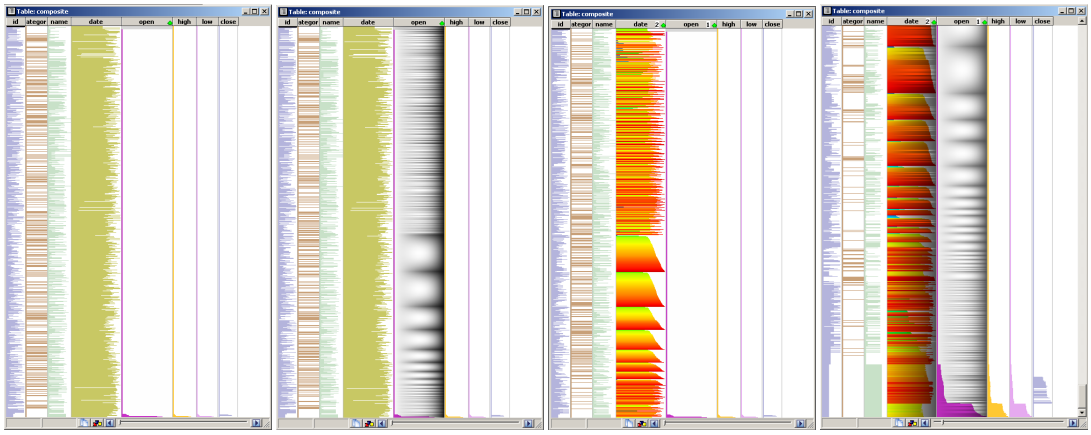


a) b) c) d)

Figure 2: *Cushion shading and coloring design*

Cushions have a second use. They show equal-value rows, so they help finding high-frequency, small-scale variations. In Fig. 3 a we have ascendingly sorted the 'open' value of a set of shares (fifth column from left). The bar graph seems to show that about 95% of the shares have the same low 'open' value, whereas the other 5%, placed at the column bottom by sorting, span the rest of the value range till the maximum. A zoom-in of the bottom-most 3000 rows is shown in Fig. 3 d. In reality, there are variations along this seemingly flat line, but these are so small that they map to subpixel distances, so the bar graph cannot show them. Drawing cushions breaks the flat line into equal-value ranges, revealing high variation regions, and also a few large, truly constant-value, regions (Fig. 3 b).

Multiple sorting emphasizes equal-value regions too. In Fig. 3 c, we have sorted the 'open' column (fifth from left), and then the the 'date' column (left to the 'open' column), and visualized 'date' with a blue-to-red colormap. As explained, this sorts the row ranges of equal 'open' value on



a) b) c) d)

Figure 3: *Interaction between cushions, sorting, and equal-value row ranges*

the 'date' value. We see indeed that the sorted, sawtooth-like, blocks in the 'date' column (Fig. 3 c) match the 'open' column as visualized with cushions in Fig. 3 b. However, the cushions show such patterns without having to sort the 'date' column.

However effective, cushions can overload the image with too much information. Sometimes we do not care to see the *exactly* same-value row ranges, but would like to use some tolerance to reduce the number of displayed cushions. We achieve this by a bottom-up agglomerative clustering of the same-value row ranges, using an average-link strategy [JMF99]. Clusters having most similar values are found and merged, the new cluster's value being set to the input clusters' values weighted by their row counts. We obtain a cluster tree with the same-value row ranges as leafs and a whole column as root. We merge only adjacent clusters, in row space, so the clustering process is $O(NlogN)$ for N rows, i.e. takes subsecond time for e.g. 50000 rows on a 2.0GHz laptop PC. Note that this tree is different from the implicit hierarchy of nested row ranges created by multiple sorting. That is, there is a cluster tree for every clustered column, whereas there is a *unique* hierarchy, having row ranges as nodes, and as many levels as multiply-sorted columns, for a given table. We now draw the cushions on some intermediate tree level instead of all its leafs, as explained below.

Figure 4 left shows the agglomerative clustering of the 'date' column (drawn with a rainbow colormap) after the scenario in Fig. 3. Clearly, the date clusters do not have now the same values. Next, we cluster the 'open' column (right of 'date'), leading to the result in Fig. 4 right. We see how the near-equal rows, previously visible in Fig. 3 b as distinct cushions, are now grouped in one tall cluster. Users can interactively choose the desired simplification level, effectively creating on-the-fly level of detail visualizations of the column data.

