Technische Universität Wien

Vienna University of Technology

DIPLOMARBEIT

Set Type Enabled Information Visualization

Ausgeführt am VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH

unter der Leitung von
Priv.-Doz. Dipl.-Ing. Dr.techn. Helwig Hauser

und der Mitbetreuung von
Dipl.-Ing. Dr.techn. Krešimir Matković

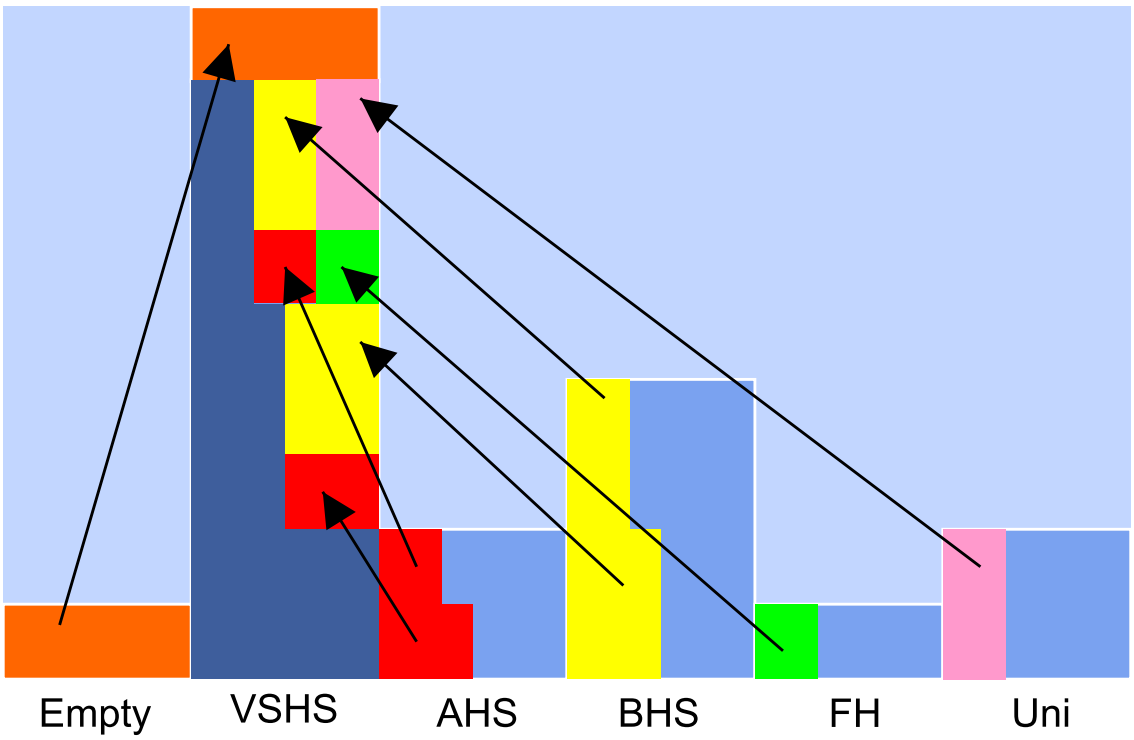
eingereicht an der Technischen Universität Wien
Fakultät für Informatik

von
Wolfgang Freiler
7053 Hornstein, Dr. L. Leser Straße 17
Matrikelnummer: 0026034
email: freiler@vrvis.at

Wolfgang Freiler

Set Type Enabled Information Visualization

DIPLOMA THESIS



email: freiler@vrvis.at
http://www.vrvis.at/vis/resources/DA-WFreiler/

Abstract

Information Visualization is a research area in the field of computer graphics that deals with visual representations of abstract and usually multidimensional data. This data can origin from questionnaires, elections, measurements or simulations. Apart from specialized tools, that are made for a special purpose, there are general purpose tools, that can be used to analyze many kinds of different data.
These tools are made to handle different data types, like numeric or categorical values, some also support more advanced data types, like time series data or hierarchical data. In this document, the data type *set* will be introduced into the general purpose visualization tool *ComVis*. A set is a collection of multiple elements, that can also be empty. In many cases, a dimension with the data type set can replace multiple categorical dimensions and make data analysis and exploration more efficient and complex datasets easier to understand.
This work will not only explain, how to use sets to explore datasets, but also introduce a new specialized view based on a histogram view, that is dedicated to the use of sets. Of course, most of the already existing views have been modified to use sets, otherwise the newly added data type would be difficult to use either. Especially views that can display multiple dimensions were a challenge, because they allow the user to mix sets with other data types. Apart from the use of sets in various views, some additional topics are covered in this document. The conversion of existing categorical data is a very important feature, as well as a fast and efficient data structure. The existing methods for user interaction like brushing and linked coordinated views have to work as expected for all supported data types.
A set should not be seen as a new artificial data type, that we have to convert existing data to, but as the natural data type in many applications. Instead of introducing another conversion step for our data, we can avoid converting data with multiple related attributes to a range of categorical dimensions. Using sets is also an efficient way of dimension reduction, and can reduce the complexity of a dataset, as well as the amount of views needed for exploration.
Additionally, there are some examples on how to take advantage of sets when analyzing a real-world dataset. Some special features of this dataset as well as some erroneous entries are easier to find by using sets and views that support them.

Zusammenfassung

Informationsvisualisierung ist ein Forschungsgebiet der Computergraphik, das sich mit der visuellen Repräsentation von abstrakten und meist multidimensionalen Daten beschäftigt. Diese Daten können aus Befragungen, Wahlen, Messungen oder Simulationen stammen. Neben spezialisierten Programmen, die für einen bestimmten Anwendungszweck geschaffen wurden, gibt es universelle Werkzeuge, die für die Analyse von vielen verschiedenen Arten von Daten geeignet sind.
Diese Programme können verschiedene Datentypen, wie numerische oder kategorische Daten verarbeiten, manche unterstützen auch weitere Datentypen, wie Zeitserien oder hierarchische Daten. In diesem Dokument wird das universelle Visualisierungsprogramm *ComVis* um den Datentyp *Set (Menge)* erweitert. Eine Menge ist eine Sammlung von mehreren Elementen, die auch leer sein kann. In vielen Fällen kann eine Dimension mit dem Datentyp Menge mehrere kategorische Dimensionen ersetzen und die Untersuchung und Analyse von Daten effizienter und verständlicher machen.
Diese Arbeit wird nicht nur erklären, wie Mengen zur Untersuchung von Daten genutzt werden können, sondern auch eine neue auf einem Histogram basierende Visualisierungstechnik, die speziell für Mengen gedacht ist, erläutern. Natürlich müssen auch die bereits vorhandenen Visualisierungen für die Darstellung von Mengen modifiziert werden, da dessen Nutzung sonst sehr eingeschränkt wäre. Besonders die Visualisierungen, die mehrere verschiedene Dimensionen darstellen können, sind eine Herausforderung, da hier die bestehenden Datentypen mit Mengen gemischt werden können. Abgesehen von der Visualisierung von Mengen beschäftigt sich dieses Dokument mit einigen verwandten Themen. Die Konvertierung von bestehenden kategorischen Daten ist ein wichtiges Thema, genauso wie eine schnelle und effiziente Datenstruktur zu Speicherung. Die existierenden Interaktionsmethoden, wie *Brushing* oder *Linked Coordinated Views* müssen natürlich für alle Datentypen einwandfrei funktionieren.
Eine Menge sollte allerdings nicht als ein künstlicher Datentyp angesehen werden, der zusätzlichen Aufwand durch Konvertierungen erfordert, sondern als den für viele Anwendungen natürlichen Datentyp. Anstatt eine Konvertierungsstufe für Daten einzuführen, kann die Verwendung von Mengen Konvertierungen von Daten, die mehrere verwandte Attribute enthalten, ersparen. Die Verwendung von Mengen ist eine effiziente Möglichkeit, um die Anzahl der Dimensionen und der zur Darstellung notwendigen Fenster zu reduzieren, was die Analyse von Datensätzen sehr vereinfachen kann.
Außerdem werden in dieser Arbeit einige Beispiele zur Untersuchung von Daten mit der Hilfe von Mengen erläutert. Einige Details dieses Datensatzes lassen sich mit Mengen und den neuen Visualisierungsmethoden sehr einfach aufspüren. Auch einige fehlerhafte Datensätze konnten hiermit entdeckt werden.

Contents

Chapter 1

Introduction

In the last years, the amount of data collected and stored on computers has increased dramatically. These data sources include simulations, measurements, surveys, census data and network traffic, but of course there are lots of other cases, also, where huge amounts of data are collected. This data can contain valuable information for engineering, marketing or security, but finding interesting and useful details in a dataset can be a tedious task. While in some cases, global trends are important, in other cases, such as security, only a very small part of the data is of high interest.
A lot of statistical methods have been developed to assist users when examining data. Along with simple valuers like mean values and variance, correlation and regression analysis can help to find out more about a dataset. However, it is often helpful, to visualize the dataset to make data analysis easier. Some techniques like histograms and scatterplots have their roots in statistics and have become very popular to give users a quick overview on the distribution of a dataset.
However, this work does not deal with statistics, but with Information Visualization. It concentrates on drawing visual representations of data, to support analysing data. In the following chapters, the data type *set* and its use in information visualization will be discussed. This data type corresponds to a mathematical set with some simplifications (no multi sets and no hierarchies) and has been implemented in the *ComVis* Visualization System. An efficient class for storage had to be implemented, as well as a special view for sets. Additionally, most other views were modified to allow the display of sets.
This document will give a detailed description of the technical background of sets along with instructions on how to convert multiple categorical dimensions of existing datasets to use the new data type. There are also detailed descriptions of the modifications to existing visualization methods in ComVis and the new *Profile Histogram*, an extended histogram exclusively dedicated to sets. After an exemplary examination of a real world dataset, various details of the implementation of sets in ComVis will be explained.

1.1 Visualization

The field of Visualization deals with the transformation of raw data into visual representations, that are helpful for the analysis or the presentation of the data. Some of the visual representations, for instance histograms, have their roots in statistics, but they have been improved and refined to allow a better insight into the data, and more effective forms of

interaction. Especially the latter has become more and more important in current visualization applications. The two main research areas in the field of visualization are *Scientific Visualization* and *Information Visualization*.

1.1.1 Scientific Visualization

Scientific visualization deals with simulations or measurements of real world phenomena. Besides the two large research areas *volume visualization* and *flow visualization* there are are lots of other situations, where scientific visualization can help to get a better insight into complex datasets. The visualization of flows in hollow objects or around vehicles are classic fields of use for flow visualization. Data normally consists of coordinates of measurement points and flow vectors of either two or three dimensions. Additional parameters like pressure and temperature are of course also possible. Volume visualization deals with visualizing interesting parts of volumes. The focus lies on finding these interesting details and make them visible, even if they lie completely inside the volume. A popular example is the visualization of computer-tomographic data based on ray-casting techniques like the one by Levoy [35]. Visualization helps to make interesting parts of the human body visible, and helps to find harmful tumors. Datasets used in volume visualization consist of a grid of density coefficients with optional other parameters.

1.1.2 Information Visualization

The main difference between scientific visualization and information visualization is, that there is no inherent spatial structure. Data used in information visualization can be manifold. From one-dimensional to multi-dimensional tabular data, also trees, networks or text can be visualized [25]. Several information visualization applications have already been released. While some are mainly used in scientific areas, others are also used for business purposes. To make the development of new applications easier, Fekete implemented the InfoVis Toolkit [14]. This library provides many state of the art techniques for information visualization, which can significantly reduce development time. The InfoVis Toolkit offers data structures, visualization algorithms, filtering methods and even allows deformations of the resulting visualization. Heer et al. [17] have collected various design patterns that have been used in the development of visualization software. These patterns can help to reduce development time of software, and make the resulting codebase easier to understand for other developers. Heer lists twelve design patterns, that are frequently used in visualization, like data columns, tables, relational graphs and schedulers. An analysis on the history of information visualization and the most active scientists in this research area can be found in "Exploring and Visualizing the History of InfoVis" by Keim et al. [26].

1.1.3 An Alternative Taxonomy

Tory et al. [48, 49] have proposed a different taxonomy of visualization. While data in scientific visualization, which includes flow and volume visualization, is not necessarily scientific, information visualization can of course be used to visualize scientific data. Instead of qualifying data as scientific, we can differentiate between discrete and continuous visualization. While continuous data can be interpolated, discrete data can not (often). Scientific visualization uses mostly continuous measurement or simulation data, that can of course be interpolated, so these methods would be classified as continuous visualization.

Information visualization uses many different kinds of data, and while we can of course visualized continuous data in information visualization, there is a much larger area of discrete model visualization, than in scientific visualization. Interpolating categorical data, trees and networks is not possible, just as the new data type *Set* introduced later in this document. Hence, techniques introduced in this work can be qualified as discrete model visualization.
Tory et al. [48, 49] also mentioned a new kind of classification scheme, originally proposed by Tamara Munzner [42]. According to her, visualization can be divided into two categories, those with a given, and the ones with chosen spatialization. Scientific visualization has a given spatialization in most cases, whereas information visualization has not. However, both of these taxonomies have no clearly defined border, and of course, they are not just a new label for the traditional taxonomy of scientific and information visualization.

Chapter 2

State of the Art

This document deals with a specific topic in the field of information visualization: the introduction of the data type *set* and some possible visual representations. Although there have been visual methods representing information for a long time, for example in statistics, information visualization as a scientific branch is considered as fairly young. Van Wijk [50] sets the year of the birth of visualization to 1987, when the US National Science Foundation (NSF) published the ViSC report [36], which encouraged investments in visualization. Now, in 2007 many new techniques and use-cases for visualization, and for information visualization in particular, have been published, some have become very popular and made it into widely used visualization packages.
The following chapters will illustrate some important information visualization techniques and use-cases with a focus on types of visualization that can be extended for the use of sets.

2.1 ComVis

The ComVis [1] visualization system is a general purpose visualization tool developed at the VRVis Research Center in Vienna, Austria. This application has been designed to visualize multivariate datasets using different visualization techniques (see figure 2.1). ComVis also supports *linking and brushing* (see section 2.3.1 for details) using multiple brushes that can be combined with different operators. ComVis already supports the visualization of numeric, categorical and time series data as proposed by Konyha et al. [29]. This document is focused on extending ComVis to support set-typed data, which includes adapting existing visualization techniques, as well as adding support for set-typed data in the application's backend.

2.2 Popular Examples of Visualization

Visualization applications can have different target audiences, some can be useful for the average computer user, while others are useful for scientific users. BiblioViz by Shen et al. [44] is a good example for the latter. By visualizing publications and their authors of the visualization community, they create images, that are interesting for people that actually read visualization papers. This tool could be interesting for scientists in other areas too, but the audience still remains rather small. On the other hand, a very popular example

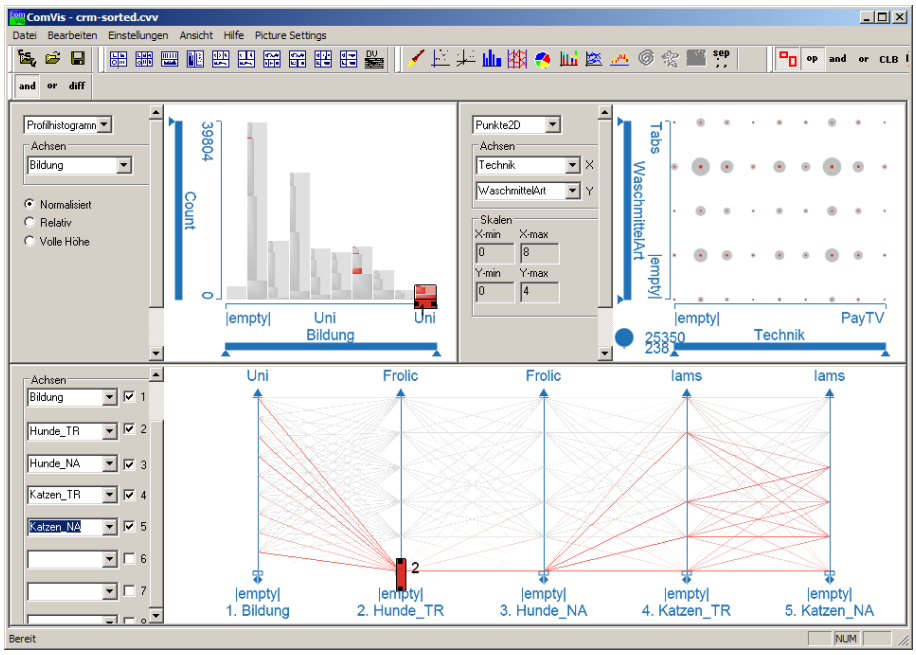


Figure 2.1: A typical ComVis workspace contains multiple views displaying different dimensions of a dataset using various visualization techniques.

of information visualisation has been published by Wattenberg: The *Baby Name Wizard's Name Voyager* [55]. This tool displays information on names given to newborns from 1880 up to now. The view consists of stacked graphs, where each bar represents a specific name over the last decades. The thickness of the bar at a specific decade represents the number of children that have been given that name. The Applet also allows entering a prefix for a name and filters the results in realtime, which makes exploring the dataset a very easy and interesting task. Although the visualization techniques used in this Applet are rather simple, it reached a larger audience of users and it is well known for its ease of use and the comprehensible representation of the results. Due to the success of the Name Voyager, Wattenberg and Kriss published an analysis [54] on that project, trying to find reasons for its popularity among average users outside the visualization community. They collected user opinions from bulletin boards and weblogs and came to the conclusions that there are different types of users. Apart from the intended audience – people who are looking for names for their babies – there were many people, who just liked inspecting the dataset, checking names of their own, of friends, relatives and celebrities. Another reason for the success of this applet was a dataset which is interesting but easy to understand.
Another popular example for information visualization in everyday's life is SequoiaView [22] by Jarke J. van Wijk. The visualization technique used in this program is called *treemap* and has been introduced by Johnson and Shneiderman [24]. Treemaps are used for the display of hierarchical data in a single space-filling view by recursive subdividing. That is why treemaps work best, if every leaf in the hierarchy has a size, and each node's size is the sum of his leaves' and nodes' sizes. The size of an item, or a group of items can then be estimated by the amount of screenspace it takes up. In the case of directory structures on hard disk drives, these prerequisites are met, so this is the ideal use case for

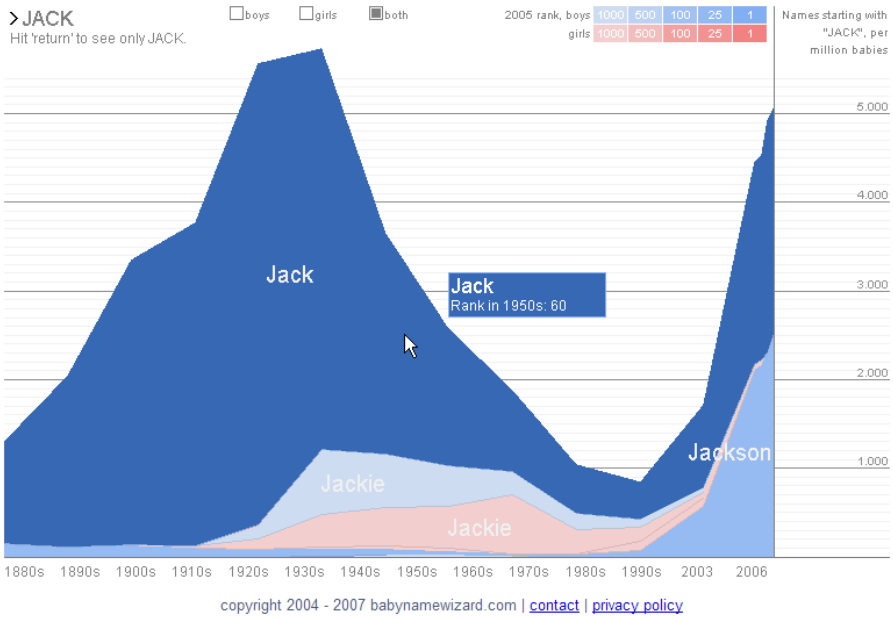


Figure 2.2: *The Baby Name Wizard's Name Voyager* is one of the best examples for information visualization reaching a wide non-scientific audience. Data exploration seems to be a good opportunity to satisfy one's curiosity. (screenshot taken from the website [55])

treemaps. Van Wijk improves treemaps by addressing two problems [52]:
He *squarifies* the subdivided rectangles, which avoids long and thin rectangles. Rectangles with aspect ratios near one are better comparable, and easier to point at. He also introduces cushion shading to emphasize structure and save screen space. Every rectangle is filled with a radial gradient and dividing lines between rectangles are omitted. The brightness of the gradient gives information about the hierarchical steps. Sudden changes in brightness indicate the border between two rectangles (see figure 2.3). According to van Wijk, SequoiaView has been downloaded 500000 times and has been distributed by the computer magazine c't. The reasons for this success are a huge number of potential users and an intuitive visualization, which can be understood by most users without consulting help files. Accompanied by a simple user interface, this program can be used by the majority of computer users. Some variations to treemaps exist, such as the facetmap by Smith et al. [46]. The facetmap uses hierarchical ovals and rounded rectangles to provide a user friendly user interface for browsing files on a computer. These files can be filtered by various criteria like the file's type, its creation date or the author.
Another well-known information visualization application is spotfire [5] by Ahlberg [7]. Spotfire supports many different visualization and interaction techniques. It can visualize data containing geographic information using background maps and supports focus and context techniques (see section 2.3.3) like *details on demand*, which displays more details about the currently selected dataset. Some advanced user interface widgets, such as rangesliders or alphasliders by Ahlberg et al. [8] have also been implemented in Spotfire. As another feature, Spotfire can access databases to load data for visualization. Spotfire has been released as a commercial application, and, because of its success, it is now marketed and developed further by the business software company TIBCO [6].

