Visualizing Business Data with Generalized Treemaps

Roel Vliegen, Jarke J. van Wijk, Member, IEEE, and Erik-Jan van der Linden

Abstract—Business data is often presented using simple business graphics. These familiar visualizations are effective for providing overviews, but fall short for the presentation of large amounts of detailed information. Treemaps can provide such detail, but are often not easy to understand. We present how standard treemap algorithms can be adapted such that the results mimic familiar business graphics.

Specifically, we present the use of different layout algorithms per level, a number of variations of the squarified algorithm, the use of variable borders, and the use of non-rectangular shapes. The combined use of these leads to histograms, pie charts and a variety of other styles.

Index Terms—Information visualization, treemap, business graphics, hierarchical data.

1 INTRODUCTION

In many organizations information is collected and used to support decision-making. We refer to such information as business information, even though it is also used in the non-profit sector. Tables provide information on a detailed level; aggregation is used to generate information on trends and global patterns. Information can be presented in many ways, with text and graphics as main categories.

Visualization allows for the analysis of larger data sets than textual presentation via tables. Visualization relies on the visual system of users to quickly transfer large amounts of information. It moves the burden of reasoning partially to the visual system, freeing other parts of the cognitive system for higher-level tasks. This can lead to immediate insight in complex phenomena that otherwise would be hard to grasp. Also, visualization can provide context to search results and make outliers visible. Context and outliers are present but often not perceivable in tables; aggregation condenses information, leaving out context and outliers altogether.

In practice, the value of visualization is recognized, judging from the ubiquitous use of business graphics, such as bar charts and pie charts. Business graphics lends itself well for presenting overviews of a limited number of values in a single image, but such overviews lack detailed information on individual cases.

The visualization community has studied how large amounts of detailed information can be presented. Treemaps [1] visualize large amounts of hierarchically organized data. Treemaps have been used successfully for visualizing various kinds of data, such as the content of file systems [1], 2], stock market data [3], process control data [4], and source code of large programs [5, 6]. However, treemaps are still rarely used for visualizing business information. One reason is that treemaps are less effective for presenting aggregate information, compared to business graphics; another reason is that treemaps are simply less familiar than business graphics.

In this paper we present a generalized treemap algorithm that aims at bridging this gap. We propose extensions to treemap algorithms such that treemaps can be created with the look and feel of business graphics. These treemaps combine the strong points of business graphics and treemaps: A familiar presentation is provided which allows for quickly understanding aggregate information, and simultaneously provides access to information on a detailed level. Algorithms presented in this paper have been implemented in a commercially available system. Applications of this system in two areas are discussed.

In section 2 we describe the data model used, business graphics and current treemap methods. Next, we present extensions on layout algorithms (section 3), handling of the size (section 4), and shape (section 5). Results are presented in section 6, followed by conclusions and suggestions for future work in section 7.

2 BACKGROUND

2.1 Data model

We assume that the data to be presented is given in a table. Each row of the table represents an item (student, product, purchase, etc.), each column of the table represents an attribute (age, price, date, etc.). A cell $A_{ij}$ represents the value of attribute $j$ for item $i$. This simple model is surprisingly generic. Multiple tables can be joined in a single table (though at the expense of redundancy), relational data can be handled as well. Each attribute can have an arbitrary data type, such as string, integer, real number, or date.

Treemaps visualize hierarchical data, hence the table has to be converted into a tree. This is done by selecting a sequence of attributes and grouping the items accordingly [7, 8]. Each attribute corresponds with a level in the tree. A level contains all nodes that have the same distance from the root. Interior nodes represent sets of items, and leaf nodes represent individual items. The number of children of a node corresponds with the number of different valuations of the items whose leaves are descendants of this node. To limit this number it is often convenient to map attributes with many different valuations, such as real numbers, to a limited number of discrete categories. Also, in some cases, discussed later, it can be convenient to use all possible valuations for an attribute for each node on a level, such that the number of branches per node is the same for each level.