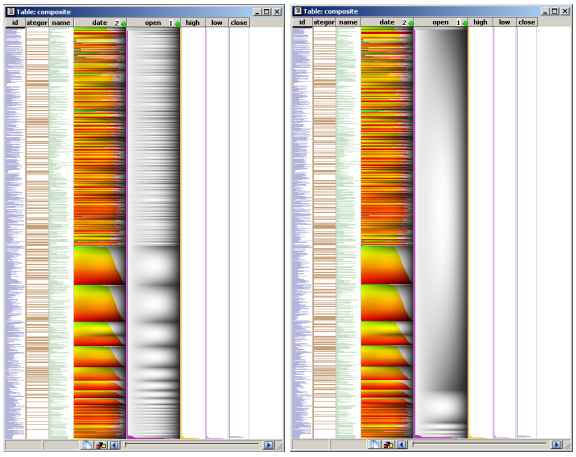


Figure 4: *Visualization simplification via clustering*

Clustering has two other uses besides the visual level-of-detail. Brute-force rendering is quite slow for tables of tens of thousands of rows. Moreover, subsampling artifacts occur, as several cushions map to a single pixel row. In Fig. 5 left, first column, we see some large cushions. Actually, we should see one cushion per row, as this is the integer row-id column $id = [0, 50000]$ (see the underlying bar graph). This problem appears as 50000 1-pixel-tall cushion textures are drawn on the 500 vertical pixel lines of the window. Subsampling yields different patterns on different graphics cards, depending on how textures are mapped on very thin polygons. However, all show fake 'large clusters' as in Fig 5 left. Both speed and subsampling problems can be alleviated as follows. First, we cluster all table columns. Next, we use only clusters taller than 1 pixel to render the bar graph, which solves the speed issue, and only clusters taller than a few (e.g. 5) pixels to render the cushions, which

solves the aliasing issue (Fig. 5 right). We do this by traversing the cluster tree depth-first from the root and rendering clusters just below the target size. This technique reduced the rendering time for a 50000 rows, 8 columns table from 1.2 seconds on a 2GHz laptop with an Nvidia Quadro card, and from 1.7 seconds on a 2GHz PC with an older Nvidia GeForce 4 card, to under 0.01 seconds in both cases. We always draw less clusters than screen pixel rows, so we guarantee a high *and* constant display rate for any input table size. The extra time and memory to pay goes into the clustering phase, which is redone only when the table structure changes.

Our clustering technique is conceptually related to the accordion drawing method for displaying large trees and sequences [SHM05]. Both methods guarantee constant rendering time for any input data size, by performing some kind of clustering. However, input data types, clustering constraints, and final results are different in the two cases. Also, accordion drawing focuses on guaranteeing visibility for selected subpixel objects. So far, we only aim to guarantee rendering time and reduce subsampling artifacts due to thin textures. However, we could adapt our method to guarantee visibility of specified (e.g. selected) row-ranges, by e.g. modifying the cluster similarity to penalize merging such ranges, or by modifying the render traversal to always return these ranges. Such experiments are subject of future work.

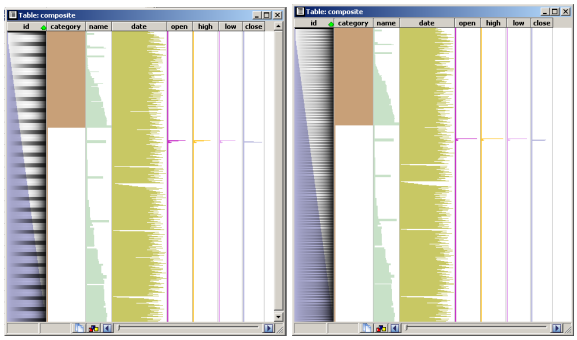


Figure 5: *Subsampling artifacts (large cushions, left) are removed by clustering (right)*

Finally, we propose a zooming technique that complements the focus and context metaphor of [RC94]. Using a global zoom slider, we continuously vary the cell height h from the text size (e.g. $h = 18$ pixels) to the maximum zoomed out level $h = h_{min}$, which fits the whole table to the window height (Fig. 6). The table view starts in the familiar text mode, similar to a spreadsheet. As we zoom out, the cell and text font size gradually decrease. Simultaneously, we decrease the text and grid line opacities and increase the bar graphs' opacity. A careful design of the transparency and size functions, shown in Fig. 7, effectively smoothly morphs the textual spreadsheet into a graphical visualization.