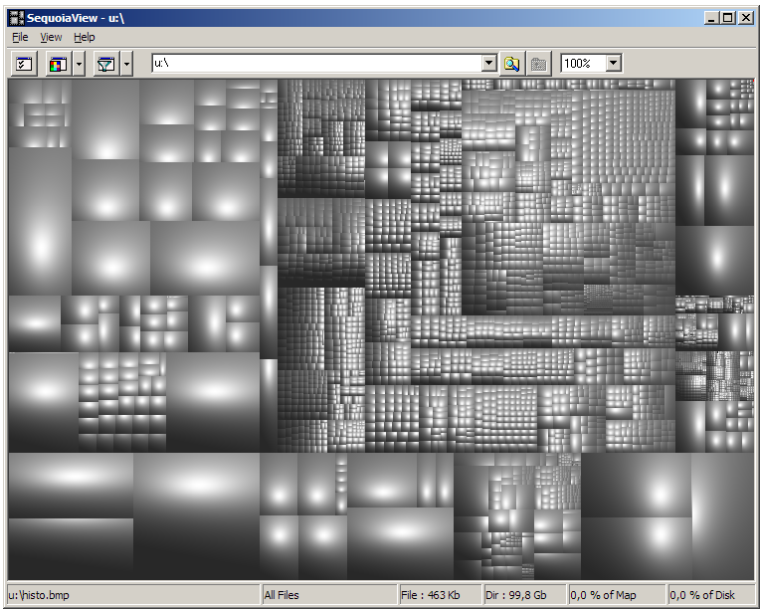


Figure 2.3: *SequoiaView* displays a treemap of the contents of a directory structure. This popular tool can be very useful when tidying up a hard drive - a very common problem for computer users (screenshot taken from the program SequoiaView [22]).

Many Eyes [3] by IBM is a web based information visualization platform allowing users to upload datasets and create visualizations, that can be interactively explored, rated and commented by other users. With every comment, a snapshot of the currently displayed image can be attached. By clicking this image, other people can continue to explore where the commenting user stopped. Many Eyes supports numerous visualization techniques, multiple coordinated views are not possible. However, this keeps visualizations easy to understand for average users.
Gapminder [2] by Hans Rosling is an online visual analysis tool that visualizes statistics on global development. The user can choose between different data dimensions, such as child mortality or economic growth, and can compare all countries (where data is available) in a scatterplot view or on a world map. A third dimension can be chosen to be represented by the glyphs' sizes, while colors are used to represent continents. Because global development is a time-dependent process, the user can select a year to show the data from. It is also possible to view an animation starting in 1975 to get an impression on which countries have developed, and where the living conditions have worsened.

2.3 Common Visualization Techniques

Throughout the history of information visualization, many techniques for different tasks have been published. As mentioned before, some have become very popular, while others are rarely used. Various taxonomies classifying data types in information visualization exist. Shneiderman [45] proposed a type by task taxonomy (TTT) in 1996, and mentioned seven data types:

1. One-Dimensional Data

2. Two-Dimensional Data
3. Three-Dimensional Data
4. Temporal Data
5. Multi-dimensional Data
6. Trees
7. Networks

Shneiderman explains the main characteristics of each data type and lists some publications dealing with the visualization of these data types. In 2002, Daniel Keim [25] gave a similar overview on existing display techniques for six different data types:

1. One-Dimensional Data
2. Two-Dimensional Data
3. Multi-dimensional Data
4. Text and Hypertext
5. Hierarchies and Graphs
6. Algorithms and Software

While visualizing one- and two-dimensional data is in most cases a relatively simple task, which is handled by histograms and scatterplots, for example, and multi-dimensional data is handled well by parallel coordinates (see the next chapter for details), the other datatypes deserve a more detailed description here. Text as a data type in visualization can be mapped to numeric dimensions, and displayed in views that were actually made for numeric data. However, this applies to short strings, that occur often, like *categories* in ComVis. Visualizing a page of text or hypertext is a different task, and needs specialized techniques.
The data type *Algorithm and Software* is a good example for missing interdisciplinary connections to other scientific branches. Most people involved in Information Visualization are familiar with software development processes, hence there are already various approaches to this problem. Other possible fields of use may have remained untouched by visualization, simply because people are not not aware of the possibilities there.

2.3.1 Brushing and Linking

Keim also mentions important interaction techniques, which play an important role in Information Visualization and should not be neglected. One of the most important tools in data exploration is *brushing and linking* or *focusing and linking*. As illustrated by Buja et al. [12], data exploration can be made more comprehensible, if the user can easily select a range of data items in one view, and emphasize the corresponding data items in all other views. This technique has been implemented in many applications up to now. Many tools including ComVis allow brushing and linking, for example the Scalable Framework by Kreuseler et al. [32] and XMDVTool by Matthew Ward [53]. The user can select ranges

on various dimensions directly in a visualization and the selected data items are drawn in a different color. These data items are also drawn as selected in all other views on the screen, which is called linking. To further refine the search, a brush can be combined with other brushes to refine a search. By localizing interesting ranges on multiple dimensions, the number of selected data items can be reduced, until only a small but interesting subset is selected. This is the point, when the user might want to inspect the raw values in the dataset. Tables displaying the selected data records and export functionality should be implemented to let users save their results.

2.3.2 Zooming & Panning

Zooming and Panning are widely used techniques in visualization, which are also used in many other applications, such as map-tools, and office applications. Zooming allows the user to view a small part of a visualization resized and with great detail. This can be done by dragging a rectangle into the view and enlarge the selected area. ComVis lets the user select visible areas by resizing a bar called *ScaleSlider* in the axis of a view. To move the resized region to another place in the view, the user can move the scaleslider. This feature is called panning and should be implemented in every application, that allows zooming. Van Wijk [51] proposed a technique for smooth and efficient zooming and panning, which allows rendering a smooth transition from one view to another. Applications that use such a combination of zooming an panning mostly have very large areas that can be explored by the user. These areas can be found in visualization views as well as in map applications. Zooming however is not restricted to two-dimensional views. Rao et al. [39] have shown a a method to display a compressed view of a table with details on demand. This *TableLens* allows the display of a large amount of tabular data by rendering previews of one pixel height. Numeric values are approximated by a small horizontal bar, categories can be color-coded. When a table row is enlarged, it shows textual representations of the values.

2.3.3 Lenses & Distortion

Lenses and distortion are other frequently used techniques to support the user when analyzing a dataset. Magic Lenses, as described by Bier et al. [10], are virtual looking glasses, that can be dragged directly into a visualization, and modify the way visual elements are drawn. Magic Lenses can be used to toggle visual elements, or emphasize details with different colors. Some applications allow multiple Magic Lenses, and allow combining them by overlapping.
When displaying large amounts of data, that do not fit into the display area, zooming and panning is often used to enlarge selected parts. However, in such cases only the enlarged area is drawn, and the user can lose the context to neighbouring data elements. To solve this problem, *Focus and Context* solutions have to be applied. A good example for such a technique is the fisheye view [16]. The focus, consisting of the most important elements in the view, is magnified, whereas the surrounding, less important elements are drawn only very small or with less detail. These less important areas form the context and can assist the user in navigating through the dataset.
To further emphasize focused visual elements, they can be drawn larger or in a more detailed way. Lamping et al. [33] developed the hyperbolic tree browser, that arranged all nodes on a hyperbolic plane. The user can drag interesting nodes into the center of the

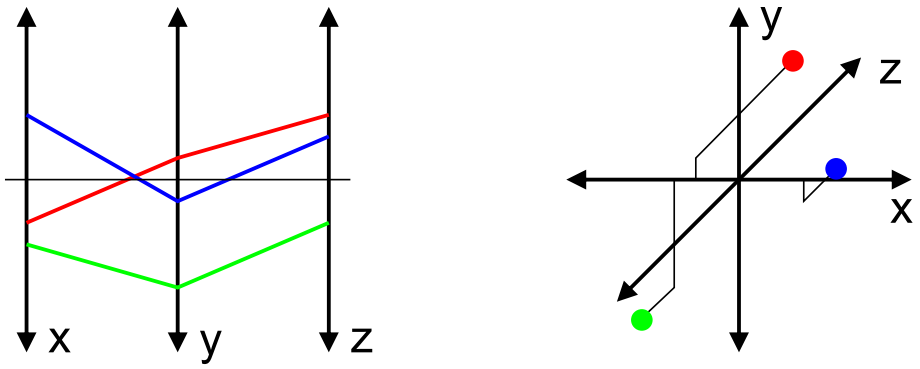


Figure 2.4: Each line in the parallel coordinates view on the left corresponds to the point with the same color in the orthogonal view on the right. The thin lines show the way on the y- and z- axis, before the point is plotted. Otherwise it can be hard to recognize the amount of the z-Value in a two-dimensional still image.

view, where they are automatically displayed larger.
Kosara et al. [31] proposed the use of a depth of field effect for regions of interest. This effect occurs in photography, where only objects at a specified distance to the camera are displayed sharply. The human visual system automatically concentrates on sharp objects, and neglects blurred ones. This effect can be used to emphasize important objects in information visualization by applying a blur filter to less important objects.

2.3.4 Parallel Coordinates

Inselberg, the pioneer of *Parallel Coordinates* [21], has released many publications on this topic. In an attempt to visualize high-dimensional structures to be easier to understand, he created one of the most popular techniques in information visualization. When using orthogonal axes to display an item in n-dimensional space, higher dimensions than three are hard to visualize and unterstand. We are used to three dimensional space, because this is the space we live in. In data analysis, datasets can have significantly more dimensions, so it is much more convenient to arrange them in a parallel way (see figure 2.4). In parallel coordinates, a data item is no longer represented by a point, but by a series of connected line segments. This line crosses each dimension at a specified position. This visualization technique may seem uncommon at first, but comparing values from different axes is a very easy task, and even with only three dimensions, it can be much easier to read, than a view with orthogonal axes. However, parallel coordinates also have drawbacks. When looking for correlations between two parallel dimensions, it is important to place them next to each other. That's why parallel coordinates need a good user interface, which allows changing the order of axes, and inserting duplicate axes for better comparison. Another problem is the significant amount of screenspace used by each data record. Because every dataset is represented by a polyline, the screen becomes cluttered very quickly (see figure 2.5). Various techniques address this problem by introducing adaptive transparency based on the number of lines, or clustering techniques. Johansson et al. [23] addressed this problem by using non-linear transfer functions for parallel coordinates. The transfer function defines, how the intensity of a pixel increases with the amount of lines passing through it. This can be used to visualize outliers or very dense regions. Another approach

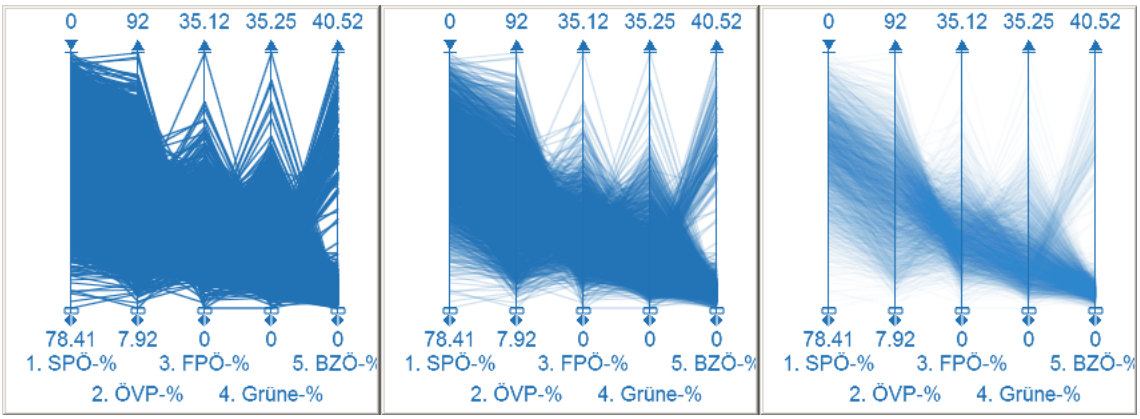


Figure 2.5: Parallel Coordinates are very susceptible to cluttering. These three views show the same dataset containing 2380 data records. The first view uses no transparency, which makes it hard to estimate the density of the lines. The second view uses some transparency to reveal some structure of the dataset. The third view uses a very high transparency, so we can analyze the dense parts of the graph. However, single lines are hardly visible in the third view.

to reduce cluttering in parallel coordinates has been proposed by Novotný et al. [37]. By treating outliers and trends differently, the number of visual elements drawn is reduced significantly. While outliers are drawn as single polylines, trends are represented by wide bars, that represent multiple similar lines. In large datasets, this approach can also reduce the rendering time.
Parallel Axes are best suited for numeric dimensions, because categorical values can be difficult to sort. Finding the correct place or region on an axis can be hard too. Kosara et al. adress this problem with *Parallel Sets* [30]. Overdraw in a parallel coordinates view can also be reduced by clustering. Data records with similar properties in many dimensions are collected and assigned to a group of records. Each one of these groups can then be rendered by drawing a polygon representing all lines in the group. The disadvantage of that technique is, that in some dimensions these polygons overlap and make analysis more difficult. Finding a good classification of the data to reduce overlapping is a tough challenge too.
To reduce the amount of lines in parallel coordinates, Inselberg [20] proposed using envelopes for clusters of related data records. These bars span from the lowest to the highest value at each axis for all data items in a cluster. Andrienko et al. [9] improved this technique, by painting multiple bands, to visualize gaps between outliers and the dense center of the band.

2.3.5 Scatterplot

The scatterplot is a simple and useful technique in information visualization. It is a good view to examine correlations between two numeric dimensions. A scatterplot consists of a Cartesian grid, ranging from the lowest to the highest values of each dimension. A data item is represented by a point, that is placed at the corresponding coordinates. If multiple points align in the neighbourhood of a line, the two dimensions are likely to correlate.

Scatterplots are part of many general purpose visualization system, because many people have already experience with them. They are also very useful for creating two-dimensional brushes. Chambers et al. [13] have shown a scatterplot matrix which can be used to analyze all possible scatterplots in the dataset. Seo et al. [43] have proposed a rank-by-feature framework, that provides scores for all possible scatterplots in a dataset. The user can choose a type of score (e.g. the correlation coefficient) to be color coded into the cells of the matrix, by clicking the cell, a scatterplot can be opened for further analysis. This method allows quick access to possibly interesting combinations of dimensions.
Büring et al. [11] have extended scatterplots to be comfortable to use on handheld computers. They implemented geometric-semantic zoom to switch between various levels of details. This zoom method provides textual details and even images when zooming on a small group of data items, and reduces details, if the user decides to zoom out. Additionally, the application provides fisheye distortion, to see few items at great details, while other items are shown as dots in the context area. Methods like these are not confined to small screens, saving screenspace is always an important topics. Large screens can show more different views simultaneously, if they can be kept small without being confusing.
Scatterplots can also be three-dimensional. Piringer et al. [38] extended 2D and 3D scatterplots with some useful features to provide focus and context visualizations. A technique called depth cueing (displaying points that further away smaller) has been implemented as well as halos. These are dark borders around points, that make it easier to recognize single points. Probably the most important feature in 3D scatterplots is user interaction. Due to the occlusion of points, it is necessary to rotate the coordinate system to get a good view on the dataset.

2.3.6 Visualization of High Dimensional Data

Besides parallel coordinates and scatterplot matrices, there are various methods to display more than two dimensions in a two-dimensional plane.
LeBlanc et al. [34] proposed a technique to explore n-dimensional data in 2D-space. By nesting dimensions hierarchically, more than two dimensions can be visualized in 2D-space. If the cardinality of two dimensions is smaller than the screen width and height, each value is represented by a rectangle. This rectangle can then contain a nested view displaying two other dimensions. This approach can be repeated to display even more dimensions, but the amount of available screen space limits the number of possible hierarchies. Another problem is, that the order in which the dimensions are used has a great impact on the resulting image. Some features of the underlying dataset may not be visible in all hierarchies, while they are obvious in others.
Hsu [19] developed a Generalizing Self-Organizing Map for Categorical Data. Self-organizing maps are Neural Networks, that project high-dimensional structures onto a low-dimensional grid. The goal of these maps is to place similar data records next to each other, while the ones that are different in many dimensions should be further away. Self-organizing maps have already been used for engineering and business data, but they had a weak performance when applied on categorical data. Hsu proposed a tree for hierarchical grouping on available categories, to have a more reliable distance function. Categories having the same parent are more closely related than categories that are far away from each other.