A simple example is shown in Figure 1. Interior nodes are blue and leaf nodes are yellow. The attribute age is mapped to the two categories child (C) and adult (A). In this example there are no male adults. If a dummy branch is introduced (shown dashed), the tree is regularized for this level.

Roel Vliegen is with Magnaview, E-Mail: roel@magnaview.nl.
Jarke J. van Wijk is with Technische Universiteit Eindhoven, E-Mail: vanwijk@win.tue.nl.
Erik-Jan van der Linden is with Magnaview, E-Mail: ej@magnaview.nl.
Manuscript received 31 March 2006; accepted 1 August 2006; posted online 6 November 2006.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.
2.2 Business graphics

Business graphics is a broad term, if we define it as all possible different graphical depictions of business data. In [9] an encyclopedic overview is given of 450 pages, with 4–10 examples per page. We use the term business graphics here in a narrower sense: Well-known graphical depictions of data that are used for instance in spreadsheets, presentations, on television, in reports, etc. Figure 2 shows some examples, produced with a well-known spreadsheet.

Such business graphics share a number of characteristics. First of all, they are familiar. Pie charts and bar charts are often already taught in primary school; they do not have to be explained when used in a presentation. Second, they are simple and effective. The relation between geometry and values is easy: Quantities can be read from position, length, angles, and areas [10]. Third, each type has its own purpose and strength. Pie charts allow for easy judgment of fractions, for instance whether a sector takes more or less than 1/4; stacked histograms allow us to judge the relative distribution for quantities over various items. Fourth, they are used for and limited to modest amounts of data. Typically at most 10 to 20 numbers are displayed, for just one or two attributes.

This limitation is not a problem when the aim is just to communicate high-level, aggregate information. However, in many cases such aggregate information raises just new questions. For instance, if a total is larger than expected, one wants to inspect the contribution of individual elements. For that purpose, pixel bar charts [11] fill the bars with detailed information. Obviously, also, extra graphics that show more detailed information with a smaller scope can be provided, but this can easily lead to loss of context and does not allow for easy comparison.

2.3 Treemaps

Treemaps were introduced by Johnson and Shneiderman [1] in the early nineties, and have gained increasing popularity. An up to date overview on work in this area can be found at the website of HCIL [12], here we limit ourselves to work that is most relevant for the approach presented here. In the following we will italicize features of related work upon which we present variations. Treemap algorithms represent hierarchical data sets via recursive subdivision. The standard shape used is the rectangle, although variations such as Voronoi diagram based subdivisions have been proposed [13]. Each rectangle is subdivided into smaller rectangles. A small, fixed margin can be used to emphasize the hierarchical structure by nesting, or for the placement of labels. Another approach to visualize the hierarchical structure is via the use of shaded cushions [14].

The area of the smallest rectangles, representing items, can be used to visualize a quantity, such as file size or market share. The area of enclosing rectangles is derived from the sum of the values represented by enclosed rectangles. In addition, color can be used to visualize an additional attribute, such as file type or growth.

For the layout of the rectangles different algorithms are used. In the standard slice and dice method [1] a rectangle is subdivided in one direction (for instance horizontally), and for the next level this direction alternates. The layout is easy to understand, but often leads to thin rectangles, which are hard to see, compare, and point at.

An alternative is the squarified layout algorithm [15]. Here rectangles are added one by one to a strip along the longest edge (left or bottom) in the rectangle. If addition of a new rectangle would give an increase in the highest aspect ratio so far for that strip, a new strip is started. The best results are obtained when the rectangles are sorted for decreasing size first. This algorithm gives rectangles that are much more square, but the overall structure is harder to see. Also, the resulting layout is sensitive to changes of the data.

The strip layout algorithm [16] is a compromise between the slice and dice method and the squarified method. Here, rectangles are added to strips, just like in the squarified method, but here the direction of the strips is fixed within a level and alternates per level, average aspect ratios are evaluated, and no sorting is used. As a result, the rectangles are not as square as for the squarified method, but the resulting layout is more robust to change.