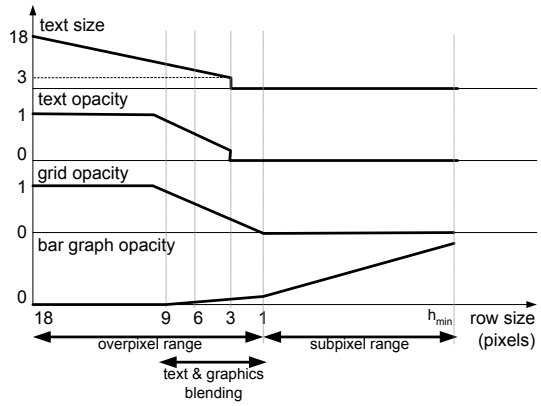


Figure 7: *Blending text and graphics*

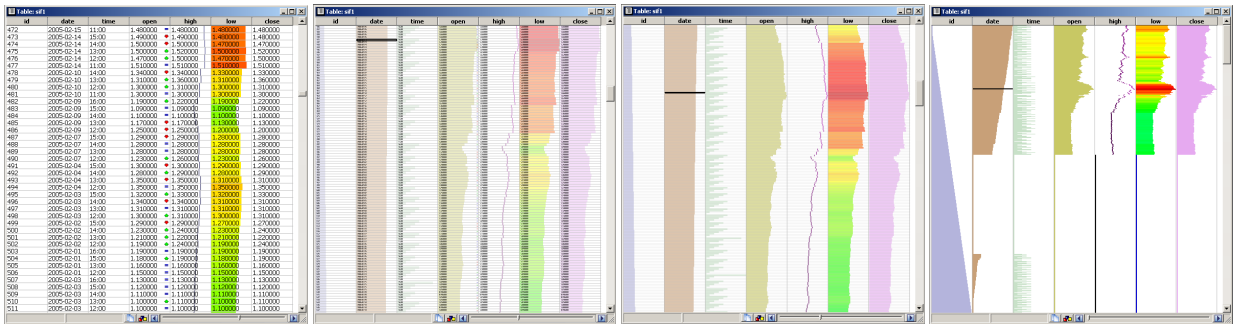
We show context information with both dynamic pop-ups under the mouse pointer and the status bar, in contrast to the text focus elements proposed in [RC94]. Our users found the textual table morphing into a graphical representation more intuitive than mixing the two in one image by means of text focus elements. Overall, our zooming resembles the MatrixZoom technique for visualizing large matrices [AvH04]. However, we involve more, and different, parameters (cf. Fig. 7) as our visual table elements are different.

3.2. Parameterizable Treemap View

The multiple sort technique (Sec. 3.1) effectively creates data hierarchies on-the-fly. Although the cluster cushions reflect these hierarchies, we show next how to use treemaps to provide better, and different, insights. The second view of our framework is a treemap view, in which both the layout and visual mapping (rendering) are user-controllable. This lets one create a rich set of visualizations, as shown by the following scenarios.

Figure 8 shows a common stock data scenario. We sort the rows first on the 'category', then on (company) 'name', and then on 'date', creating a four-level deep hierarchy with three clicks, the leafs being rows with the same 'date'. Next, we drag-and-drop this hierarchy from the table view into the treemap view. To visualize it, we specify the *layout*, i.e. how to position the hierarchy nodes, and *rendering*, i.e. how to draw them, by using several preset visualization scenarios, as follows.