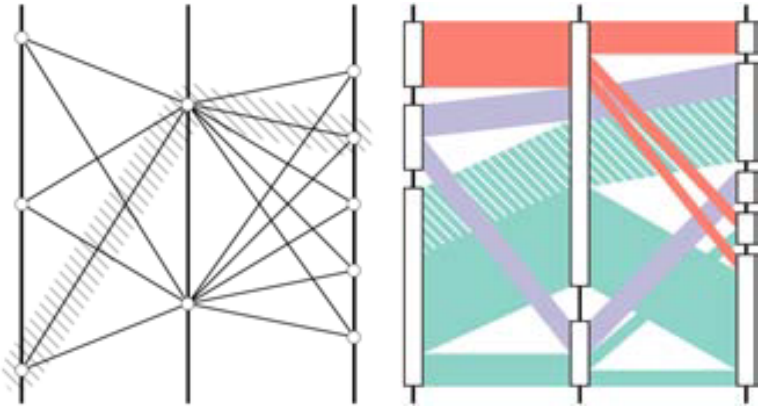


Figure 2.6: Because parallel coordinates (left) show a line for each data record, the categories' sizes cannot be estimated. In parallel sets, the amount of data records in each category comes apparent. By using colors, it is easier to follow the bars after splitting up. (image courtesy of Kosara et al. [30])

2.4 Visualization of Categorical Data

Whenever categorical values should be displayed on a numeric axis, it is necessary to map categories to numeric values. This can be hard, because many categorical representations can not be sorted, or must be sorted by hand. For example, names of cities can be arranged alphabetically, which can happen automatically. Educational levels should be sorted by hand from the lowest to the highest level. Another problem is the distribution of categories on axes. Most types of visualization distribute all categories on an axis with equal distances in between. Because in most cases, there is no meaningful measurement of a *distance* between categories, the placement can be used to reflect the amount of data records in each category.
Kosara et al. [30] adress this problem with *Parallel Sets*, by combining advantages of parallel coordinates and mosaic plots (Theus et al. [47] and Hofmann et al. [18]) Their implementation supports categorical as well as numeric values, and maps categories to ranges on horizontal axes. These ranges are visualized as boxes containing the name of the category. A range's size is defined by the number of data records it represents. Connections between axes are established by bars, that are as wide as the originating box. These bars can split up and distribute to the boxes on the next axis. By comparing the widths of the bars, it is very easy to estimate the amount of data records represented by them. With some experience in arranging axes and categories right, it is possible to create visualizations that are very easy to analyze (see figure 2.6). Keim et al. [28] have been working on categorical data for some time and have published various papers on *Pixel Bar Charts* (see figure 2.7). Pixel Bar Charts are histograms that reveal details about the data represented by a bar. A bar in a Pixel Bar Chart consists of one pixel per data item. Some information about each single data item is coded into the color of the pixel. This type of visualization is very effective to display one numeric and one categorical dimension of a dataset. The bars' sizes are determined by the number of data items in each category, the bars themselves are filled with differently colored pixels for each value in the numeric dimension. To increase the amount of screenspace available for the visualization, it is

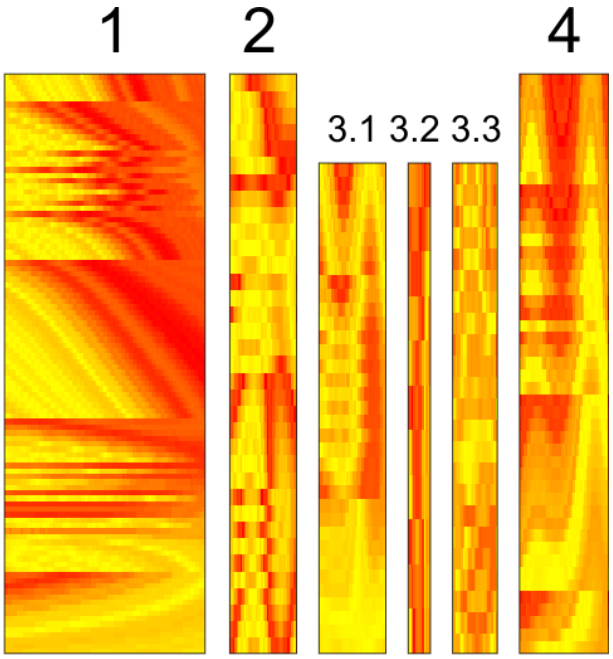


Figure 2.7: Sketch of a Pixel Bar Chart. These are extended histograms where each pixel of a bar represents a data item. The size of a category is represented by a bar's width instead of its height. This improves the use of screen space, and allows displaying hierarchies by using lower bars.

possible to use the bars' widths instead of their heights as an indicator for the size of the category. In 2002, Keim also presented a hierarchical version of pixel bar chars [27]. Bars can be split into sub-bars that are rendered a bit shorter to reflect the hierarchy. With deep hierarchies this solution may take up a lot of screen space, but flat hierarchies can be displayed very well along with the additional information provided by the pixels' colors. Using the area of histogram bars to display further information is a technique picked up later in this document for the *Profile Histogram*.

Chapter 3

The Integration of Sets

In the field of information visualization, the variety of applications is overhelming. However, many datasets have in common, that they are organized in a tabular manner. Every line contains a data record consisting of various attributes in different dimensions. Most datasets use numeric and categorical data, some applications also support more advanced data types.
In the following chapters, the data type "Set", which is new to information visualization, will be illustrated. Support for sets has been implemented in the ComVis visualization system, a general purpose visualization tool, which is suitable for many different areas of data exploration and analysis.

3.1 The Problem

This Chapter deals with the idea that led to the implementation of sets. The first idea of an implementation of sets came up while analyzing a CRM Dataset containing information from a questionnaire filled out by about 65000 people. This dataset contains data about the education and shopping habits as well as income and pets. Along with some numeric and categoric dimensions there are many boolean dimensions describing educational institutions each person has graduated from, or products the person has frequently purchased. This leads to a high number of columns containing only the values "1" and "0". Visualization of all educational institutions was only possible by using the parallel coordinates view, because no other view supports such a high number of dimensions. Analyzing data with this view is very difficult, because visualization results in a grid with diagonal intersections between "0" and "1". Given a high number of data records, most visualizations of high dimensional boolean data will give a similar result. In many questionnaires, only the last educational institution is queried, either because others are not important, or due to simple analysis of the data. However, when querying frequently visited shops or purchased products, it is not a feasible solution to only ask for the most important one, since there is a significant loss of information.
At this point the idea of the implementation of sets has started. A set dimension should store information of multiple categorical dimensions to make the use of low-dimensional visualization techniques like histograms and scatterplots possible. No information should be lost, therefore multiple elements have to be allowed for each data record.

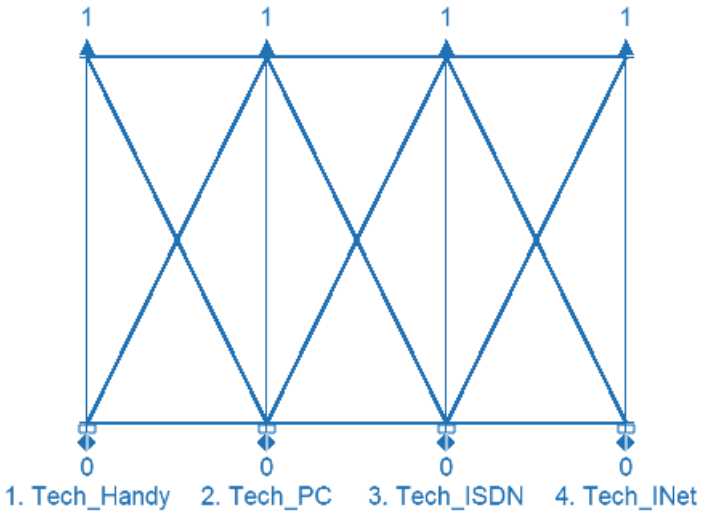


Figure 3.1: A parallel coordinates display of 4 binary dimensions. There are only four possible lines between two axes, and in cases with many data records, all of them are used. The resulting visualization is useless.

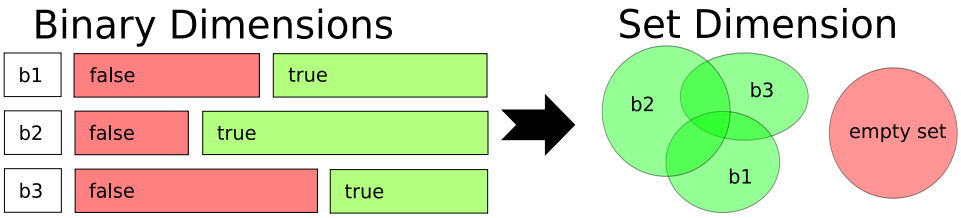


Figure 3.2: A single set dimension can be seen as multiple related binary columns Each Data record can be part of the empty set, or have one or more elements assigned to itself. Without sets, one binary dimension per element would be needed.

3.2 Sets - The Basic Idea

Sets in information visualization are very similar to sets in mathematics. A set is a container, that contains various elements, in our case a finite amount of textual attributes. A set $S1$, that only contains elements, that are also part of another set $S2$ is called a subset of $S2$. In ComVis, sets have been implemented as collections of strings. Every string, that can be used as a category, can also be used as an element in the set. A set dimension is defined by a main set, that contains all valid elements. All values in this dimension have to be subsets of the main set.
To represent the amount of information stored in a set dimension, one binary column per element would be necessary (see figure 3.2). Definitions for the terms ”set” and ”element” follow this paragraph.

Element An element is the smallest part of a set. In a set-domain, values can be empty or consist of up to all available elements. An element is analogous to a single category

in categorical dimensions, and is therefore defined by its name. Other types of elements, such as numbers, have not been implemented.

Set A set is a collection of elements. A valid value in the domain of a set is either an empty set, or a collection of an arbitrary number of available elements. In the current implementation of sets in the ComVis project no multisets are allowed, which means, that each element can occur only once in a set. Nested sets are not supported either, meaning that a set cannot be an element of another set.

3.3 The Profile Histogram

During experiments with sets in conventional InfoVis views the histogram proved to be very useful. Due to the inherent complexity of sets compared to categorical data, a simple view displaying only one dimension simplifies analysis of the data and makes it easy to get used to set-typed data. Therefore, the histogram has been chosen as a base for improvements. In the design process for the new, improved histogram, the first idea was a histogram for two dimensions. This could look similar to the mosaic plot by Hofmann [18]. Adding a second dimension to the histogram also adds complexity, which was not desired. A better approach than displaying more data, is displaying the data in a more detailed way. The question is: What are the interesting features of set-typed data? The histogram only displays the distribution of elements through all data records. It is possible to identify frequently used elements and rare ones. The accumulated number of elements can be used to estimate, how many elements each data record holds. But by using a traditional histogram for sets, the user cannot tell if some elements occur in combination with other ones, as single elements, or if the distribution of elements within the set is random.
The idea for the solution of this problem emerged when analyzing set-typed data in set-enabled views. A data record containing n elements in its set gets n times the screen space than a record with only one element. This circumstance can be accepted, if the user wants the number of elements to be reflected in the view. If the user does not want to be irritated by a high number of elements, he or she may want to have the same amount of screenspace reserved for each data record. Considering that in a data record with n elements, each element represents $1/n$ of the data record, we can divide the amount of screen space by n and each data record has a representation of the same size. A view can show both, equal screenspace per element, and equal screenspace per data record.
The following chapters illustrate the design of the adapted histogram view in a more detailed way.

3.3.1 Bars & Blocks

The main view, which resembles a classic histogram, consists of bars and blocks. Each possible element in the set is represented by a vertical bar in the view. The height of the bar indicates the number of data records containing this element. A data record containing more than one element adds its share to multiple bars. For data records containing no elements at all, a special bar representing the empty set is provided.
One aspect of a histogram is, that each data record is represented by the same amount of screenspace. This is ensured by the fact, that each data record has exactly one value. The amount of screenspace can vary according to the distribution of the histogram, but it will

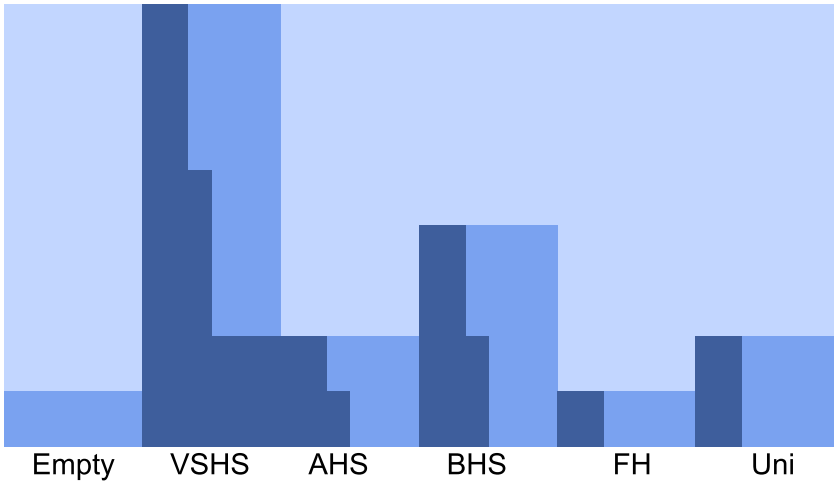


Figure 3.3: Sketch of a profile histogram. The width of blocks (in dark blue) represents the number of total elements in their sets. While the light blue bars use screenspace proportional to the number of elements, the blocks use screenspace proportional to the number of data records.

always be equal for all records in the dataset. When visualizing a set-typed dimension, the amount of screenspace for each data record is increased for each element it contains. Therefore, a visualization method providing both, equal screenspace per data record, and equal screenspace per element-to-record assignment, is desired.
To get a more detailed insight into the structure of the dataset, blocks have been introduced. Each bar contains blocks up to the number of possible elements in the set. These blocks are used to display the distribution of the cardinalities of the data records' sets. Each block represents all data records having a common number of elements as the set value in a set-typed dimension. If a set-typed dimension containing n elements is visualized, there can be up to n blocks in each bar. The height of the first block, starting from the bottom, represents all data records containing only the element represented by the bar – the cardinality of the set-value is one for all these records. The second block's height is proportional to the ones having exactly one element in addition to the one represented by the bar. The n^{th} block represents data records containing all elements of the set.
With the concept of blocks, the amount of screenspace used for all blocks is proportional to all data items. Data records containing two elements are represented by two blocks, with each block's width divided by two. Figure 3.4 shows a diagram illustrating that this fact. We can also learn from it, that narrow bands of small blocks represent fewer data items containing many elements, whereas wider blocks represent many data items with only few blocks.

3.3.2 Scaling Modes

Another feature of the profile histogram are the three different scaling modes (see figure 3.5). These modes alter the way, bars and blocks are scaled to give a good impression of proportions.

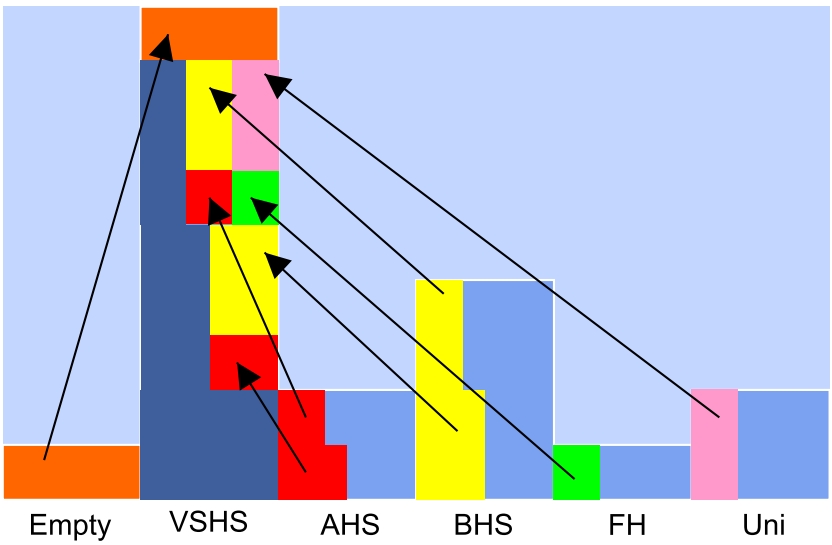


Figure 3.4: Sketch of a profile histogram showing, that the sum of all blocks is equal to one full height bar. By moving blocks related to the same data item together, the gaps in the second bar can be filled. Since we created a special element to represent the empty set, this element can be treated as a single one with full width (special case).

Normal Mode In the normal mode (figure 3.5a), the profile histogram acts like a conventional histogram. The highest bar is exactly as high as the display window, and all other bars are scaled accordingly. It is fairly easy to compare the sizes of bars and blocks, which makes this a good mode for most use-cases. The major drawback of the normal mode is the fact, that it is hard to estimate proportions in respect to the total number of data items. In a histogram without sets, adding all bars together would account for a theoretical bar representing all data items. If we add a data item to a certain bar, we have to subtract it from another one. In the case of sets, this is no longer applicable. Each bar can contain a number of data items between zero and the total number of data items, without necessarily leading to a different visual representation.

Scaled Mode In the scaled mode (figure 3.5b), the height of the bars is defined by the total number of data records in the dataset. A bar containing all available data records is exactly as high as the display window. In most cases, there is no such bar, that is why there can be a significant loss of screen space in relation to the normal mode. The main advantage of the scaled mode is, that each bar's height can easily be compared to the display window's height, which represents the total number of records in the dataset. In most cases that do not require exact numbers, it is not necessary, to keep the total number of data items in mind, since relations can easily be estimated. One drawback of this mode is that in most cases, all bars are lower than in the normal mode. This can be a problem if there are some very low bars, whose height can hardly be estimated. The problem gets even worse with blocks, because in most cases there will be even smaller blocks that are hard to see and to point at – and of course difficult to brush. This problem can be partly solved by the scalesliders (see section 3.3.5 for details), but when zooming into the view, it is no longer possible to compare the bars' heights to the window size.

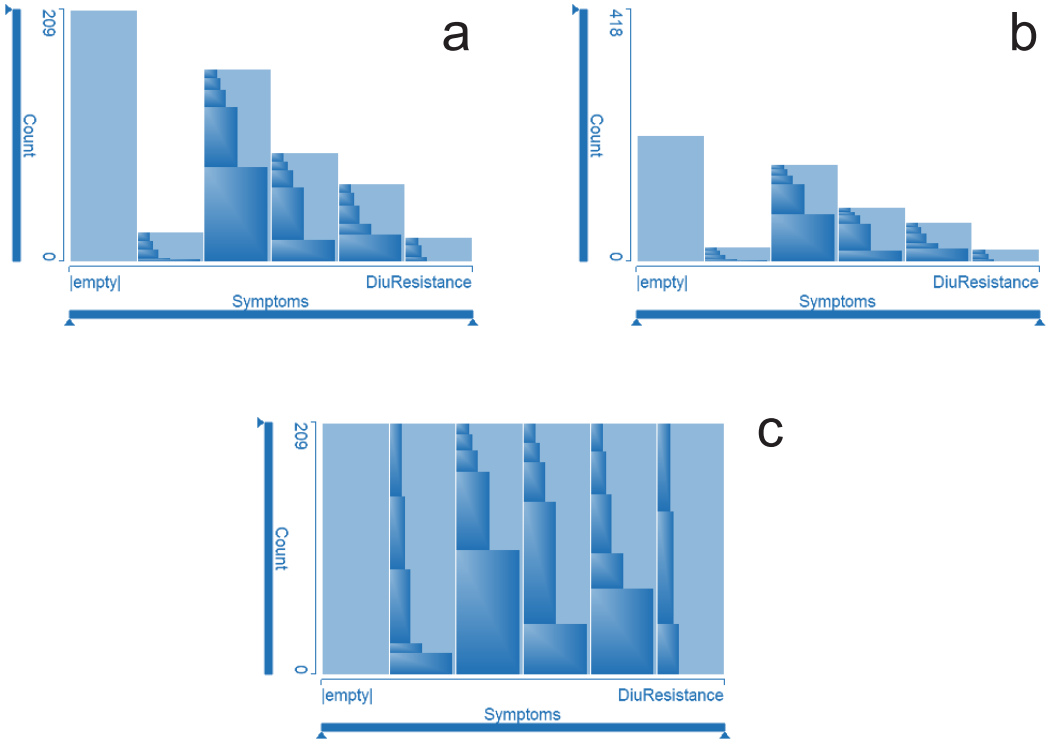


Figure 3.5: Figure *a* shows the normal mode, the default mode, which is also used in traditional histograms. Figure *b* shows the scaled mode, where all bars can be compared to the window's height, to estimate relations. In the relative mode (*c*), all bars are at the same height. This mode is useful to compare blocks within the same bar to each other.

Relative Mode In the relative mode (figure 3.5c), each bar is scaled to the display window's height. It is not possible to compare bars to each other any more, but it is the best mode to compare the sizes of blocks inside a bar. However, when comparing blocks of different bars, one of the other two modes should be used instead. When brushing single blocks or using the mouse-over feature, this mode has to be used very carefully, to avoid wrong conclusions because of differently scaled bars and blocks. However, this mode is good to compare the amount of brushed data records in a bar in relation to all data records, if a brush has been created in another view.

3.3.3 Brushing

Brushing in the profile histogram has to be designed carefully to circumvent various problems mentioned before. It is important that brushing of each visible element has to be possible, and in contrast to the traditional histogram, each bar contains multiple blocks. That is also the reason why the y-dimension of a brush is important in this view. A brush is no longer identified by an interval of brushed values for a certain dimension, but also by the amount of elements in a set. Figure 3.6 illustrates brushing of blocks in the

profile histogram. Another problem with brushes comes apparent when using the different scaling modes of the profile histogram. In contrast to the conventional histogram, bars can be partly brushed by brushing single blocks. In case of two high bars and lower bars between them, it is possible to select the top blocks of two bars without brushing the bars in between (see figure 3.7). This leads to a brush, which is impossible to create with a single rectangle in relative mode. Also the opposite situation is possible. Bars of different height can be partly brushed in relative mode and create brushed blocks that are not connected in the other two modes. These problems do not exist when switching between in normal and scaled mode.