The quantum treemap algorithm [16] only uses sizes that are multiples of a fixed quantum for displaying rectangles with fixed aspect ratios more regularly. For almost all methods described, the same layout method is used for each level. An exception is the combined use of the quantum and strip treemap algorithms [16].

With these algorithms convincing visualizations can and have been produced. Nevertheless, by varying on these a range of other visualizations can be obtained. In the following sections these are explored.

3 Layout

In this section we reconsider layout algorithms. We first introduce mixed layouts, followed by variations on the slice and dice layout and the squarified layout. Finally we present a new layout algorithm, the matrix layout. In the examples the same data set is used. It represents the telephone administration of a student house. Each of the 1173 items is a phone call, attributes are the person who made the call (Gijs, Paul, Roel, Linda, Koert, Geen), the type of call (LDC - Long Distance Call, LOC - Local call, MOB - call to Mobile number, INT - International call, SRV - service call, OTH - other calls), the type of cost (S - Standard, E - Evening), and the cost of the call in euro which is mapped to size. Color is used to identify
persons. Furthermore we highlighted calls with a cost of €0.35 to indicate the effects of the variations of the layout on individual calls.

3.1 Mixed treemaps
The slice and dice algorithm creates structured and regular layouts. Figure 3a shows an example. It is easy to see here that most costs are made for calls to mobile numbers, and that Roel often calls long distance. A drawback is that nodes can become very thin and elongated, and thus become difficult to compare and select, or, even worse, may not even be visible. The squarified algorithm solves this by producing fairly square nodes, but here the global structure is much harder to understand (Figure 3b).

We propose mixed treemaps, where the strong points of these algorithms are combined, while avoiding the weak points. The use of the slice and dice treemap algorithm for the higher levels and the squarified treemap algorithm for the lowest level gives a treemap that has a structured layout, but also has clearly visible, squarified, nodes representing items (Figure 3c).

In principle, an algorithm could be chosen for each individual internal node. However, this is unpractical because of the large number of nodes and the data dependency it would create. Therefore, we allow the user to choose per level of the tree which algorithm has to be used. The concept of level matches well with the intuition of user: Each level adds a new subdivision that can be reasoned about separately.

A first example is shown in Figure 4, in following sections other examples are given. On the highest level a distinction between local, long distance and mobile calls is made like in Figure 3c. Here, an additional level makes a distinction between standard and evening types of cost. The slice and dice algorithm alternates the strip direction, leading to hard to compare contents. The use of fixed vertical strips for the highest levels gives columns side by side that allow for a direct comparison of fractions, revealing that the LDC/E/Roel and LDC/S/Roel fractions are equally large.

3.3 Squarified treemaps
In the squarified algorithm rectangles are added one by one to a strip along the longest edge in the rectangle. If left or bottom edges are used to position new strips, the small nodes are located in the top right corner. This leads to a diagonal structured layout, where the small nodes are located in one corner (Figure 5a). We call the point where small nodes are located the vanishing point.

The use of just left and bottom edges is a choice. We enable users to position the vanishing point at will, such that other visualizations can be obtained. Figure 5b and Figure 5c show the effect of positioning the vanishing point at the center of the top edge and in the center of the rectangle. Integration of this feature in the squarified layout algorithm is straightforward. Each time that a choice can be made to use either the left or right edge (or, similarly, the top or bottom edge) the edge is chosen such that the vanishing point remains as close as possible to the center of the remaining rectangle.

In the standard squarified algorithm the nodes are sorted for decreasing size. Experimentally it was found that this gave the best results with respect to squarification. However, when the number of rectangles is large, sorting for increasing size also gives acceptable

![Fig. 3: Mixed treemaps.](image)

![Fig. 4: Strip direction.](image)

![Fig. 5: Vanishing point squarified layout.](image)
results. The layouts produced look different: Instead of the smallest, the largest rectangles are positioned in the vanishing points. Figure 5d shows how this can be used. Here, more important nodes are displayed in the center, which seems more natural.