Figure 8 visualizes data from the Romanian Rasdaq technology index [Ras05], which contains over 2000 companies. Discovering 'quick movers' in this emerging market is a task the brokers named as challenging and important, as most Rasdaq companies have sporadic trades and hard to predict behavior. This visualization uses a layout preset: squarified treemap [BHvW00] on the first ('category') and second ('name') levels and a strip treemap [BE95, BSW02]



a) $h = 18$ b) $h = 6$ c) $h = 3$ d) $h = h_{min}$

Figure 6: *Table view zooming from text level (a) to whole-table-in-window (d)*

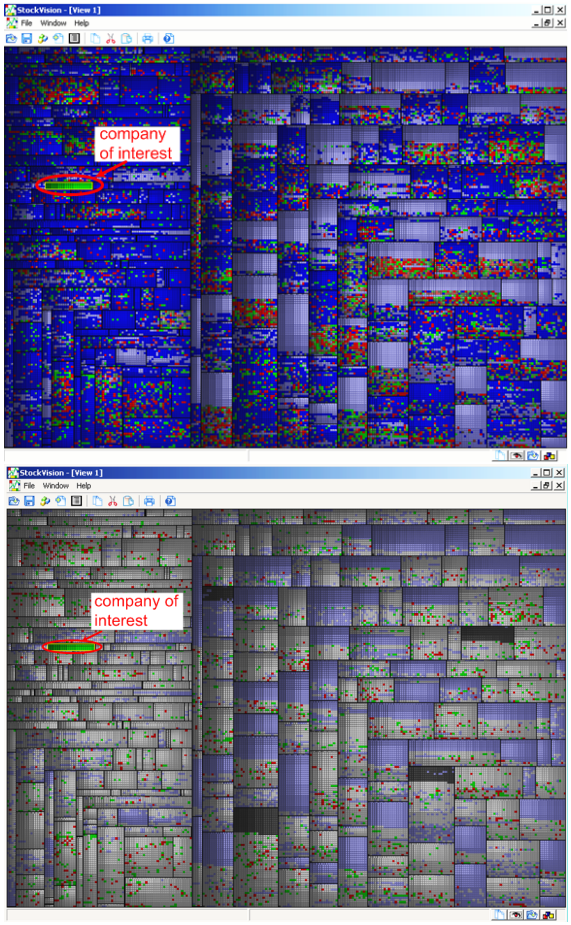


Figure 8: *Treemap view: stock data visualization scenarios*

on the third ('date') level. Since the strip layout maintains children order in a parent, and the 'date' level is sorted, we easily read the visualization as increasing dates (days) associated to the treemap leaves in a left-to-right, top-to-bottom order. The size of the treemap leafs is constant, as each one shows a single day. Thus, their parents (companies) and grandparents (industry categories) get sizes proportional with the number of days we had data for their respective elements. After specifying the layout, we select for the third ('date') level a renderer that colors the treemap nodes as a function of the difference between the share's 'open' and 'close' values during the node's date value (1 click). Red shades map to losses, green shades to gains (the color saturation indicating the magnitude), blue unchanged values, light blue missing values, light gray values outside a user-specified value range of interest, and dark gray values outside a user-specified period of interest respectively. Users can customize the rendering settings, as well as filter the data for a specific time period, in a GUI. Building this visualization took only a few clicks. The table view (e.g. Fig. 3) showed that there's little variation in the individual 'open' and 'close' columns. Figure 8 top strengthens this discovery saying that most shares do not change price ('open' equals 'close'), shown by the large amount of blue treemap cells. The large light blue blocks in the upper half of several company 'name' cells shows that data is missing for the first part of the considered period. We also see that shares either do not change value (blue zones) or have a very dynamic behavior (dense red-green alternations). This allows discovering which companies are most active, and during which period of time, which was exactly one of the aims of the stock brokers interested in our application. Figure 8 bottom shows a related scenario. Here, we select using the GUI a specific period (2001-2005 out of the full 1999-2005 data range), and a specific price change range (above +5% and below -5% of the daily value). A few dark gray regions appear, indicating the filtered-out period. A large part of the visualization becomes light gray, as these shares have not changed outside the specified price range during these days. We can now focus on the remaining green and red cells, and we discover an outlier: the compact upper-left green block. This is a company that has continously increased for all the considered period, and is thus an interesting buy option.

Figure 9 top shows a similar scenario to Fig. 8, this time however for the about 300 heavyweight (mainstream) Romanian Exchange companies. We quickly see that these companies are much more frequently traded and exhibit a different evolution pattern that the ones in Fig. 8. Hence, we created here a fourth hierarchy level by multiple sorting, the 'time of day', to show how many intraday samples we have for each stock. Creating this level for the data in Fig. 8 would be useless as we don't have intraday values for that dataset.