3.3.4 MouseOver Effect

The mouseover effect is a special feature used in the profile histogram to emphasize visual items belonging together. One of the main problems with the analysis of sets is, that visual items representing elements of the same data record are not visually connected, or at least marked as related items. The mouseover effect adresses this drawback by highlighting related blocks. When we recall the fact, that each block is a group of elements having the same number of additional elements in its set, we can say, that all other bars can have a block that may contain related elements.
Whenever the mouse cursor is positioned over a block, this block, and those parts of other blocks, that are related to it, are highlighted. Considering a highlight on the i^{th} block of a bar, each element of this block has additional $i-1$ elements scattered over all i^{th} blocks of other bars. During a mouseover action the corresponding parts of related blocks are highlighted as well, so we can get a quick overview on how the sets are composed. Because the width of the main block we are highlighting is divided by i, and we have $i-1$ additional elements in each set, we can show easily, that the accumulated amount of screenspace is equal for each element.
Simplifying the last paragraphs, we can say, that the mouseover effect highlights those parts of the view, that would be brushed, if we would brush the block currently under the mouse cursor. However, it does not allow selection of multiple blocks and in the current implementation it is not linked to other views. When highlighting a block surrounded by other blocks, it may also be impossible to move the mouse out of the view without changing the focused block. On the other hand, the mouseover effect is implemented to be fast, because the results have to appear instantly - otherwise users may be confused, because the mouse cursor does not match the highlighted block. It is not supposed to replace brushing, but as a simple method to support understanding complex sets.

3.3.5 Scale Sliders

Zooming and panning are very important interaction features in information visualization. Whenever there is not enough screenspace to display all available information, it is important for the user to zoom into the view, to provide a better insight into the details. If zooming is allowed, also panning should be possible. Both, zooming and panning are provided by scalesliders (see figure 3.8). These sliders allow to reduce the visible area on the left, right, top or bottom edge of the view. The remaining blue bar represents the size of the visible area in respect to the entire visualization area. These bars can be moved around to move the viewport around. By doubleclicking the white area next to a bar,

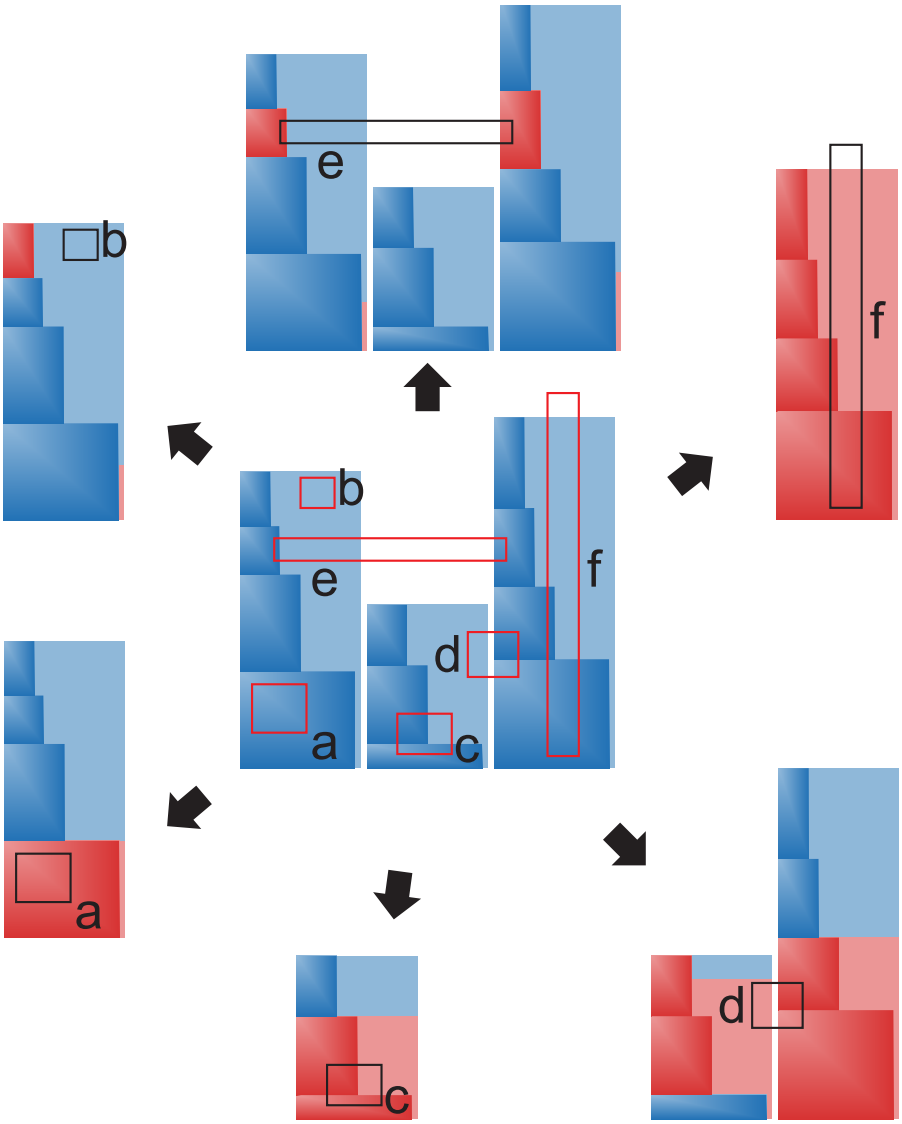


Figure 3.6: These six examples demonstrate brushing of blocks. The image in the center shows three bars with several blocks inside them. Six brush rectangles have been drawn into the diagram and the results can be seen in the surrounding sketches. Note, that only directly affected blocks have been highlighted – data records can of course contain multiple elements and therefore lead to partly brushed blocks in other bars. Brush *a* shows, how a single block can be brushed by drawing a rectangle into it. A block is also brushed, if the brush geometry is at the correct y-coordinate inside the bar, but does not intersect the actual block (figure *b*). Brush *c* selects a range of two blocks, that are intersected by the brush rectangle. Figure *d* shows a brush rectangle intersecting four blocks of two different bars. It is not important, if the blocks have the same index, one single brush can select blocks with different indices (starting from the bottom). The Brush in figure *e* spans over three bars, but affects only two of them, because the one in the middle is too low. Brush *f* shows, how an entire bar can be brushed. The brush geometry has to intersect the y-coordinates of all blocks, but not the blocks themselves. Of course, the brush rectangle can be taller than the bar.

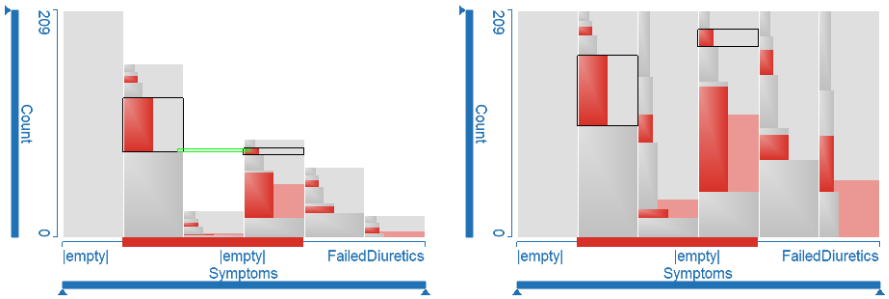


Figure 3.7: A single brush (green rectangle) is affecting two non-neighbouring bars in the image on the left. Because in the right image, all bars are at the same height, this brush can not be created there.

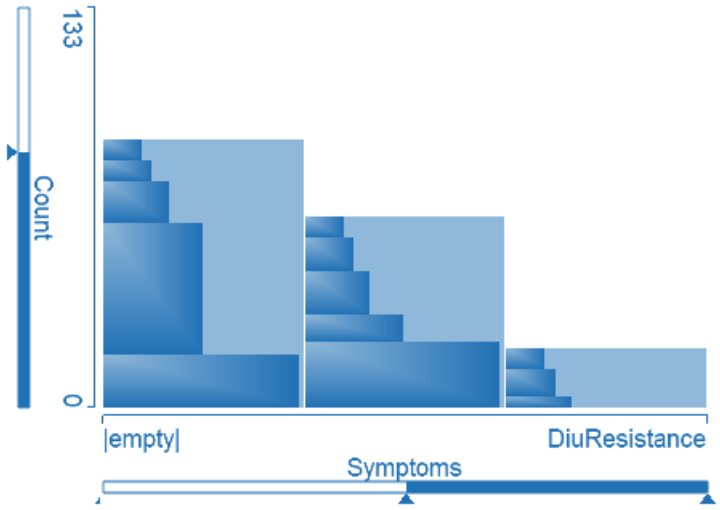


Figure 3.8: The blue triangles can be dragged along the scaleslider bars to define the size of the viewport. The remaining blue bar can be used to move the viewport.

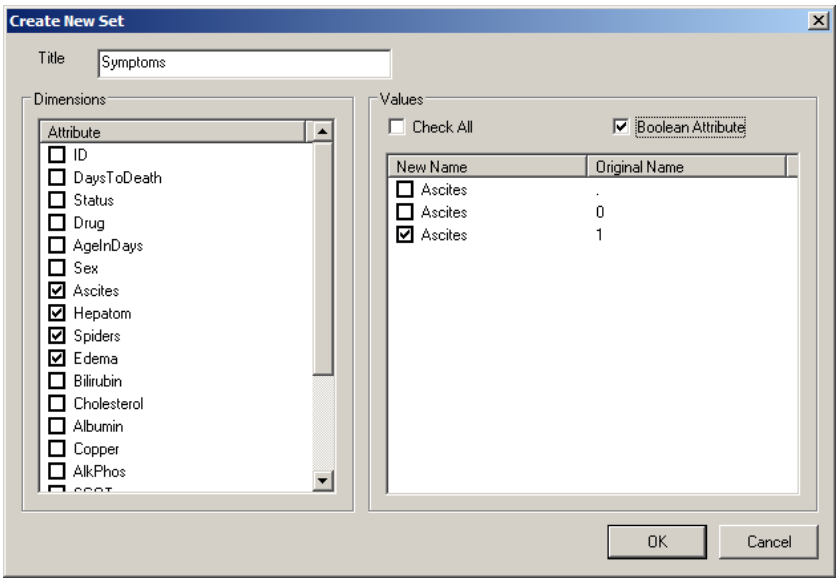


Figure 3.9: The converter dialog can be used to create a set dimensions from categorical ones. If all selected elements in a dimension map to the same set element, we are performing a boolean/binary conversion.

this one end of the bar is returned to the original position. This feature is consistently provided by most views in the here discussed visualization framework.

3.4 Data Conversion

Because sets are a new topic in information visualization, datasets containing this datatype are rare. However, there are many datasets, that contain groups of categorical dimensions with only few different categories. In many cases, only two categories exist, such as "yes" and "no", or "1" and "0". These *binary* dimensions are best-suited for a conversion to a set-typed dimension. However, also dimensions containing more categories can be converted, and even existing set-typed dimensions can be used as source dimensions. The user interface for conversions is depicted in figure 3.9.
If we have, e.g., a dataset containing education data about people, like the CRM dataset (see chapter 4.2), there are multiple dimensions representing levels of education. If we just wanted to store the highest level of education for each person, we would only need one categorical dimension containing values like "high school" or "college". If we want to keep track of all levels of education, we need multiple categories for each person. This can be done by multiple categorical dimensions or by creating a set-typed dimension. Each data record (representing a person) can then contain multiple elements (representing educational levels) in its set. In this chapter, the theoretical details of set-conversions will be explained in more detail. A conversion generally involves at least one source dimension and exactly one destination dimension. While the the source dimension can be categorical or set-typed, the destination is, of course, a set-typed dimension. In general, we can distinguish between two different types of conversions, that are treated differently.

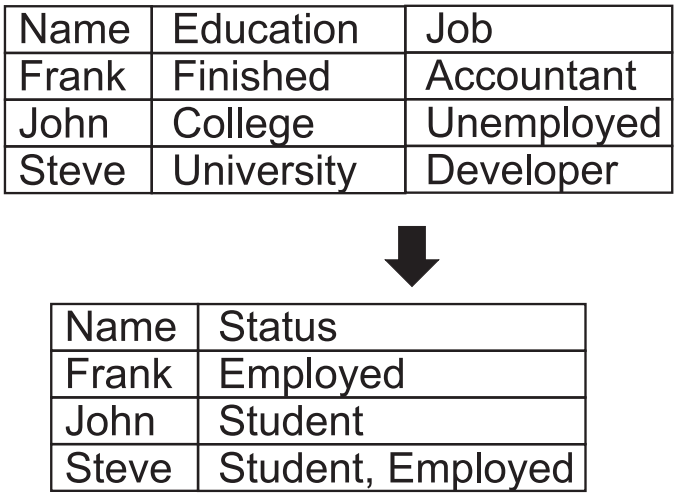


Figure 3.10: Binary conversion from two categorical columns to a set-typed dimension. We extract the information if a person is a student from the second column, and if he or she is employed from the third column. The result is a set-typed dimension containing information if a person is studying and/or working.

Binary Conversion Conversions are called "binary", if the source dimension contains two different categories in a column, or the number of categories is reduced to two values, that can be seen as "true" and "false" (or the like). In the case of a data record containing the value "true", an element named like the column's name is added to its set, if it is false, nothing is added to the set. If we have, e.g., a table containing people, and a column called "student", containing "yes" and "no", we can add the element "student" to the set in the destination dimension, if this particular person is a student. If he or she is not, we do not add "not a student", but we simply do not add any element to the data record's set at all.
A conversion is still considered as binary, if there are more than two categories in the source dimension, as long as they can be divided into two groups: "true" and "false". If the matching category for a specific data record belongs to the "true" group, an element is added to the set in the destination dimension. The name of the element does not necessarily need to be equal to the source dimension's name, but in most cases this solution makes sense. An example of a binary conversion is shown in figure 3.10. In our approach, binary conversion has been partly automated. When choosing a categorical column with only two different values (1/0, yes/no, true/false, on/off), the "positive" category (1, yes, true, on) is selected and the name of the element to be added to the set is set to the name of the categorical column. Of course these settings can be changed by the user and also columns not recognized as binary can be converted that way. The only important fact is, that only one element per dimension can be added to the set.

General Conversion General conversion takes place, when more than one element per column is transferred from the categorical column to a set-typed dimension. This method can convert a categorical column to a set-typed dimension by simply adding each category's name to it, if it occurs in a data record. In some cases it is necessary to form groups of

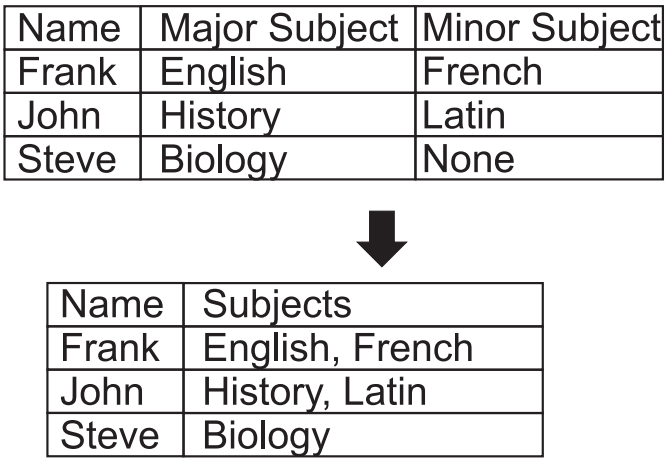


Figure 3.11: General conversion from two categorical columns to a set-typed dimension. We extract information on major and minor subjects of students, and add them as elements to the set-typed dimension "subjects". The result is a set-typed dimension containing information on the students' subjects. Note, that it is still possible to omit categories (in our case "none").

categories that are converted to the same element in the set. A group of categories that does not correspond to an element in the set is optional. It is also possible to add multiple elements to the set for a category or a group of categories. An example of a general conversion is shown in figure 3.11. In the dataset "primary biliary cirrhosis" (see chapter 4.1 for details) we convert the column *Edema* to a set column. This column has three different values (0, 0.5, 1) and the values 0.5 and 1 have a complex meaning. For details, consult the next chapter.
In our visualization framework, general conversions are preselected for categorical columns exceeding two different values. The names of the elements added to the set column is set to *Title:Category* of the source column. Using the column's title as part of the element's name reduces the probability of name conflicts. In most cases the elements' names can be shortened though.

Reordering Elements Because categories, as well as elements of set-typed dimensions have no inherent order, they can be reordered by the user. This can be done using the *Reorder Dialog* (figure 3.12). The order defined by the user is consistently maintained in all views.

3.5 Adapting Other Views for Set-Typed Data

The ComVis visualization system provides a wide variety of general purpose views to display different kinds of data. It can be used to visualize numeric data, categorical data (strings) and time series. Since we want to deal with multivariate data, we have to make it possible, to combine the visualization of set-typed data, as well as dimensions of other types in one application – and even in the same views. These views enable the user to explore a multidimensional dataset by using suitable views for different dimensions and

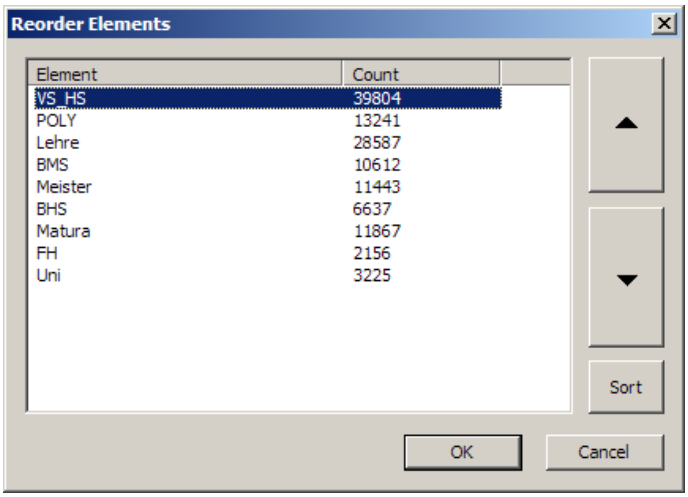


Figure 3.12: The Reorder Dialog allows both, alphabetical and user defined sorting of the elements of a set. This order is maintained in all views, where this set-typed dimension is visualized.

the ability to combine brushes with operations. Whenever a visualization view should be extended to display set-typed dimensions, three important special cases have to be considered.

1. Due to the nature of sets, each data record can have multiple elements and has to be represented accordingly. The number of glyphs does not necessarily represent the number of data records, in most cases it will exceed it.

2. Data records can have an empty set, containing no elements at all. If no special precautions are taken, the correct solution for such a data record would be to not display it at all.

3. Brushing sets is a difficult topic. While implementation is a fairly easy task, understanding and analysing the result can be challenging depending on the type of dataset and the user's experience. If a data record is brushed because of one of its elements, all elements of this record have to be shown as brushed. If the brush is rendered as a rectangle, there can be brushed values outside this region, which can be confusing for the user.