3.4 Matrix layout
In some cases it is desired that all rectangles representing sibling nodes have exactly the same size and shape. One reason is that this allows for a fair comparison, another is that this gives a more smooth visual result. We have therefore added a type of layout, the matrix layout, similar to quantum treemaps [16]. We subdivide the rectangle using a regular grid. When the grid has more cells than items to be placed, dummy nodes are added, and blank cells are used.

Suppose that \( n \) rectangles have to be fit in a parent rectangle with width \( w \) and height \( h \). We subdivide this rectangle into \( r \) rows and \( c \) columns. The values of \( r \) and \( c \) have to be chosen such that \( rc \geq n \). Furthermore, assume the user can specify a desired aspect ratio \( \alpha \) for the cells. The realized value \( \beta \) for the aspect ratio is \((h/r)/(w/c))\). To achieve that \( \beta \) is as close as possible to \( \alpha \), while not introducing too many empty cells, we use the value of \( c \) for which \( |\alpha - \beta| \) is minimal, with \( r = \lceil n/c \rceil \). An example of the use of a matrix layout is given in Section 6.2.

4 Size
In this section we consider the handling of sizes and areas. First the standard solution is presented, followed by a number of variations.

4.1 Constant size
An important strength of treemaps is that the area of rectangles can be used to express a quantitative attribute of items. The use of treemaps to visualize the contents of a hard disk is a prime example: Large files lead to large rectangles. Also, a constant value can be used for leaf nodes. This expresses that every leaf node is equally important and should get the same area.

We refer to the target value to be used for a node as the size. Each node \( n \) has such an associated size \( S(n) \). In the standard algorithms, the size of interior nodes is the sum of the sizes of the children. Also, with each node corresponds a rectangle with a certain area \( A(n) \). In the standard algorithms the rectangles are generated such that for each node \( n \) the following property holds:

\[
\frac{S(n)}{A(n)} = C.
\]

We call this the Uniform Density (UD) property. If this property holds, all areas of all nodes are directly comparable. A consequence of the standard algorithms is that most of the time higher level nodes have different areas. In Figure 6a strips for different types of phone call have different widths; within a strip, sizes per person vary.

However, sometimes it is preferable that all nodes within a level use the same area. One case is when ratios have to be compared within the higher levels, another case is when one wants to enforce that each node in a higher level has equal visibility. Therefore, we introduce the use of constant size for nodes in the higher levels.

The use of constant size allocates for each node the same area. In the standard algorithms this is an option for leaf nodes, we also allow the use of constant size on higher levels. When used on the top two levels of the same sample data, this leads to Figure 6b. Such a matrix structure is useful for studying the effect of two different attributes on distributions. In this case, persons are mapped to the vertical axis and types of calls to the horizontal axis. In Section 6.2 two examples for comparing ratios using constant size are given. Roel's local calls are now equally visible as the other combinations. We see that his few local calls all have the same cost, while the costs of his calls to mobile phones vary strongly. Unfortunately, it is also clearly visible that the UD property is strongly violated: The yellow highlighted nodes are not equally large anymore. The rectangles for the local calls of Roel cannot be compared directly with for instance the rectangles of his long distance calls.

4.2 Adjust for uniform density
As a rule of thumb, the UD property should always hold, treemaps that do not conform to it can be misleading. In our experience, users intuitively interpret area as the most important visual property for associating nodes with quantitative value. Even when users are explicitly told to ignore the area and focus, for example, on the width or height of nodes, this still results in confusion.

Treemaps such as presented in the previous section can be adjusted for UD if we use a fraction of the area of a parent node for its children. Specifically, suppose that a parent node \( p \) has a rectangle with area \( A(p) \), and that we use a rectangle with area \( f_p A(p) \) for its children, where \( f_p \) is a scale factor. Let \( C(p) \) denote the set of all children of \( p \), and \( P \) denote the set of all nodes at the same level as \( p \). For UD now the following constraint has to hold:

\[
\sum_{c \in C(p)} \frac{S(c)}{f_p A(p)} = U, \quad \text{for all } p \in P,
\]

where \( U \) is constant. The maximum value for all \( f_p p \in P \), is one. If we use \( f_p = 1 \) for the most critical node \( p \), we find that

\[
U = \max_{p \in P} \left( \frac{1}{A(p)} \sum_{c \in C(p)} S(c) \right),
\]

which we can use to determine \( f_p \) for all nodes \( p \).