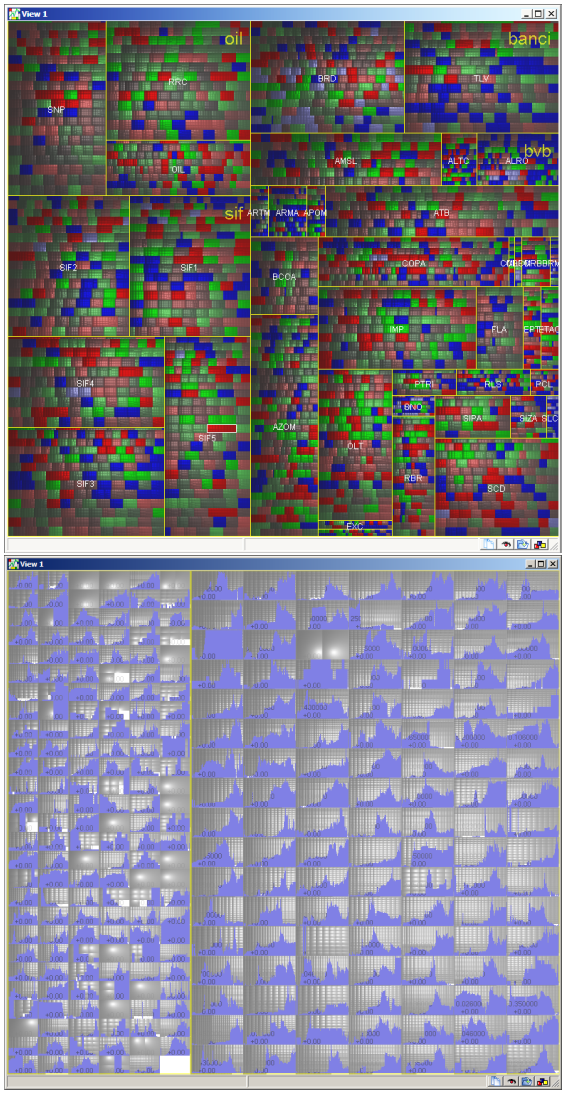


Figure 9: *Treemap view: intraday change (top); open price versus time (bottom)*

Figure 9 bottom shows a third scenario. We multiply sort the table on categories, then on share names, and finally on a derived 'date & time' attribute (3 clicks). The last attribute is computed from the primary 'date' and 'time' columns by a simple SQL operation. Next, we open a treemap view that uses a layout preset: squarified for the first ('category') level and a regular grid layout for the second ('name') level. The grid layout divides the parent space in equally sized children of given aspect ratio. Finally, we select a yellow border renderer for the 'category' level, a 2D graph renderer for the 'name' level, and a cushion renderer on the third 'date & time' level. We configure the graph renderer to draw the 'open' (Y axis) versus 'date & time' (X axis) values. In this way, we constructed one of the most widespread, classical types of stock visualization, in a few clicks. As shown by the white (bagkground) spots in Fig. 9 bottom, the regular grid layout on the second level cannot always perfectly fill its parent, since it enforces same-size children with fixed aspect ratio. If we relax one of these constraints, e.g. using a strip layout, we can achieve a perfect space fill. However, our users preferred the regular grid as it produced nicely shaped, equal graphs which are easy to read. The combination of graphs and rendered cushions display both the evolution of the shares' open values and the number of time samples we have data for them. This image is conceptually quite similar to the table view's cushions and bar graph combination (Sec. 3.1), but uses a different spatial layout. Clearly, the above scenarios are not final and can be further improved. However, we present them here to illustrate the simplicity of visualization construction in our framework.

Our idea of visualizing on-the-fly constructed hierarchies with treemaps is similar with the flexible hierarchies of Treemap 4.0 tool [CPS04]. However, we construct (and visualize) the hierarchy in the table view and then examine it in the treemap view, whereas [CPS04] uses the treemap view, and additional text-based GUIs, for this.