3.5.1 Histogram

The histogram is a simple but yet very effective and useful view that is often used in data visualization. It can display one dimension of numeric or categorical data. The bars displayed in the view show the accumulated number of data records for the specified category or – in case of a numeric dimension – the interval represented by the bar. The dimensions of the bars are normalized to that the height of the view, so the highest bar is as high as the view and all other bars are scaled accordingly.
Necessary changes to the visual concept of this view are minimal. When using the histogram to display a set-typed dimension, each bar in the view represents one possible

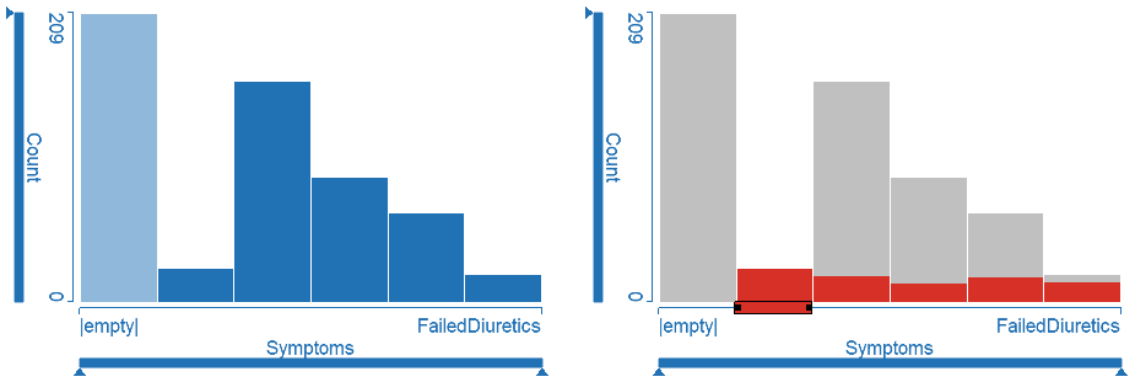


Figure 3.13: In this realization, a histogram that displays a set-typed dimension is recognized by the semi-transparent first bar. Because data records can contribute to multiple bars, there can be selected items outside the brush geometry. In the second image, only the second bar is brushed. The red parts of other bars represent data records, that also contain the element in the second bar.

element of the sets. The height of the bar scales with the number of data records containing this element in their respective sets. An additional bar is displayed to represent empty sets. In the current solution, the empty bar is drawn with an opacity of 50%, to emphasize the fact, that it differs in various ways from other bars. Because each data record can contribute to multiple bars, brushing one element can lead to partially brushed bars outside the brush geometry (see Figure 3.13).

3.5.2 Scatterplot

The scatterplot is a very common view with widespread use in different information visualization systems. The user can select two dimensions from the currently opened dataset to be displayed in a Cartesian grid. The domains of numeric axes are contrained by the minimum and maximum values of the columns. In case of a categorical dimension, categories are placed along the axis in usually equal distances. Each data record is represented by a dot (or a small glyph) in the view. The dots are positioned according to the values of the corresponding data record in the two selected dimensions. ComVis supports variable point sizes to to reflect the number of data items at a specific position. If many points are to be drawn at the same position, they are replaced by one larger point.
The scatterplot view is a good choice when it comes to comparing two dimensions. The more the points in the view align along a line or a curve, the higher their statistical correlation is. The scatterplot needs more modifications to support the handling of sets. The most important difference is, that the number of points is no longer limited by the number of data records. The maximum number of points can be multiplied by the number of elements per set of both dimensions. Of course, when having a very high number of points projected into a small area, there will be a high number of overlapping points. These points can then again be substituted by fewer larger points.
The scalable points feature (see figure 3.14) is very important when dealing with sets. It can be used to reduce the amount of screenspace taken up by large amounts of points. If a specific data record d has n elements in its set, we need to draw n points into the

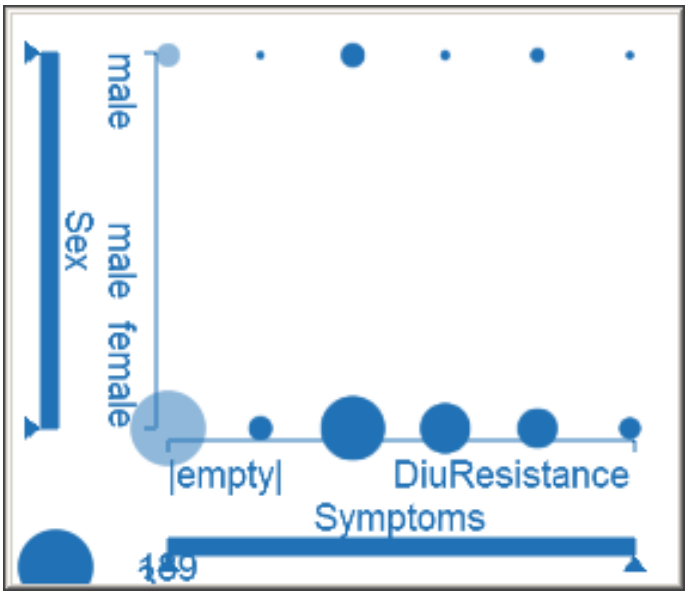


Figure 3.14: This scatterplot view displays one set dimension and one categorical dimension. The semi-transparent points represent the empty set - in this case patients without symptoms.

scatterplot. Since the sum of all n points represents the data record, each point can be seen as $\frac{1}{n}$ of the data record. In case of accumulated large points we do no longer increase the point's size by one for each record, but we simply add the fractional value. Drawing smaller points than the minimum point size is not possible in the current implementation, because it could be irritating for the user if he or she defines a minimum size and even smaller points appear on the screen. A scatterplot can also visualize two set-typed dimensions (see figure 3.15), but such a visualization can be more difficult to read and to interpret. It is possible to estimate the number of data records by comparing point sizes, but because many data records contain multiple elements in both of their sets, they affect multiple points in both set-typed dimensions. This becomes obvious if a single point in the scatterplot is brushed, and brushed regions appear in many other points, too.

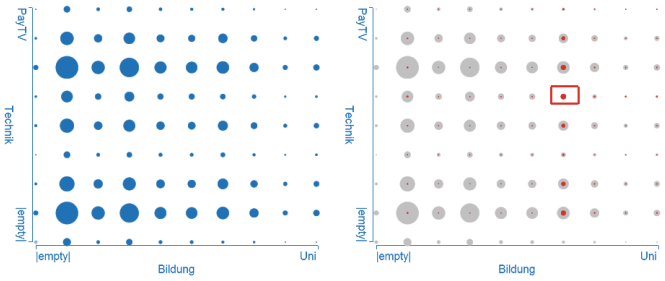


Figure 3.15: This scatterplot view visualizes two set-typed dimensions. When brushing a single point (right image), there appear red dots in all other points (except the ones containing empty sets in at least one dimension).

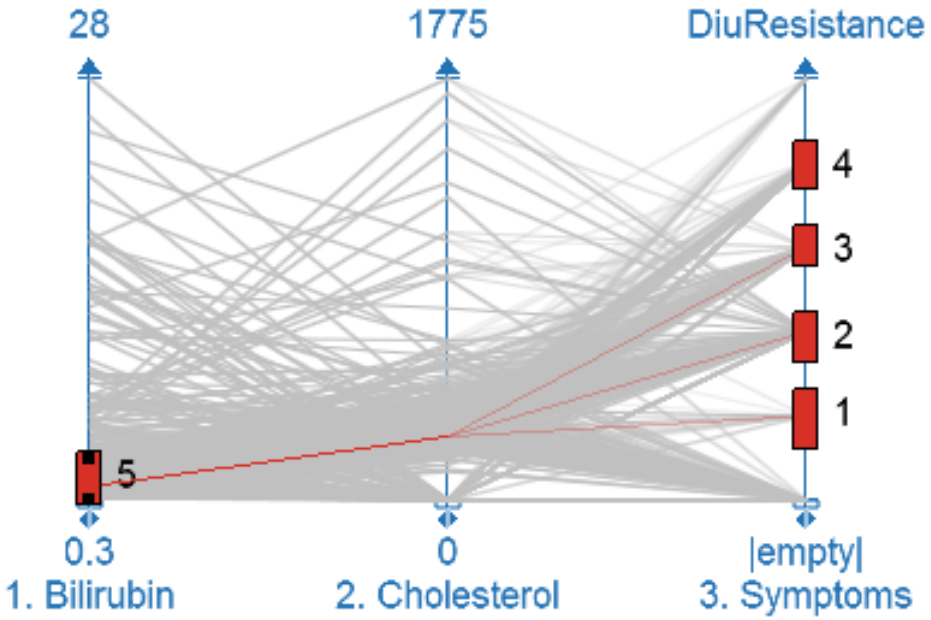


Figure 3.16: Various brushes are used to narrow the selection down to one data record. The highlighted line segments represent one unique data record. Unlike traditional parallel coordinates, the line splits to intersect the set axis at multiple positions. The opacity of the three sublines is divided by three here. Brushes in this view are only used to narrow the selection down to one data record.

3.5.3 Parallel Coordinates

The parallel coordinates supports the display of more than three dimensions. This feature makes this view very important for high-dimensional datasets. In parallel coordinates, there can be a potentially larger number of axes – ComVis supports up to eight. Each axis represents a dimension, and each data record is represented by a line crossing all axes. The position of the intersection with an axis represents the actual value in this dimension. The problem when using parallel coordinates with set-typed dimensions is that one data record can contain multiple elements. The polyline representing this record has to cross the set axis at all possible intersections, which can be irritating for the user. A line representing a data record with multiple set elements has to split at the axis before the set axis, each segment has to cross the axis at a different position, and all segments have to be joined at the next axis. When two different set axes are places next to each other, each available segment of the first axis has to be connected with all necessary positions on the second axis. This can lead to massive overdraw in crowded parallel coordinate views. Because the high amount of used screen space is a general problem in parallel coordinates, a solution has to be sought. Again, the question is, whether the amount of screenspace should be equal per record or per element. Generally it is a good idea, to have an equally prominent representation for each record, as it makes estimating the amount of data easier. The parallel coordinate view also supports transparent lines to make it easier to find clusters in the dataset. This kind of view can be very hard to analyze if there are a few thousand lines. When using semi-transparent lines, it is easier to get a good insight into the structure of the dataset.

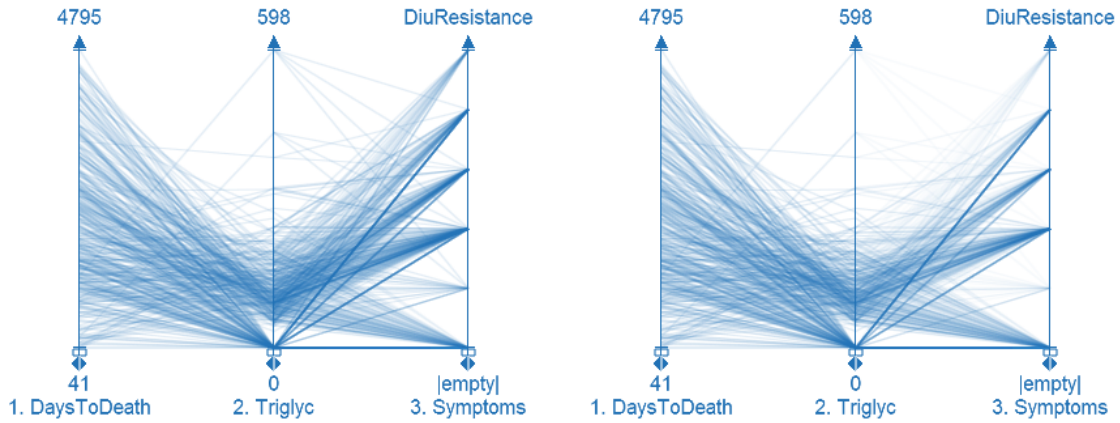


Figure 3.17: Both scatterplots represent the same dimensions of the same dataset. The first two axes represent numeric data, while the third one displays a set-typed dimension. In the left parallel coordinates view, each line is drawn with the same opacity. Since there are more lines between the second and the third axis, the overall opacity reflects the number of element to record assignments. In the picture on the right the opacity of the lines between the second and the third axis is lowered. The opacity of a subline is divided by the number of elements this data record contains in its set. Therefore, data records containing many elements are not overrepresentated anymore.

With sets this problem gets even worse. In most cases, the number of lines crossing an axis displaying a set-typed dimension is higher than with axes of other data types. So while transparency is still a good way to make the distribution of a set visible, the coloring in the neighbourhood of a set axis becomes more intense. If we decide to represent each data element with the same intensity, this result is reasonable. We can estimate the number of lines crossing the axis at a specific point, and compare it to other crossings on other axes. However, in some cases, especially if many elements are used by each data record, the colors can become too intense next to a set dimension. In such situations we can divide the opacity of the line by the total number of elements used in this record. This way, the overall intensity of the view stays the same. If we have two neighbouring set axes, and want to display a data record with n elements in the first axis and m elements in the second axis, we have to draw $n * m$ lines. In this case, the opacity of the lines representing this dataset between these two dimensions has to be divided by $n * m$, too. That way the overall amount of opacity resembles the situation between other axes.

Chapter 4

Demonstration

This chapter will give an overview on two interesting datasets and thereby illustrate the possibilities of data analysis based on sets. The focus here is on comparing the use of sets to categories and the advantages of the profile histogram as a set-optimized view over traditional views that have received slight modifications to allow the visualization of sets.

4.1 Primary Biliary Cirrhosis

The dataset "Primary Biliary Cirrhosis" [15] is a collection of blood measurements and symptoms collected in the Mayo Clinic trial in primary biliary cirrhosis between 1974 and 1984. The dataset contains data about 418 patients, 312 patients took part in the trial, the other 106 patients did not participate in the trial, but allowed measurements and the recording of the data.
This dataset has already been investigated by Revett et al. [40, 41]. While they have examined this dataset with statistic methods, namely *rough sets*, this document will concentrate on methods of visualization. These methods do not claim to deliver statistically correct results, but to assist users in exploring the dataset, and finding correlations on their own.
The dataset contains various numeric dimensions for blood measurements and some categorical columns containing binary values of the occurrence of various symptoms. We will concentrate on symptoms rather than on numeric blood measurements. However, other data types are important for proper analysis of the dataset, too. Each type of information in a dataset should be represented by the data type that fits best.

4.1.1 Set Conversion

The dataset contains four categorical columns defining symptoms we analyze further. When converting these columns to a set-typed dimension, a column called *Symptoms* is created and the information stored in these four columns is converted to elements of the sets. The description included in the dataset gives further information on the symptoms shown in table 4.1. The four binary columns describe common symptoms of liver cirrhosis, that can occur with different probabilities. *Ascites* is an abnormal accumulation of fluid in the abdominal cavity, which in most cases is an indicator for liver cirrhosis. It can be resolved by salt restriction and *diuretics*. A diuretic is a drug that increases the rate of excretion of water from the human body. It can also be used to resolve an *edema*, which

column	description	value domain
ascites	presence of ascites	0=no; 1=yes
hepatom	presence of hepatomegaly	0=no; 1=yes
spiders	presence of spiders	0=no; 1=yes
edema	presence of edema	0=no edema; 0.5=edema without diuretics, or edema resolved by diuretics; 1=edema despite diuretics

Table 4.1: Available categories in the Primary Biliary Cirrhosis Dataset.

is an accumulation of fluid in a human organ, in our case, the liver. A *hepatoma* is a dangerous tumor in the liver. *Spider angiomas* are small reddish spots under the skin, looking like small spiders due to little red extensions. The spider angiomas themselves are benign, but they are often a result of a liver disease.
As we can see, the first three columns are rather easy to handle. A boolean variable tells us, if a patient shows this symptom. If we have the value "1", we simply add the column's name to the patient's set, otherwise we do not add anything. If we have a set containing all symptoms for a patient that shows these three symptoms, the set would include the elements *Ascites*, *Hepatomegaly* and *Spiders*. Since there are no negated elements, a patient with no symptoms is described by an empty set.
The column *Edema* is more complex. The value "0" indicates, that no edema has been diagnosed, whereas the value "1" indicates an edema that has not been resolved by diuretics. However, a value of "0.5" means, that the patient has not undergone a diuretical treatment to cure his edema, or the edema has already been resolved by the diuretics.
Considering these hints given in the dataset, we can say, that all patients having a value different from "0" in the column *Edema* suffered from an edema. Since we cannot distinct between patients with successful diuretical treatments and patients without treatments, the only other fact we can deduce is, that a value of "1" indicates an unsuccessful diuretic treatment.
When converting the column *Edema* to a set, we simply add *Edema* to the set, if the categorical value was "0.5" or "1", and we can also add *DiuResistance* if the value equals "1".

4.1.2 Analysis

When analyzing this dataset, we concentrate on the dimension "symptoms" because it is the only dimension of the datatype set. Of course there are coherences with other dimensions and they will be examined too.
In the first step, we open a profile-histogram view and display the dimension Symptoms (see figure 4.1). This view gives a good overview on probabilities of different symptoms. We can see, that half of the patients do not show any symptoms at all. The most frequent symptom is a hepatoma (third bar), the least probable symptom is a resistance to diuretics. However, we have to consider, that only patients with a diuretic treatment are in this group, and a treatment only makes sense, if an edema or ascites has been diagnosed. We can also see, that there is no block of the full width in this bar, so this symptom cannot appear without any other. Figure 4.2 shows the profile histogram after a brush has been applied to the bar representing the element "DiuResistance". By looking at the height of

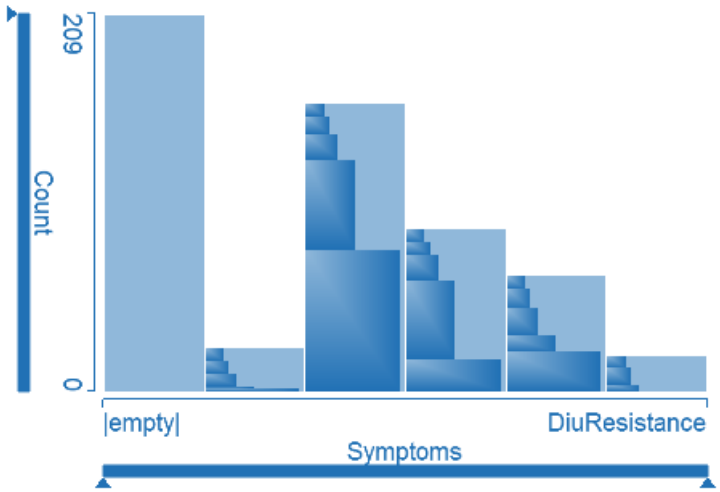


Figure 4.1: This profile histogram shows the distribution of the elements of the set *Symptoms*. We can easily estimate probabilities of symptoms. The height of the full-width block in the hepatoma bar tells us, that about 50% of all patients with liver tumors do not show any other symptoms.

the light-red bars, we can see, that people resistant to diuretics can have all symptoms with a nearly equal probability. If we switch to relative mode (see figure 4.3), we can see, that most people with failed diuretics also have ascites, which makes sense, because diuretics can be a treatment to it. All of the selected patients also suffer from an edema, which is obvious, since we extracted this feature from the binary dimension "edema". While there can be a significant correlation between the patients' gender and a specified symptom, the number of additional symptoms can also have an influence. Figure 4.4 shows, that female patients are more likely to have additional symptoms to a hepatoma, while many male patients (see figure 4.5) do not have additional symptoms. We can get another insight from this view by analyzing the "spider" bar. Female patients greatly outnumber male ones when looking at people with spiders and at most one other symptom. Patients with more than one symptom do not show any influence by the gender.
These insights cannot be gained using a traditional histogram which shows no information on the number of additional elements in a set. It would be possible to get this information by creating an additional dimension containing the number of elements in each set. Then, any view allowing the display of two dimensions allows to brush the same parts of the data. This would involve multiple combined brushes, and would be less intuitive. Another advantage of sets and the profile histogram is, that the setup time is much shorter compared to multiple scatterplots or a parallel coordinate view containing the same data from categorical dimensions. The mouseover-feature in the profile histogram prevents some unnecessary brushes as well, and can be very useful when looking for correlating elements in a set.

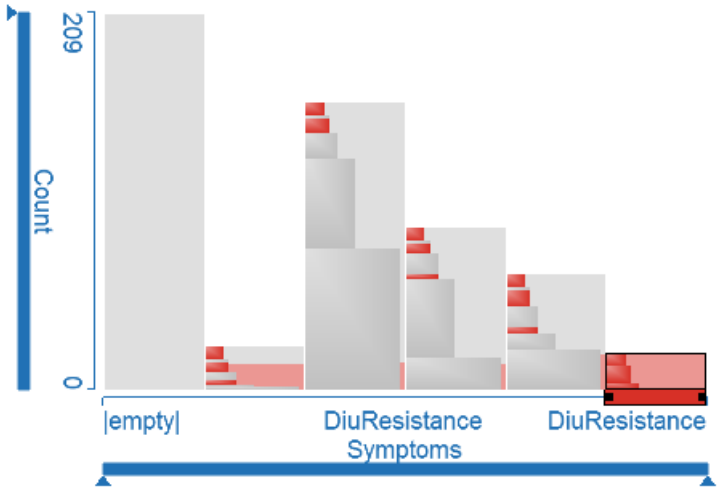


Figure 4.2: When selecting patients with a diuretic resistance, we can see that there is an equal distribution across other symptoms.