To realize the reduction in area, additional margins are added to rectangles for the level in case. The top, bottom, left and right margins can be enlarged to reduce the area. Various settings are

Fig. 6: Various definitions of size.

Fig. 7: Variable margins for adjustment of areas.
shown in Figure 7. Use of the top margin leads to easy comparison of rows, to obtain easily comparable columns the right margin can be adjusted. These settings lead to the familiar bar charts. Also, all margins can be enlarged, leading to nested rectangles, which can be compared both horizontally and vertically.

Such a table obtains its readability from the clear matrix structure. However, no matrix structure will emerge if columns contain a different number of rows, or rows contain a different number of columns (Figure 8). A matrix structure can be enforced by adding empty nodes, such that each column has the same number of rows and each row has the same number of columns. This can easily be automated. For each relevant level all possible valuations of an attribute are determined, and empty nodes are added where certain values are missing.

![Matrix Structure Examples](image)

**Fig. 8:** Addition of empty nodes to obtain a matrix structure.

### 4.3 Nesting

Margins can also be used to accentuate the hierarchical structure of treemaps, leading to so-called nested treemaps (Figure 9a). Margins use some of the area of nodes, thus leaving less area available for the child nodes. Related work uses fixed margins that have the same width on all four sides [1, 3, 4, 16], often based on the depth of the node in the tree structure [3, 4]. We generalize this, such that, for each level, the width or height of the margins can be individually set for the left, top, right and bottom margins, both in absolute pixel values and percentages of the rectangle.

The use of margins in general violates UD. In related work, small, absolute margins are used and the change in density this causes is neglected [1, 3, 4, 16]. However, even small, absolute margins can result in empty or degenerative children rectangles, thus leading to disappearing leaf nodes. Relative margins that are fractions of the nodes do not suffer from this problem, but are also less suitable for displaying labels.

![Nested Treemap Examples](image)

**Fig. 9:** Addition of empty nodes to obtain a matrix structure.

Margins can be used in several ways. For Figure 9b a fixed subdivision direction was used, in combination with constant size and adjustment for uniform density. This gives a somewhat unclear vertical histogram, but when margins and labels are added (Figure 9c) a clear structure emerges. Note that this resembles (when rotated over 90 degrees clockwise) an icicle plot [17]. To make the structure even more clear, lines can be used that connect points on the boundaries of rectangles of child and parent nodes (Figure 9d).

### 5 Transformations

Pie charts are well known visualizations and are often already taught in primary school. This makes pie charts attractive visualizations; users do not have to learn how to use pie charts anymore. To create treemaps that resemble pie charts, we have added transformations in pixel space.

Transformations map points \( x=(x,y) \) from a source rectangle \( R_s \) to points \( u=(u,v) \) in destination rectangle \( R_d \). The coordinates of the source and destination rectangles are equal, \( R=R_s=R_d \), with \( R = [x_{u},x_{nu}] \times [y_{nu},y_{mu}] \subset \mathbb{R}^2 \). To define the transformations we use a square \( S = [-1,1] \times [-1,1] \subset \mathbb{R}^2 \), hence, before the shape transformation the coordinates are normalized using

\[
\mathbf{n}(x) = \begin{pmatrix} \frac{x-x_{nu}}{s_x} -1, 2 \frac{y-y_{nu}}{s_y} -1 \end{pmatrix},
\]

and scaled back after the transformation using \( \mathbf{n}^{-1}(x) \). When undesirable distortions due to differences in width and height of the rectangle must be avoided, we use

\[
s_x = s_y = \min( x_{nu} - x_{mi}, y_{nu} - y_{mi} ), \text{ else }
\]

\[
s_x = x_{nu} - x_{mi} \text{ and } s_y = y_{nu} - y_{mi}
\]

are used. For each pixel \( x \) of the destination bitmap a color is calculated by sampling the source bitmap at pixel \( \mathbf{n}^{-1}(u=(\mathbf{n}(x))) \).