4. Discussion

We implemented the TableVision framework in C++ using OpenGL and the wxWidgets GUI toolkit. Careful user interface design and several speed optimizations, such as the clustering technique (Sec. 3.1) were required to gain user acceptance. Surprisingly, SQL database access was a major speed bottleneck, even for in-memory databases, that was removed by extra caching techiques. TableVision was tested by several stock brokers. We explained the table view and treemap view basic functions in a few online training sessions of several hours, and provided some preset scenarios to start with (see Sec. 3.2). To limit confusion, we offered also a few presets for the many tool parameter values. After a while, the users were able to navigate the data and got interested to create own scenarios. The table view's more powerful features, e.g. multiple sorting and cushions, were accepted only after the resulting hierarchies were understood in the treemap view. Overall, the evaluation was positive, as the combination of table and treemap views helped achieving several planned discovery goals, e.g. finding 'market movers' which became object of further study.

Acknowledgements

We acknowledge the help of Cristian Micu and Rãzvan Pasol from ING Romania and Intercapital Invest for providing us with stock data, insight into brokering strategies, and tool evaluation.

5. Conclusions

TableVision tightly integrates enhanced table and treemap views to create visualizations for table data. We extend the table lens with several mechanisms. First, we use multiple sorts to create hierarchical structure from unstructured tables. Second, we add shaded cushions to help seeing the data structure and small-scale variations, and combine these with the underlying bar graphs with minimal visual clutter. Third, we use a bottom-up clustering scheme to render very large tables interactively and also minimize subsampling problems. Finally, we use a smooth zooming mechanism to navigate between text and graphics. The treemap view shows how we can use several presets combining layouts and renderers to create a rich set of visualizations for the hierarchical data with just a few clicks. We connect the two views by allowing to drag-and-drop data from one to the other. Overall, we achieve a simple to use and learn, yet versatile exploration framework for table data, in which users can gradually pass from the familiar spreadsheet view to the more complex table lens and treemap views.

We plan to extend TableVision in several ways. First, there are yet unexplored directions for compactly and effectively showing information in the table view. Second, we plan to include new renderers and layouts in the treemap view and validate our framework in new application areas.

References

[AvH04] ABELLO J., VAN HAM F.: Matrixzoom: A visual interface to semi-external graphs. In *Proc. InfoVis* (2004), IEEE Press, pp. 183–190.

[BE95] BAKER M., EICK S.: Space-filling software visualization. *J. Vis. Lang. Comp.*, 6 (1995), 119–133.

[Ber81] BERTIN J.: *Graphics and Graphic Information Processing*. De Gruyter, 1981.

[BHvW00] BRULS M., HUIZING K., VAN WIJK J. J.: Squarified treemaps. In *Proc. VisSym* (2000), IEEE Press, pp. 33–42.

[BN01] BARLOW T., NEVILLE P.: A coimparison of 2d visualizations of hierarchies. In *Proc. InfoVis* (2001), IEEE Press, pp. 131–138.

[BSW02] BEDERSON B., SHENIDERMAN B., WATTENBERG M.: Ordered and quantum treemaps: Making effective use of 2d space to display interactions. *ACM TOG 21*, 4 (2002), 833–854.

[CPS04] CHINTALAPANI G., PLAISANT C., SHNEIDERMAN B.: Extending the utility of treemaps with flexible hierarchy. In *Proc. Information Visualisation (IV)* (2004), IEEE Press, pp. 335–344.

[Inx05] INXIGHT: *Inxight Data Visualization Solutions*, 2005. www.inxight.com.

[JMF99] JAIN A., MURTY M., FLYNN P.: Data clustering: a review. *ACM Comp. Surv. 31*, 3 (1999), 264–323.

[Lev66] LEVENSHTEIN V.: Binary codes capable of correcting deletions, insertions, and reversals. *Sov. Phys. Dokl.*, 6 (1966), 707–710.

[Mar05] MARKETMAP: *Map of the Market*, 2005. www.smartmoney.com.

[PR96] PIROLLI P., RAO R.: Table lens as a tool for making sense of data. In *Proc. AVI (Advanced Visual Interfaces)* (1996), pp. 59–63.

[Ras05] RASDAQ: *Rasdaq Technology Stock Exchange Index*, 2005. www.rasd.ro.

[RC94] RAO R., CARD S. K.: The table lens: Merging graphical and symbolic representations in an interactive focus+context visualization for tabular information. In *Proc. SIGCHI* (1994), ACM Press, pp. 1–7.