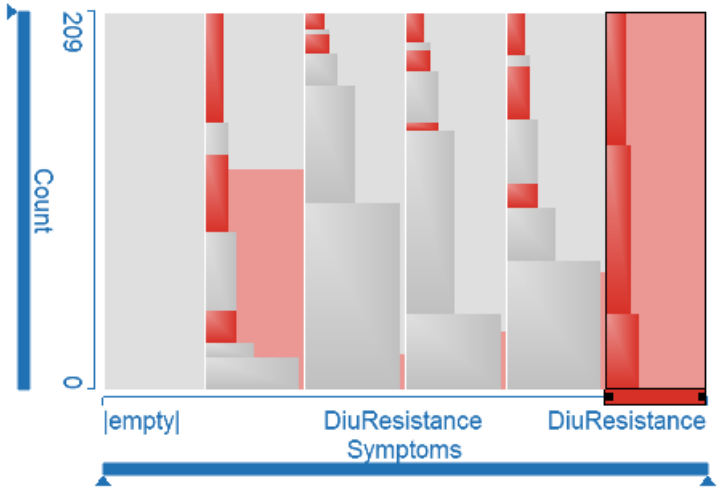


Figure 4.3: In relative mode we can see, that more than half of the patients with ascites (second bar) suffer from diuretic resistance. The small width of the blocks in the rightmost bar shows, that people with diuretic resistance also suffer from least two other symptoms.

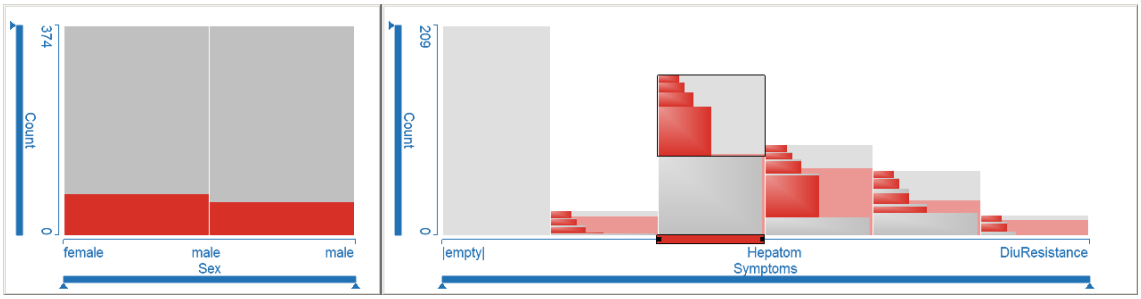


Figure 4.4: Patients suffering from hepatoma and at least one other symptom have a slightly higher (in fact nearly equal) probability to be female.

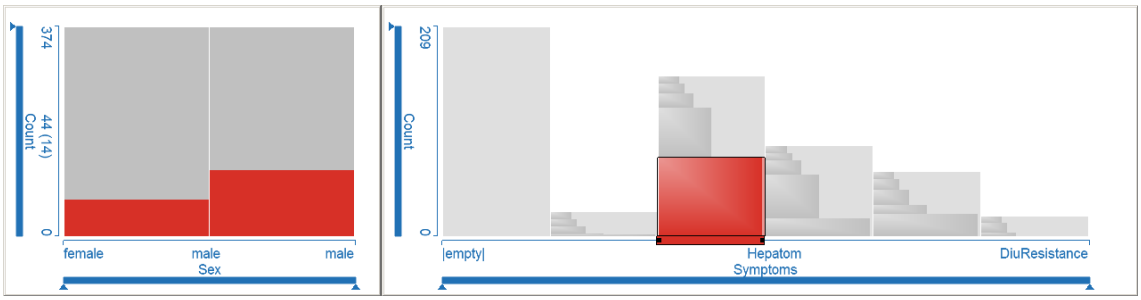


Figure 4.5: Patients suffering from a hepatoma without showing any other symptoms have a higher probability to be male.

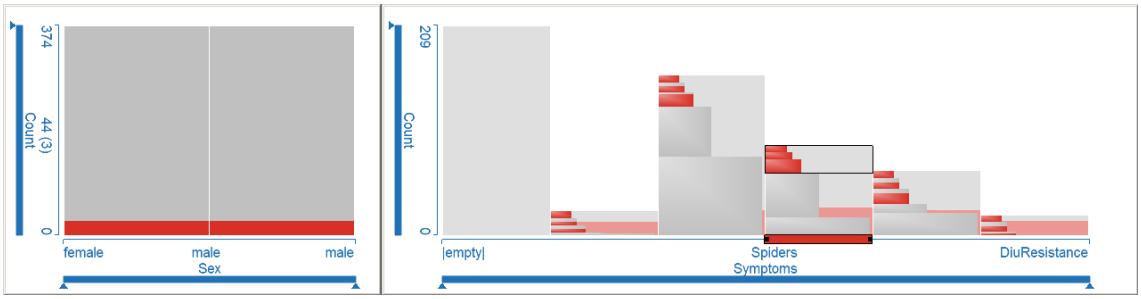


Figure 4.6: According to this brush, spiders in combination with at least two other symptoms seem to equally affect female and male patients.

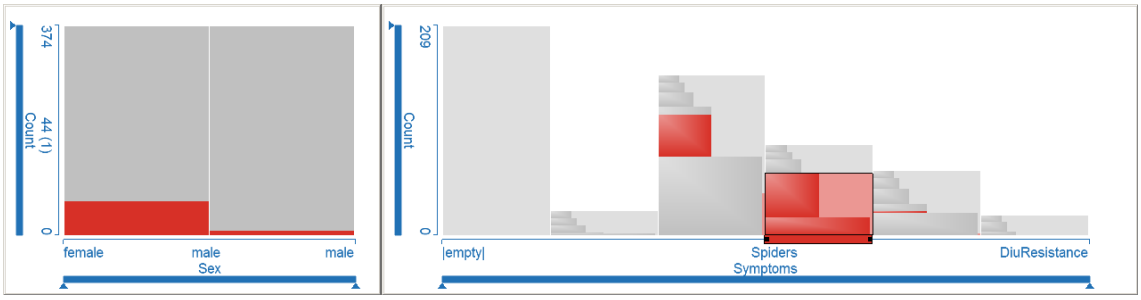


Figure 4.7: Patients with spiders and at most one other symptom are mostly female (in this dataset).

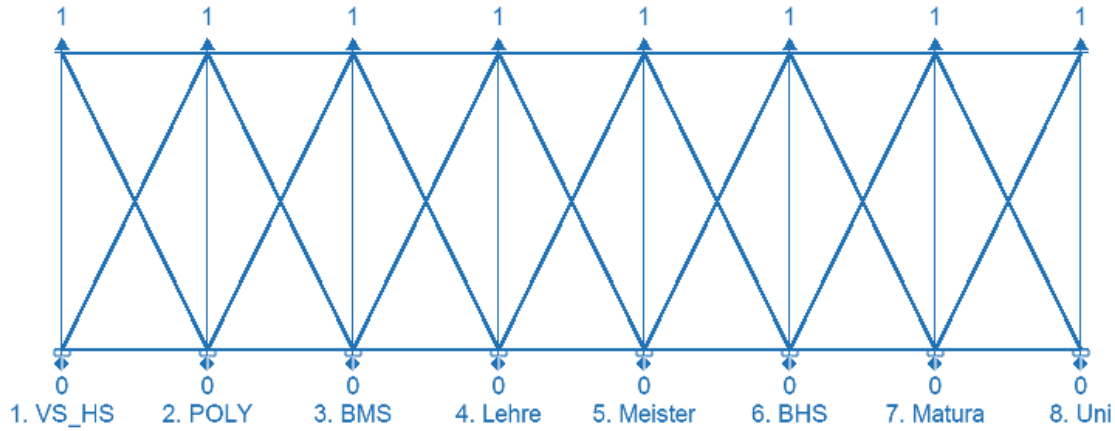


Figure 4.8: A parallel coordinates view displaying data about education. Each axis represents a kind of school. Even with transparency it would be very hard to estimate the number of congruent lines.

4.2 The CRM Questionnaire

The CRM Dataset has been provided by our CRM cooperation partner and contains results of a questionnaire of more than 65000 persons on their income, education and shopping habits. This Dataset contains more than 60 dimensions, with some groups of binary dimensions, which we will convert to set-typed dimensions. These binary dimensions originate from multiple choice questions. Each answer that can be checked results in a binary dimension of the dataset. The high number of dimensions makes this dataset hard to analyze. In many cases, it is very difficult, to display many dimensions at the same time, and only the parallel coordinates view is capable of displaying all binary dimensions belonging to the same group. However, this view is not suitable for binary dimensions, because in most cases the resulting visualization consists of thousands of congruent lines connecting all neighbouring ones and zeroes. A feasible solution to this problem is the parallel sets approach [30]. By replacing lines with bars, the amount of data records in a category can be estimated well. Because this solution uses different colors per category, it is much easier to keep track of the bars when they split up further. However, in large views, coloring can be mixed up a lot. Also, the setup time of a parallel sets view is significantly higher than in the profile histogram. Categorical dimensions have to be dragged into the view, may have to be reordered, and in case of non-binary dimensions, the categories may be in the wrong order too. While these minor problems can be solved by the user through an intuitive drag and drop user interface, it is still striking that binary columns are not the right way to deal with answers to multiple choice questions. A set is a much more natural way to represent all answers to one question, and that is why we are going to use sets for this dataset.

4.2.1 Set Conversion

The original dataset contains ten groups of binary columns, each group representing the possible answers for one question. Because all other questions do not allow multiple answers, they can be represented by single columns. By grouping binary columns and

Set	Description	Number of Valid Values
Bildung	Educational Institutions	9
Technik	Technical Devices	8
Geschaeft	Frequently visited Shops	16
Hunde_TR	Dry Dog Food Brands	5
Hunde_NA	Canned Dog Food Brands	5
Katzen_TR	Dry Cat Food Brands	5
Katzen_NA	Canned Cat Food Brands	5
WaschmittelArt	Kinds of Detergent	4
TReiniger	Cleaning Utilities Brands	4
Binden	Sanitary Napkins	4

Table 4.2: 65 binary columns are reduced to 10 set-typed dimensions. The number of dimensions in the whole dataset has been reduced from 105 to 50. This is a good example for efficient dimension reduction.

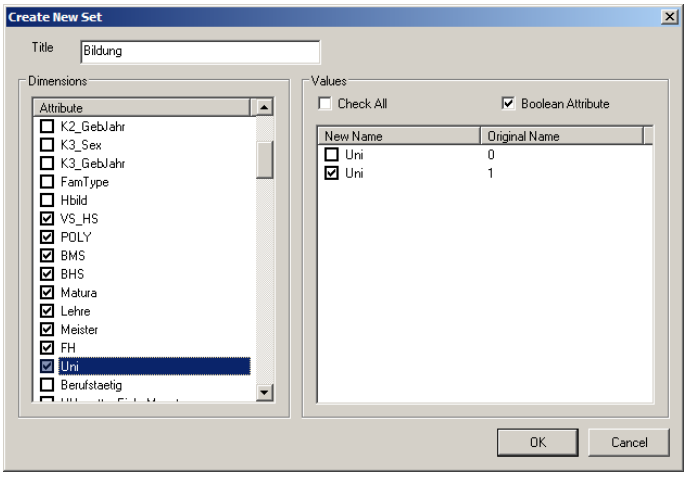


Figure 4.9: The Conversion Dialog creates a new dimension of the datatype *Set*. The user can select categorical values from different dimensions, and define rules for the conversion.

converting them to sets, this can also be achieved for multiple choice questions. See table 4.2 for the ten groups that can be converted to sets. There are nine types of educational levels in this dataset, that are converted to the set *Bildung (Education)*. After loading the dataset, ComVis brings up the *Data Settings* Dialog which allows the creation of a new set. By clicking the *New Set* Button, the *Create New Set* Dialog (see figure 4.9) is opened and the new set can be customized. We enter *Bildung* as its name and select the affected dimensions in the dimension list. ComVis automatically checks, whether each column can be converted binary. All our columns contain only the values 1 and 0, so the checkbox *Boolean Attribute* is enabled. A Value of 1 will create a set element named like the originating column, a value of 0 will be discarded.
After the set is created, the *Reorder* Dialog (see figure 4.10) is brought up. It allows the user to rearrange elements. In out case, educational levels should not be sorted alphabetically, but from the lowest to the highest level.

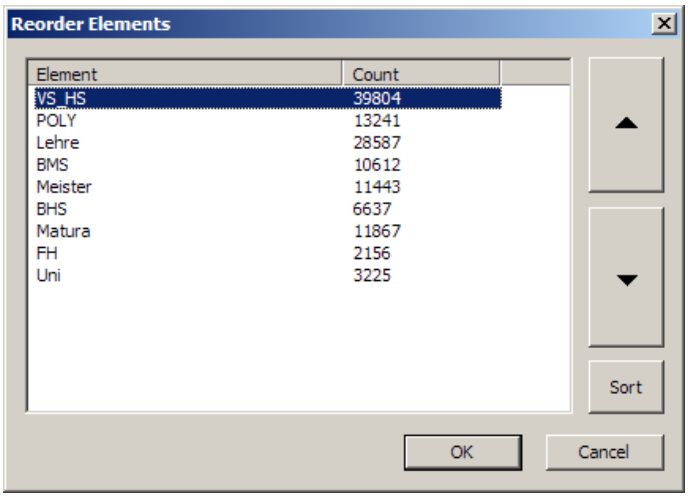


Figure 4.10: The Reorder Dialog allows both, alphabetical and user defined sorting of the elements of a set. This order is maintained in all views, where this set-typed dimension is visualized.

4.2.2 Data Cleaning

While datasets that originate from simulations or measurements usually contain mostly valid results, questionnaire data has a higher probability for errors. People filling out questionnaire sheets can make unintentional and also intentional errors. If the questionnaire is printed and filled out on paper, additional errors can occur when the collected data is imported into digital datasets. Some of these errors are obvious and can be corrected. Most other errors can be recognized, but not corrected. These errors can then be represented by a specified *invalid* or *missing* value.
However, a dataset can also contain errors, which cannot identified by simply looking at a value. Some values can occur as valid values in the dataset, but can be identified as invalid in connection with other values. Especially in datasets containing set dimensions, such values can often be found.
When examining the questionnaire-dataset we can see that much more people graduated from primary school (VSHS) than from universities (see figure 4.11). While this is not really striking, we can also brush alumnis of universities, and notice, that only a small subset of them also have graduated from primary schools (according to the data). This circumstance can be caused by different problems. Either some people did not understand that multiple boxes on the questionnaire sheet may be checked, or they were simply too lazy to provide all the details of their educational career. Another possibility is that some persons did not want to give the suggested amount of details due to reasons of privacy.
We can however assume, that all people attended some kind of primary school before going to university. In most developed countries, children are not only allowed to go to school, but they are obliged to do so. The VSHS element describes exactly the minimal amount of education each citizen has to reach. So we again can assume, that there should not be people, that refused to go to school at all (i.e. the empty bar in our diagram should be of zero height). However, there may be people that attended primary or secondary school, but did not graduate.

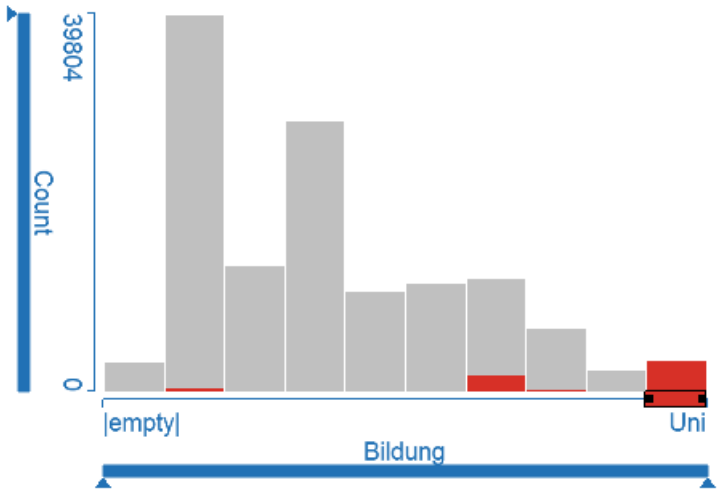


Figure 4.11: When brushing university graduates, we can see, that most of them did not indicate having graduated from primary schools (2nd bar). About half of them indicate being highschool graduates (7th bar).

Other products There are many other data records that are probably incorrect. Figure 4.12 shows a histogram displaying dog owners, along with two histograms displaying various brands of dog food. In the second and third view, all brands have been brushed combined by the operation *or*. Then, people that do not own dogs have been brushed and combined with *and*. We can see that there are more than 3000 people buy dog food, but who do not have a dog (according to the data). Because it is rather unlikely that people without dogs indicate that they are buying dog food, we can assume that many people did not indicate that they own a dog. These datasets can be (kind of) corrected, by extending dog food customers to dog owners. This will probably introduce some new errors, but correct many data records.
However, there are still some cases of data records, that are very likely to contain intentionally wrong data. Figure 4.13 shows six histograms for dogs and cats. Their contents have been filtered down to two data records. The persons filling out the questionnaire pretended to buy products of each brand of cat and dog food, both dry and canned, without having a cat or a dog at all. These data records are not to be taken seriously and instead of setting the properties for cat and dog ownership to *true*, the whole records could be discarded. There may be reasons for that, e.g., people who feed pets of friends and relatives, or people feeding dog or cat food to pets that are neither dogs nor cats, but the fact, that they buy each brand of cat *and* dog food renders these scenarios very unlikely.

4.2.3 Interesting Insights

Video-Gadgets People with higher education are more likely to own modern entertainment gadgets. This can can be observed, when brushing the DVD (DVD player) or the VCR (video recorder) bar. In figure 4.14 we can see the composition of owners of video recorders and two other electronical devices at maximum. People with low education are more likely to own a video recorder. By brushing only the the lower blocks, we set an

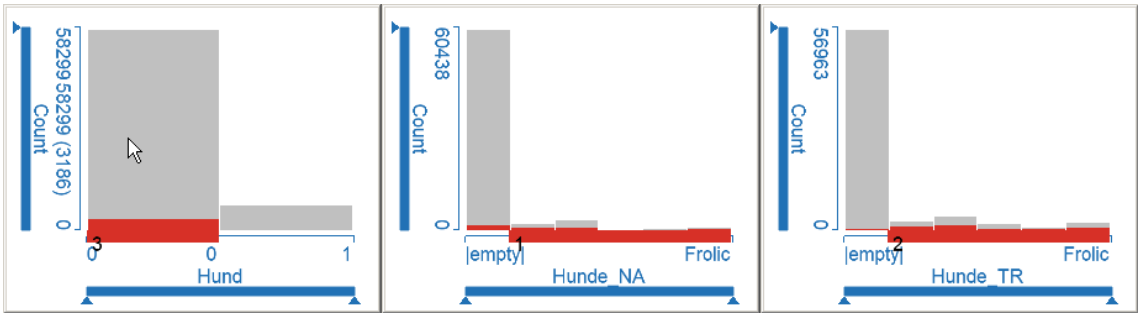


Figure 4.12: The first histogram shows a histogram displaying dog owners on the right, and people without dogs on the left. The two other histograms show customers of different brands of canned (middle image) and dry (right image) dog food. Here we have brushed people without dogs, that buy at least one kind of dog food. Apparently 3186 persons fall into this category.