Pixels that are not defined by the transformation are kept transparent, such that the color of the parent rectangle remains visible.

Transformations can be used to create visualizations that look completely different as standard treemaps, but do not require the development of new layout algorithms. Also, the main advantage of treemaps is retained: both detail and overview remain visible. The base structure and semantics of the visualization are not changed, only the presentation differs.

Transformations should keep the density of the visualization uniform (UD) such that the area, and thus also the size, of nodes
remains comparable. If UD holds before a transformation, then UD should also hold after the transformation. For a transformation \( \phi : x \rightarrow u \) this is the case if
\[
\frac{\partial u}{\partial x} \frac{\partial v}{\partial y} - \frac{\partial u}{\partial y} \frac{\partial v}{\partial x} = C.
\]

For a composite transformation, UD holds if all base transformations maintain UD and the original, non-transformed, image has UD. We now present several shape transformations.

*Flip horizontal* flips the pixels of the source bitmap horizontally. This option is also known as mirror horizontal. The effect can often also be mimicked using other options, i.e., changing the sort method from descending to ascending, but it is often more convenient to use a transformation for quickly inspecting flipped images. *Flip vertical* is the vertical counterpart of flip horizontal.

*Pyramid* squeezes one side of the rectangle, thereby creating a pyramid. A simple transformation \( u(x) = (x(1-y)/2, y) \) violates UD. The top of the pyramid is squeezed more than the bottom of the rectangle; rectangles in the top become smaller than rectangles in the bottom (Figure 10a and b). A pyramid transformation that does satisfy UD is
\[
u(x) = (x\sqrt{2(1-y)} / 2, 1 - \sqrt{2(1-y)}),
\]
shown in Figure 10c.

Finally, a pie transformation twists the input rectangle around the center point, which creates a pie chart (Figure 10d). This transformation is given by
\[
u(x) = (r\sin \alpha, r\cos \alpha), \text{ with } \alpha = (x+1)\pi \text{ and } r = \sqrt{(y+1)/2}.
\]

For each level of a visualization a transformation can be used. This enables us to create composite transformations by stacking the base transformations, resulting in intriguing visualizations, such as shown in Figure 11. Here we use two pyramid transformations for the lower levels and a pie transformation for the highest level. We still have to find a use case for this, and such an image is not space filling. Nevertheless, it still has other benefits of regular treemaps: both detail and overview are present in one image, and the uniform density requirement is satisfied.

6 Results

We have developed a system for the visualization of business information, called MagnaView [18], in which we have embedded our extensions of the standard treemap algorithms. MagnaView offers a large number of options to influence the resulting visualization. For instance, per level the attribute to be used, the layout algorithm, possibly extra transformations, rendering of links, etc., can be specified. This flexibility comes at a price: It is not always easy to tune visualizations such that they meet the demands. We therefore have made a distinction between two types of users. Expert users design visualizations, or views on the data, and are enabled to modify every aspect of a visualization. End users are provided with a set of views on their data, and can easily browse through these, request detailed information, and zoom in and out.

Rather than a user experiment, we are currently performing a more challenging test: We are marketing our system commercially, with encouraging results so far. In this section we present examples of applications.

6.1 Notaries

In the Netherlands, a notary is a legal professional with academic training comparable to a lawyer, and not to be mistaken with the Anglo-Saxon notary public. The profession has changed considerably in recent years. Currently, tariffs are no longer fixed, offices merge, and there is a strong need to operate more commercial. However, notaries do not have a background in this. To increase insight and awareness in financial aspects, visualization is a useful tool.

We have visualized datasets from a dozen notary offices, varying between 3,000 to 10,000 files or cases per office. Each item represents a file or case, with as main attributes the sector (real

Fig. 12: Distribution of cases over sectors.