[SHM05] SLACK J., HILDEBRAND K., MUNZNER T.: Prisad: A partitioned rendering infrastructure for scalable accodion drawing. In *Proc. InfoVis* (2005), pp. 41–48.

[Shn92] SHNEIDERMAN B.: Tree visualization with treemaps: 2d space-filling approach. *ACM TOG 11*, 1 (1992), 92–99.

[TR97] TENEV T., RAO R.: Managing multiple focal levels in table lens. In *Proc. InfoVis* (1997), pp. 59–63.

[Tuk77] TUKEY J. W.: *Exploratory data analysis*. Addison-Wesley, 1977.

[TVvW04] TELEA A., VOINEA L., VAN WIJK J. J.: Ezel: A visual tool for performance assessment of peer-to-peer file-sharing networks. In *Proc. InfoVis* (2004), IEEE Press, pp. 41–48.

[VTvW05] VOINEA L., TELEA A., VAN WIJK J. J.: Cvsscan: Visualization of code evolution. In *Proc. ACM SoftVis* (2005), ACM Press, pp. 47–56.

[vWvdW99] VAN WIJK J. J., VAN DE WETERING H.: Cushion treemaps: Visualization of hierarchical information. In *Proc. InfoVis* (1999), IEEE Press, pp. 73–78.

[Wat99] WATTENBERG M.: Visualizing the stock market. In *Proc. SIGCHI (abstracts on Human factors in computing systems)* (1999), ACM Press, pp. 188–189.

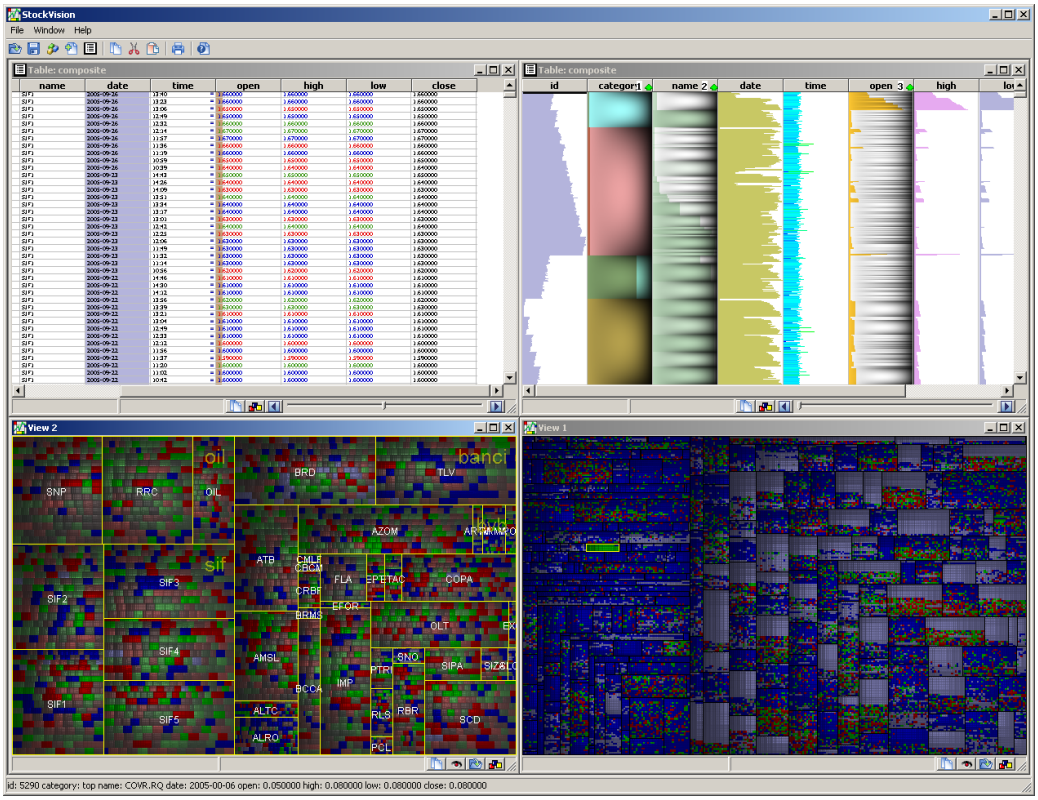


Figure 10: *TableVision overview: standard table, enhanced table lens, and treemap views*