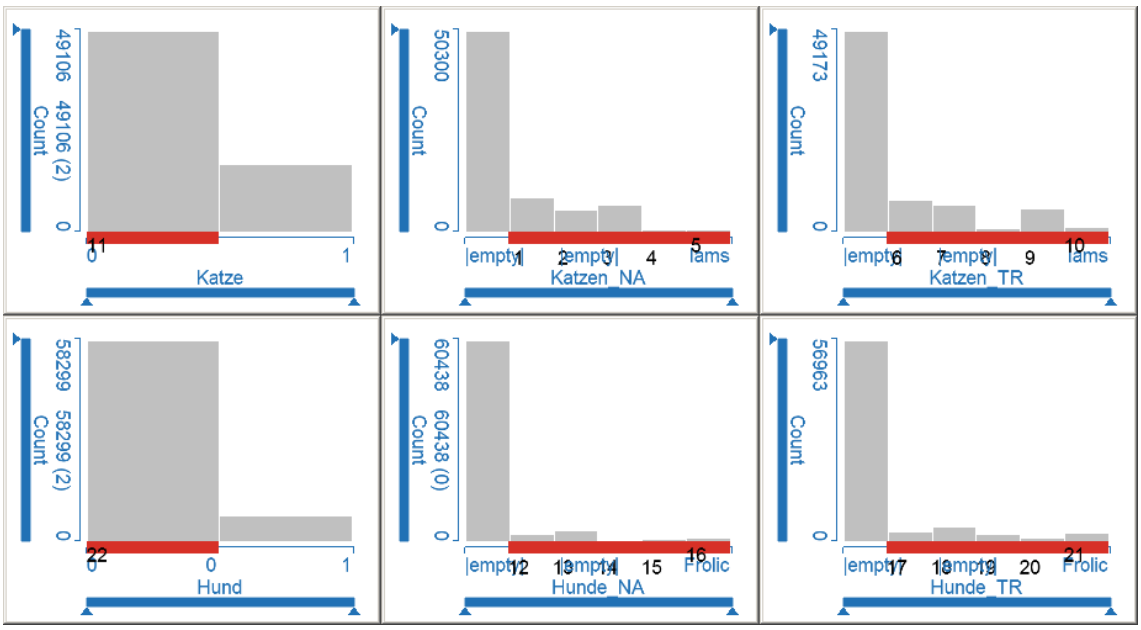


Figure 4.13: These six histograms display people that pretend to buy all different brands of pet food without having a pet. To accomplish this, every single brand of pet food has been brushed, and all brushes have been connected with the and-operator. Two persons are in this subset.

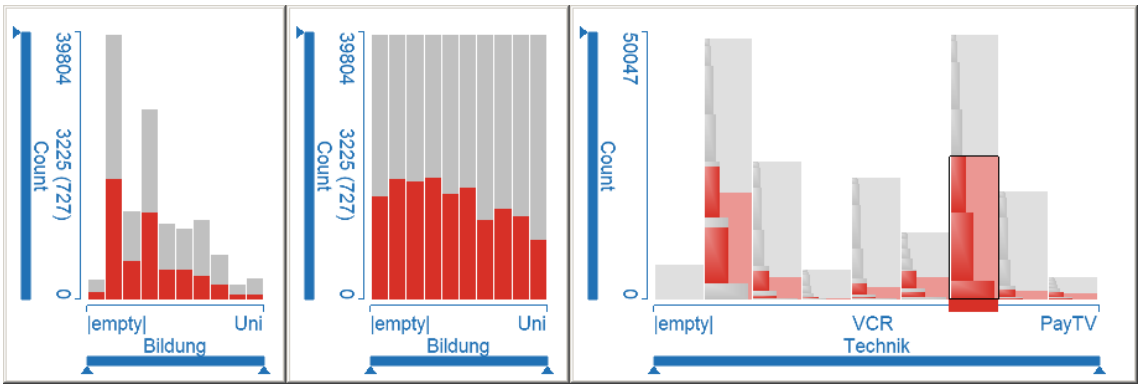


Figure 4.14: Owners of video recorders and at most two other technical devices have been selected here. The education of these people is obviously below average.

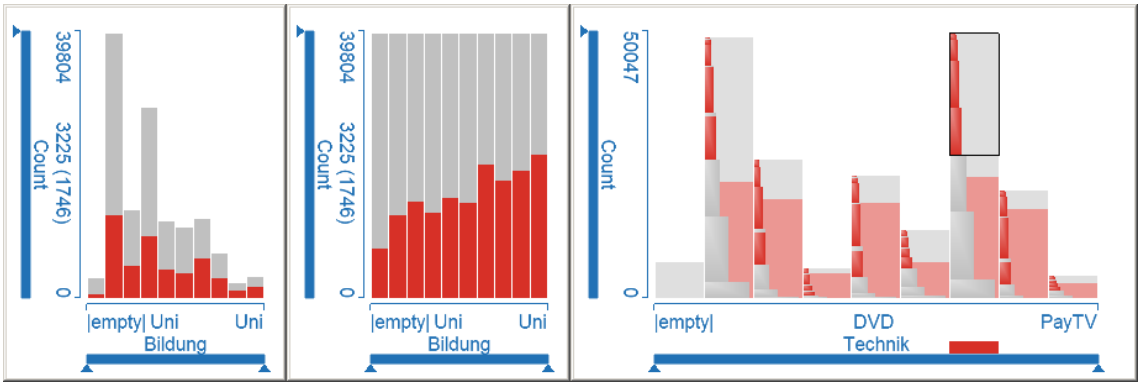


Figure 4.15: Owners of video recorders and at least three other devices are selected here. Most of these people also own a DVD player, and their education is above average.

upper limit to the number of devices in the household. That way, we exclude people having lots of devices, including a VCR, they probably use rarely. We can also see that the number of owners of DVD players is very low in this group, whereas most owners of video recorders and four or more gadgets also own a DVD player (see figure 4.15). The situation in the education histogram is reversed with this selection. People with higher education own more technical devices, which is not really surprising. When selecting people, that own at most three devices of which one is a DVD player (Figure 4.16), we can learn that these people, even though they do not have many technical devices, their distribution on the educational scale is rather uniform, with the exception of people having no education at all (first bar). Even more interesting is the fact that most of those people also own a video recorder. Having a modern device capable of playing videos seems to be connected to having a similar device of the previous generation. As we have seen before, this assumption does not necessarily work in the opposite direction. Without set-typed visualization, these analyses would require more views and more screenspace. Brushing people with a specified number of gadgets would not be possible, unless we create a separate dimension and insert precalculated values.

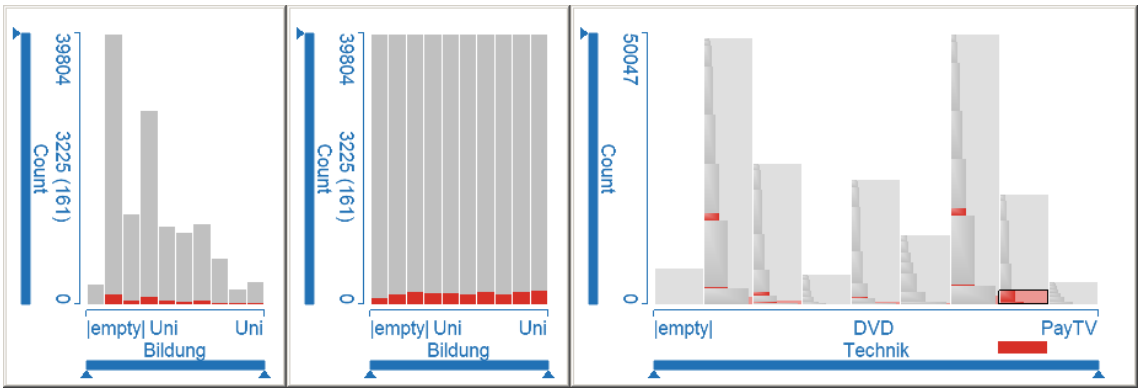


Figure 4.16: Owners of DVD players and at most two other devices are brushed here. We can also see, that more than half of these people own a video recorder as a second device. There is a slight tendency of higher educated people in this group.

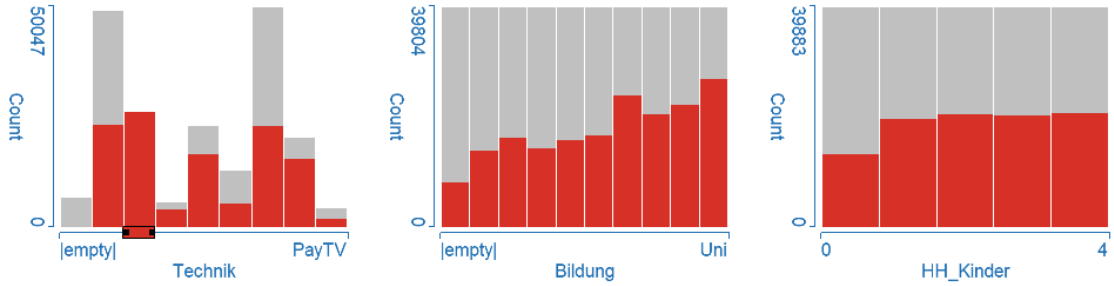


Figure 4.17: The histogram on the left displays technical devices and services, while the view in the middle gives an overview on educational levels. The brush in the left view (third bar) selects computer owners. We can clearly see, that highly educated people are more likely to own a computer. Households with children are more likely to own a computer, too.

Personal Computers and Internet Figure 4.17 shows three histogram views, one representing technical devices and services, the second one displaying educational levels, and the third one the number of children in the household. A brush has been created to highlight computer owners (third bar) in all views. We can see in the second histogram, that people with a higher educational level are more likely to own a computer. While this is not surprising, we can perform a similar task with internet users. In figure 4.18, internet users have been brushed. The histogram view displaying educational levels looks similar – highly educated people are more likely to have internet access. The third histogram showing the number of children in the household shows that while the number of children does not show a big influence on computers and internet, the presence of at least one child does. This may be due to the fact, that adults are less likely to use computers and internet, but buy computers for their children. If we take a look at offline computer users, the situation changes. In figure 4.19 computer users have been brushed, and internet users have been subtracted from them. The remaining users are about equally distributed over all educational levels (with exception of the empty set). While education does not seem to have an influence here, the number of kids has a small (maybe insignificant) influence.

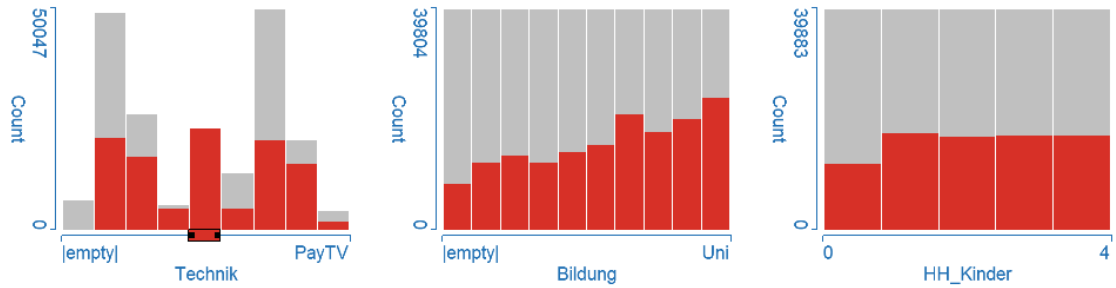


Figure 4.18: If we brush internet users (fifth bar), we get a similar impression. Highly educated people are more likely to have internet access. The presence of children seems to encourage internet usage, or children themselves use the internet, while their parents do not.

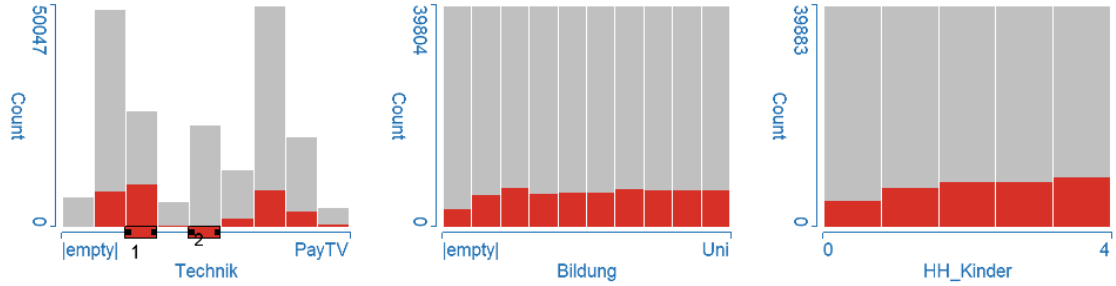


Figure 4.19: If we brush computer owners (third bar) and subtract internet users (fifth bar), the influence of education seems to disappear. Offline computer users occur in all educational levels with about the same probability but increase with the number of children in the household.

If more children live in a household, it is more likely, that there is a computer without internet access. One possible reason for this can be, that children are more likely to use a computer for offline tasks, such as playing games. Running costs for internet access can be another reason for this phenomenon. Households with many children already have lots of expenses, that can make internet access difficult to afford.

Chapter 5

Implementation

This section provides insight into related implementation aspects, dealing with loading and handling of data as well as with the implementation of the profile histogram and modifications to exisiting infovis-views. The implemented data structure is also be illustrated to give a good estimate on memory consumption and running time for searching the dataset. The implementation has been done in the ComVis framework [1], i.e., a general purpose visualization application developed at the VRVis research center in Vienna. ComVis uses OpenGL [4] for hardware acceleration and supports multiple coordinated views.

5.1 Data Input and Output

ComVis supports three kinds of files for storage of data and settings. The simplest supported file format is the CSV format. CSV is often referred to as "comma separated values" or "character separated values", because data items are saved in textual formats separated by a special character. In most cases this character is a comma, but it is often replaced by a semicolon, if the comma is used as the decimal separator. Other frequently used separators are the semicolon, the tabulator, and the space character. ComVis supports automatic selection of the separating character as well as the decimal separator. The CSV format is not standardized and is therefore not used for data types consisting of a variable number of values, like sets or time series. There can be a header line specifying the name of each dimension, and a data type descriptor line to identify its type. A heuristic algorithm is used to figure out, if a line is used as a header, a data type descriptor, or a simple data line.
ComVis also supports the ComVis file format, which is an extension of the CSV-format. This format allows the storage of time series and sets. While time series data is appended to the end of the tabular structure after an identifiying string, set data is inserted directly into the data table. A valid value in a set column is a string containing different set elements, that is still valid if it is empty. Elements are separated by a special set separator, which is normally "|", but can be overridden in the file's header. When reading ComVis files containing sets, it is not necessary to provide a list of possible elements, because it is built during the loading process. However this strategy leads to unsorted lists of elements, that have to be reordered by the user. When the loading process is done, the elements are in the same order, they were read from the file. The ComVis file format is a safer way for data exchange, because it allows setting the separator character by the optional

directive "#separator". Because sets have to be separated too, it is also possible to set a set-separator by using the "#setseparator" directive. Otherwise, the set-separator is assumed to be "|". It is also necessary to supply datatype information for each dimension, otherwise the file is not valid. This is done by adding a line containing datatype shortcuts like "NUM" for numeric values, "CAT" for categorical values or "SET" for sets.

```
#ComVis 2.0 input file
#separator ,
#setseparator |
#comment: this is a test dataset
CAT,SET,NUM
alpha,red,3
bravo,blue|green,5.4
charlie,,0.33
alpha,green|blue|red,2
```

Another file format supported by ComVis is the ComVis View format. This format does not only store data, but saves a complete workspace into a binary file. With most new features, the format of this file has to undergo slight changes, and may be unreadable in older versions of ComVis. This file format is also not easily editable for the end user, and cannot be imported into other programs. On the other hand, loading and saving these files is very fast, because numbers do not have to be parsed from textual representations, and it is very useful for people working with the same data set. Saving set data to this file is very easy because the data structure can be saved, like it is stored in memory.

5.2 Data Manager

The ComVis visualization system stores the currently opened dataset in a so-called *Data Manager*. This class provides functions for loading and saving csv-files, ComVis-files and sessions in a binary format. The Data Manager contains a list of columns, each storing one dimension of the dataset. While storing a list of numeric or categorical values is done by a simple dynamic list, sets need a more sophisticated approach. To decrease further complexity in the Data Manager, storage and handling of sets has been implemented in a separate class, the Set Item Table.

5.2.1 Set Item Table

The Set Item Table is a class managing set data. It keeps two synchronous lists, a list of records and a list of elements (see figure 5.1). The record list keeps track of all records in the dataset, each containing a list of its elements. The element list contains one entry for each possible element in the set. Each element is represented by a name and a list of records that use this element. To ensure integrity of the data structure, both lists are encapsulated by the Set Item Table, which offers methods for adding records. To do this, the record has to be added to the record list, and each of its elements has to be added to the element list, if it does not exist yet. Then, references from these elements to the new record have to be created, and the elements indices have to be added to the record's private element list.

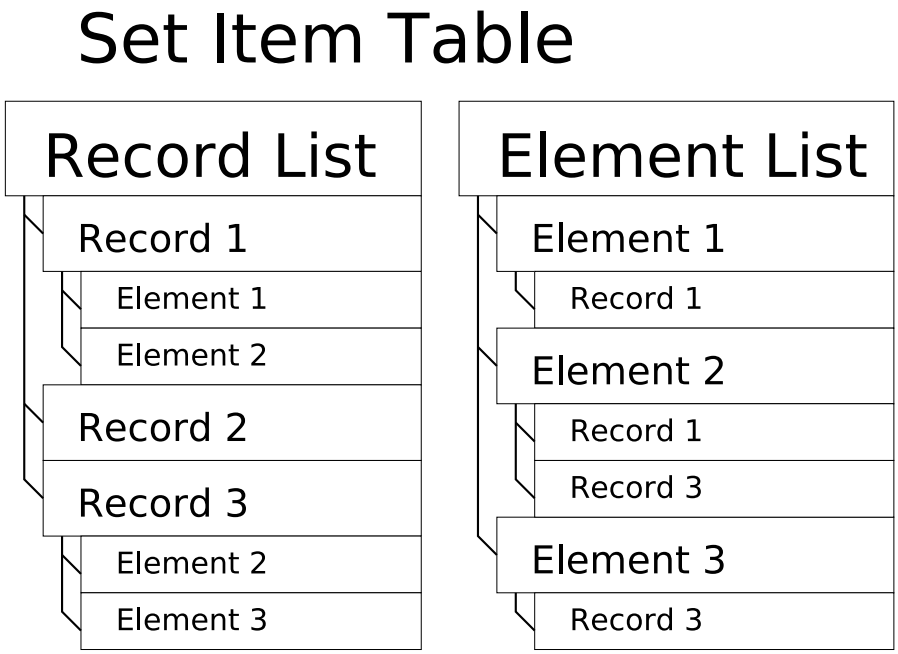


Figure 5.1: Structure of the Set Item Table. Information ist kept redundantly in the data structure to allow fast per-record and per-element access.

Sorting Since both, the record list and the element list use indices to reference elements and records, sorting is nontrivial. Therefore a reordering function has been implemented to allow sorting elements alphabetically or by a user-defined order. While the order of elements has a significant impact on the visualization, the order of records is not important, because the records' indices are not used in any visualization view. The element sorting method takes a list of strings (representing all elements in the desired order) as a parameter and creates a new list of elements in the correct order. Then, each element's record list is copied from the old element to the new element with the same name. Then the old list can be deleted as well as the record list. Since both, the record list and the element list, contain all necessary information of the dataset, a new record list with correct indices can be created from the element list.
If we have n possible elements in our set-typed dimension and m data records in our dataset, each element can be assigned to all data records, so we have a maximum of $n * m$ assignments. In our implementation the empty set is used as a special element, so we must have at least one element. We also consider having datasets with at least one record. Because every record has at least one assigned element, and each element is at least used once, the minimum number of assignments is $max(n, m)$, while the maximum number of assignments is $n * m$. When the record list is rebuilt, all assignments have to be copied to the correct data record, so the running time equals the number of assignments. Creating a new element list in a user-defined order takes n steps. Assigning the correct record lists to the new elements takes approximately $\frac{n^2}{2}$ steps, because each element needs to be looked up in the old element list. A binary search is not possible, because the list is not

alphabetically sorted.
The running time for reordering a worst case dataset is the sum of the number of steps it takes to reorder the element list and the number of all records and their elements in the record list. The running time for the worst case is then limited to $O(n^2 + n * m)$. We can neglect the term n, because it is dominated by n^2, but we cannot neglect $n * m$, because m can be larger than n (and in most cases it is significantly larger). In most datasets m greatly outnumbers n, because we have lots of data records, with few elements. In these cases, $n * m \gg n^2$ applies, and the running time is $\Theta(n * m)$. Sorting all 16 elements of a set-typed dimension containing about 90000 data records takes about five seconds on a Pentium-M Processor running at 1733 MHz.