Fig. 13: Distribution of cases over employees.

Fig. 14: Distribution of cases over employees and sectors.
estate, family law, corporate law, etc.), the person who has handled
the case, the turnover, and the loss or profit. Figure 12 shows the
distribution of 2485 cases over different sectors for one office. The
size represents the turnover, color is used to indicate if a loss or
profit is made. It is clearly visible that there is a strong variation over
the different sectors. Real estate transactions are profitable, while
family law is not. Figure 13 shows the distribution of cases over
employees. It is clearly visible that Marty is not performing well.
There could be different reasons for this, for instance, Marty could
do only cases in family law. Figure 14 shows the distribution over
employees and sectors, and this shows that Marty also handles cases
in real estate, where he again is performing less than his colleagues.

Although merely an observation and not the result of an extensive
user test, we do note the following. The visualizations have been
used in a dozen notaries offices. Explanation of the visualizations
takes less than one minute, and leads to strong reactions among the
notaries. This is in particular true when compared to reactions to pies
or bar charts or mere presentations of statistics, containing
comparable aggregated information for instance that successions
result in a considerable loss. In terms of Van Wijk [19], these
visualizations have a certain value. The cost of understanding a mere
list of statistics is lower than the cost of understanding the business
treemap. Notaries gain considerable insight in their organization in
comparison to more simple presentations, hence the value of this
visualization.

6.2 High school

Dutch high schools typically have 1,000 to 2,000 students. Their
marks on tests are collected centrally. We have visualized a data base
with 200,000 of such marks. For each mark a large number of
attributes is available, obviously the score itself (ranging from 1 to
10 in the Netherlands, where 6 is sufficient and 10 is excellent), but
also the student, the class, the subject, the teacher, etc. Figure 15
shows an overview of the data base. The original image has a
resolution of 7,200 by 5,100 pixels. When printed out on a wall
poster of 122 by 86 cm both an overview and detailed information
are given.

From left to right the school years (1-6) are shown, and each
column contains a number of classes. Color is used to show the value
of the marks, red indicates an unsatisfactory result. From this view


previous years are available. These can be used to show their paths through the school. Furthermore, on the boundary the marks of the students are shown. Such a display can be used to study the relation between the path and the scores. A surprising conclusion from this display was the variety of possible paths, which was much larger than expected. There does not seem to be a strong influence on school career on the level of the scores. However, often outliers seem to have out of the ordinary school careers.

To construct this visualization, nearly all extensions presented so far have been used. The pie chart transformation enables us to easily see the size of fractions, and also, the boundary of a circle offers more space for visualization than the edge of a rectangle. To show the paths, a sequence of standard subdivisions, all in the same direction, were used, as well as added lines. For the marks, we used a squarified layout.

7 CONCLUSIONS

We have presented variations on standard treemap algorithms. We have shown how different layout algorithms for different levels can be used to take advantage of the strengths of each of them. Variations on the slice and dice algorithm and the squarified algorithm were presented, as well as a new layout algorithm, the matrix layout. Various choices can be made for the sizes of rectangles. Flexible margins can be used such that the uniform density property is satisfied. Via transformations triangular and circular treemaps can be produced.

All these options can be used in concert to produce visualizations that mimic familiar business graphics, such as histograms and pie charts, on a high level, while still providing insight and access to all individual items on the lowest level.

We have presented two applications of a commercially available tool which includes the techniques presented.

8 FUTURE WORK

We have shown that many variations are possible on the basic treemap algorithms. We are not convinced that we have touched the bottom yet. We expect for instance that other useful transformations can be defined, for instance along spirals or for mapping on geographic maps. Another area where work can be done is the user interface. We have described how we have split tasks between expert users and end users. Not all combinations of settings lead to useful results. We have found some rules of thumb (see for instance Section 3.1), we are currently deriving more of these and studying how these can be offered to the user. One option is to use automatic selection of parameters, another one is via the use of templates.

REFERENCES