5.3 Profile Histogram

In ComVis, the histogram and the scatterplot view share a common superclass for various common tasks and user interface features. This superclass implements two axes to support labelling and to perform coordinate transformations from data space to image space. Also, capturing mouse events to assemble rectangles, used for brushing, is a common task done by the superclass. However, the interpretation of which data records actually are brushed, must be done by the currently active view.
Since the Profile Histogram is very similar to the traditional histogram, it was implemented using the common superclass mentioned before. The rendering process of the view is very similar to the histogram. First, the semi-transparent bars have to be drawn for each element, then the opaque blocks are drawn over them. See Chapter 3 for details on how and where the blocks are placed. The cushion-effect, which was inspired by cushion treemaps by Jarke van Wijk [52], is a simple color gradient. In contrast to cushion treemaps, there are no hierarchical levels that can be reflected by the blocks' colors. Instead, the gradients are only used to emphasize the blocks' borders without the use of lines. The effect is a simple interpolation done in hardware with OpenGL using a brighter color in the top left corner.

5.3.1 View Draw Data

In ComVis, each view has a corresponding data class, containing the *View Draw Data*. This class holds data from the data manager in an optimized structure to reduce direct accesses to the data manager and make rendering of the view fast and easy. To render a Profile Histogram, we do not need information about every single data record, we display bars and blocks representing accumulated numbers of records with similar characteristics. The appropriate data structure for this purpose stores one object per bar and one object per block.
The most basic element in the data structure is the *block*. Each block represents data records having the same number of elements in their sets. To minimize calculations, the block's bounds are stored in device coordinates, which are ready to use for rendering. When resizing a view or switching to a different scaling mode, these values have to be recalculated. This can only be done, if the number of elements in this block is stored, too.
The next element in the hierarchy is the *bar*. In addition to a list of blocks, this object stores the postion and number of represented elements of the whole bar. The number of elements in a bar equals the sum of its blocks' elements, except for the bar representing

the empty set. This special bar does not contain any block.
The View Draw Data object contains a list of bars and a list of brushed bars. Brushed bars have the same structure as normal bars, but they save the amount of brushed data records per bar as well as per block. To fully support multiple combined brushes, the amount of brushed elements has to be stored for brushes combined by and, or, and op (which allows choosing different operations for each brush).

5.3.2 Mouseover Effect

The mouseover effect emphasizes parts of those blocks in the view, that belong to the block which is currently below the mouse cursor. The information, which parts of which blocks should be emphasized, is stored per-block. Every block has a list of blocks, that share data records. This list is precalculated when the dimension that should be displayed in the histogram is selected.

5.4 Histogram

The histogram view of ComVis has been used as a base for the Profile Histogram. Therefore most of the differences have already been mentioned before. In contrast to the profile histogram, this view has to work with numeric, categorical and set data. When a dimension of the data type "set" is selected, the histogram has to react differently to user inputs and provide slightly different feedback to the user.
The first difference is, that data items are represented by one bar per element they contain. Therefore the sum of all bars' heights does not correspond to the number of data items, but to all elements assigned to all bars. The other difference is the special handling of the empty set. It is represented by a semi-transparent bar to remind the user of the fact, that these items would not have a representation at all, if there was no special handling for them.
When creating the View Draw Data for the Histogram, there is no increase in running time due to the data structure of the data manager. Since we have a list of data records, and a list of elements, we can simply access the list of elements, which contain references to the data records, and return the size of the reference list. This size can then be used to determine the height of a bar.
Like in every view that can handle sets, brushing is a bit different. In a histogram, the user can brush a range of bars. These bars represent elements, that have been assigned to data records. When element a is brushed, some selected data records can contain element b as well - so we have to highlight a part of the histogram bar representing element b. This is done in the data manager, by using element lists as well as record lists. First, we have to iterate through all elements, whose bars have been brushed. Then we iterate through all data records that have been referenced in the element lists. Each one of these records must contain one of the elements we have brushed, but it can contain elements that have not been brushed. Those bars have to be highlighted too. So after we have a list of all brushed data records, we can iterate through all of their referenced elements, and add them to the brushed bars. After that, we have the number of selected items for every bar in the view. Figure 5.2 illustrates brushing in a histogram.

In a setup with multiple combined brushes, we have a list of brushed items for every brush and another list for the combined result. After adding a new brush, we compute the

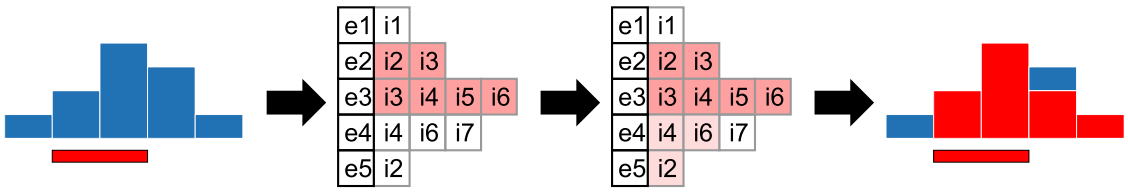


Figure 5.2: Brushing in a histogram using a set dimension works like shown here. Two bars are brushed in the first image. The second image shows the elements (e1 - e5), and their associated data records (i1 - i7). The light red records are brushed, even though they are not within the brush geometry. The last picture shows the final image of the view. Parts outside of the actual brush have been highlighted, accordingly.

result by combining each record's brush values in all lists. Consider having two brushes, and a record is brushed in one of them. If the operation "or" is used, then this record will be selected in the result list. If we use "and", it will not be selected. When the final brush is rendered, we treat the result like a single brush. We iterate through all brushed data records, and sum their elements up. Then the highlighted bars can be drawn.

5.5 Scatterplot

The 2D-Scatterplot is a very important view in ComVis, so it was definitely necessary to extend it for the display of sets. Generally most aspects of histograms also apply to the scatterplot, but some special considerations have to be taken. First, each of the two available axis can be used to display a set-typed dimensions, even both of them.
However, points in scatterplots are managed differently from bars in histograms. In most scatterplots there are lots of points covering each other. Therefore, points can be scaled to reflect the amount of congruent points. In case of categorical or set-typed dimensions, the amount of points could easily be calculated by counting the number of dataset per category or element. Numeric dimensions are somehow different here. Dots can can be placed anywhere in the valid range without big steps between them. To find out, whether two points have to be drawn at the same position, it is necessary to calculate their position on the screen. Points with slightly different coordinates can be projected onto the same pixel in the display area. If the view is resized, it is possible, that these two points map to different pixels. Therefore it is necessary to know the resolution of the display area to determine the amount of congruent points on a specified position.
To solve this problem, the coordinates for all single points are calculated first. For every virtual screen position, an entry in the coordinate-list is created, containing the amount of data records it represents. This number is then used to determine the point size (see figure 5.3). When using categorical or set-typed dimensions, points with different coordinates at the same pixel can only occur, if the number of categorical values is higher than the width or height of the display area, so in most cases, large points in the view represent exactly the same category. The situation changes with the use of sets. A data item of a set-typed dimension can be represented by multiple points in the view. If we want to represent each data item equally, we can represent its elements by fractional points. We cannot draw fractional points, but if the number of points for a specified coordinate is

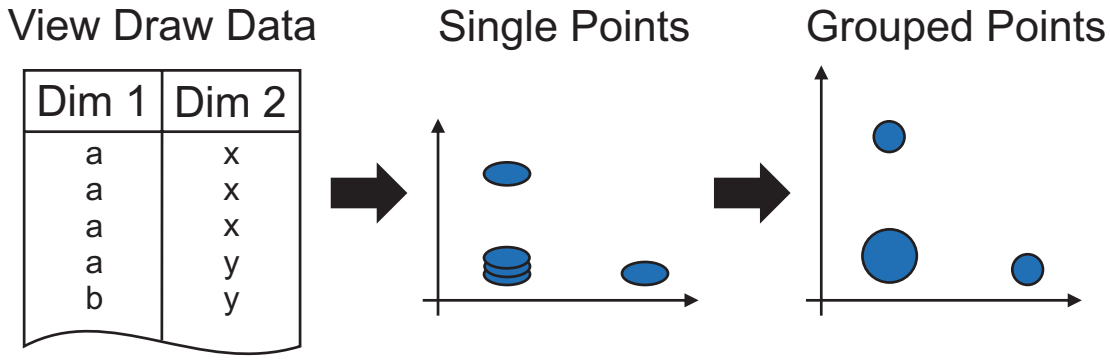


Figure 5.3: Multiple points at the same position receive special handling. First, a list of points is created from the raw data. Then the points' coordinates on the screen are calculated. If there are multiple points on the same position, a larger point is drawn, instead of repainting it multiple times.

summed up, it is possible, to add the fractional values. This avoids overrepresentation of data records with many elements.
To build a list of all points we have to iterate through all elements of all data items, we iterate through all data items and construct a list of points that have to be drawn. Each point's position is calculated by querying its value in the two selected dimensions. If a set-typed dimension is used, the record's elements are iterated, and multiple points are created. In case of two set-typed dimensions, the number of points is the product of the element counts in both dimensions. Again, points at the same screen position are summed up, and these summed up points are then used for rendering the view.
When brushing a scatterplot with two set-typed dimensions, the brushed range of elements on the first dimension is iterated. Then, the list of referenced data records is iterated, and if a record also contains at least one brushed element in the second dimension of the scatterplot, it is marked as brushed in the brush list. If there is only one set-typed dimension, we can save the last iteration of the algorithm before.
When rendering a list of brushed elements, we can treat this list exactly like the list of all elements. First, all virtual points on the screen are created, and their positions on the screen are calculated. Then a list of summed up points is created for every position occupied by at least one point. The weight of each point is summed up, and responsible for its radius in the final image.

5.6 Parallel Coordinates

The Parallel Coordinates view, which is mainly based on work by Inselberg et al. [21] has also been extended for the use of sets. If a dataset containing set-typed dimensions is displayed in a Parallel Coordinates view, there are some special cases we have to keep mind, when extending this view to the display of sets.
The main difference in a Parallel Coordinates view is, that lines can split up at set-typed axes. For every element that is assigned to a data record, one intersection at the corresponding axis is necessary. Therefore we do not only need to store one crossing position per line and axis, but a list of positions. When the data is displayed, we have to iterate

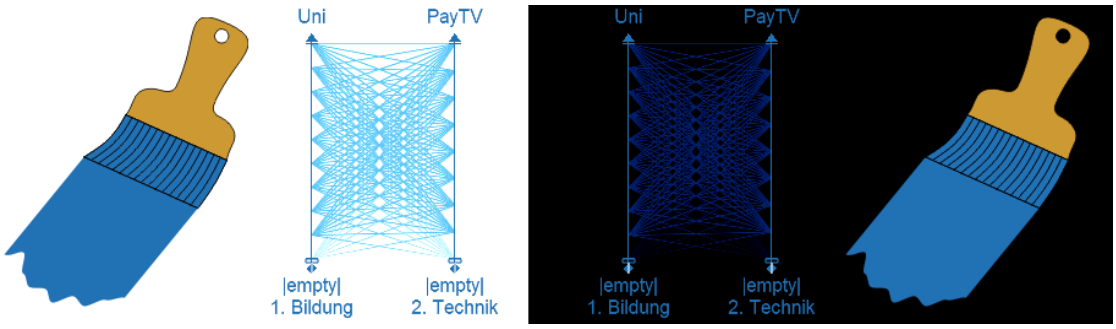


Figure 5.4: The line color in this view is set to (33,113,181) in the RGB color model. The paintbrushes show, how the color should look like on white and black backgrounds. Due to the low opacity of less than 1%, the colors change. If the red component of the line color is multiplied with a lower opacity than $frac166$ it is rounded to zero. No matter how many lines are drawn at the same position, if the background is black, the red component will always be 0. On the white background, a low opacity has a big impact on the green component. Not enough green is subtracted from the white background color, which results in a color similar to cyan.

through all positions and draw multiple lines. If we have two neighbouring set axes, we have to connect each intersection of the first axis with all intersections on the second axis. The maximum number of line segments between two axes is the product of the two axes' element counts. Apart from simply drawing more lines, the new method is also more complex. Before the addition of sets, a line consisting of multiple connected segments could be drawn with OpenGL by using a *GL_LINE_STRIP*. This method takes $n + 1$ vertices and connects them with n line segments. Now we have to use *GL_LINES*, which takes $2 * n$ vertices and connects them pairwise, by drawing n lines, that do not have to be connected. So if we have at least one set-typed dimension in a parallel coordinates view, we have to draw separate line segments. And because every vertex is submitted by a single OpenGL call, we have nearly twice as much calls, even if the number of line segments does not increase (i.e. only one element is used by each data record in each set-typed dimension). Another problem, that occurs with the increasing amount of line segments, is the insufficient precision of the framebuffer. In the current implementation, only eight bits are available for every color channel to maximize compatibility for older graphics hardware. In a dataset with many records, a higher transparency value is required to get a detailed insight, but if the opacity of the lines becomes too low, colors can suffer significantly. When using sets, the number of lines increases, the opacity is decreased even more, and the resulting color changes (see figure 5.4). Brushing in a parallel coordinates view is as simple as in a histogram. It is only possible to brush a range on a single axis, and by iterating through selected elements, all data records referenced by these elements can be marked as brushed. When redrawing the view, we simply convert the list of brushed data records into lines and render them in a different color.

Chapter 6

Summary

Because the vast amount of information stored on computers continues to increase, it becomes more and more difficult to analyze without visual representations. Information visualization is the scientific research area that tries to solve this problem by providing adequate techniques to visualize large datasets. The type of the data to be visualized plays an important role in finding the right visual representation. This work proposes visualization techniques for the data type "Set" that allows each data record to have multiple values in one dimension. This data type needs a special data model and adapted visualizations to be allow detailed analysis of set-typed data. Analyses with two datasets will show, that set-typed data is easier to handle can be visualized very efficiently.

6.1 Introduction

The field of visualization is the part of computer graphics, that deals with the transformation of data into visual representations to support the exploration and analysis of the data as well as the presentation of results. Depending on the underlying data we distinguish between volume visualization, flow visualization, information visualization and various other types of visualization.

Information Visualization This work deals with information visualization, a research area, that visualizes informational data from a variety of sources. The main difference to other types of visualization is that there is usually no inherent and dominant spatial information. Visualization techniques for these abstract datasets can be manifold, and heavily depend on the dataset and purpose of the application. The number of dimensions to be visualized can be relatively high, too, therefore many tools allow the user to create multiple views and select different dimensions. General purpose information visualization tools include XMDVTool by Matthew Ward [53] or the InfoVis Toolkit by Fekete [14] which can be used to build new applications.

The Datatype "Set" Up to now there are no published visualization techniques for the datatype "Set". However, dimensions containing set data can be seen as categorical dimensions - the only difference is that each data record can contain a set of multiple categories. We can also convert multiple categorical dimensions containing few elements to a new set dimension. Sets also behave similar to categories in data handling and the

resulting visual representation. So while there are some basic differences, that need special handling, most techniques for categorical data are also suitable for sets.

6.2 State of The Art

This section deals with various techniques that are already published and had an influence on the work presented in this document. Since this is a new topic, most of these publications deal with the visualization of categorical data.

Focusing & Linking In this work, the visualization application ComVis has been extended to handle the new data type "set". Applications like this one implement a technique called "Linking and Brushing" or "Focusing and Linking" [12], which allows the user to select data items in one view, and highlights the same items in all other views too. Most tools providing multiple views, like XMDVTool by Matthew Ward [53], also implement this feature.

Parallel Coordinates Parallel coordinates, as proposed by Inselberg et al. [21] are a very useful tool to visualize a relatively high number of numeric dimensions. A point in two- or three-dimensional Cartesian space is represented by a polyline in parallel coordinates. The polyline intersects each axis at a position, indicating the values of its coordinates. While parallel coordinates are great to visualize several dimensions, neighbouring axes are easier to analyze. Therefore, the user should always have the possibility to reorder axes. Because parallel coordinates are very susceptible to cluttering, drawing semi-transparent lines can improve the visualization significantly.

Scatterplot The scatterplot is a Cartesian grid, displaying two or three data dimensions. Each data item is represented by a glyph, that is positioned in the Cartesian space according to its values in the three dimensions. The scatterplot is very popular in information visualization, because it is easy to read and helps finding correlations between two dimensions. The scatterplot matrix by Seo et al. [43] helps finding possibly interesting combinations of dimensions. In this work, a scatterplot had to be extended for the use of sets.

Parallel Sets While parallel coordinates are great for numeric data, they are not as well suited for categorical data. The less categories a dimension contains, the more lines are painted exactly at the same positions, which makes it hard to estimate the number of congruent lines. Another problem is, that it is hard to arrange categories at the right position on each axis. This problem has been addressed by Kosara et al. [30]. They proposed parallel sets, a technique, which draws bands across parallel coordinates, that can split up at each axis. The widths of these bands represent the number of data items. By assigning different colors to the bands, they can still be traced after splitting up.
Keim et al. [28] proposed a technique called Pixel Bar Charts, which are histograms whose contents provide additional details on each data item. Each pixel represents one data item and its color can be used to reflect a numeric or categorical value. The idea of providing additional details inside the bar of a histogram has been picked up during the design of the profile histogram.

6.3 Sets

To support sets in a general purpose visualization application, various parts have to be changed. Besides adding a specialized view for easy exploration of sets, also some other views have been extended to make the display of sets possible. A converter for existing categorical datasets was implemented as well.

The Problem Since many datasets contain categorical data, visualization techniques targeting this kind of data are already available. However, the data stored in categorical dimensions can often be represented more efficiently in a set. For example, in a dataset containing patients and their symptoms, the natural way to represent symptoms would be a list of strings for each patient. Up to now, such data has mostly been stored in multiple binary dimensions. Each dimension represents a symptom, and can contain the values "true" and "false". Visualizing a single set dimension can be much more efficient, than visualizing multiple binary dimensions.

Sets - The Basic Idea A set is usually a collection of multiple elements, in our case textual attributes. If we use set as the datatype for a dimension, each data record has to contain a valid value in this dimension. A set-typed domain is defined by a collection of possible elements, and each data record can contain an arbitrary collection of these elements An empty set is also a valid value, but it needs special handling in various visualization techniques.

The Profile Histogram We introduce a view specialized for the exploration and analysis of sets. This view, the profile histogram, is an extended histogram, that provides some details on the distribution of elements inside a set. Instead of trying to visualize multiple set-typed dimensions, we tried to provide more details about one dimension. The main view of a profile histogram consists of bars and blocks. In contrast to a traditional histogram, bars are divided into blocks, that show more details of the underlying data. Each bar in the histogram represents a single possible element in the set-typed dimension. Because a single data record can contain multiple elements in its set, it is also represented by (parts of) multiple bars in the view. That way the used amount of screenspace is not constant for each record, but only for each element-to-record assignment. This can be changed by using blocks of variable width inside the bar. A data record containing two elements is then represented by two blocks of half width inside different bars. This model also works with data records containing more (up to all) elements. However, since it is a histogram, there is no way to distinguish individual data records. We have introduced a visualization method, that allows to distinguish records with different numbers of elements.

Scaling Modes The profile histogram supports different scaling modes. In the normal mode, the highest bar in the view is always as high as the display area. All other bars are scaled accordingly. This is the default mode which is also used in traditional histograms. This view is good to compare the sizes of bars and blocks to each other. In scaled mode, the window height represents the total amount of data records in the dataset. Each bar's height is scaled accordingly. In fact, this view is a scaled down version of the normal mode.

It is useful, when the number of records in a bar has to be compared to the total amount of records. The last mode is the relative mode. In this mode, each bar is scaled to the window's height. This mode is very useful to compare the size of blocks inside a bar, or to get a closer look at very short bars.

Brushing Because the profile histogram introduces blocks as new visual items, it makes sense to brush these individual elements. So in contrast to the traditional histogram, the y-dimension of a brush is important, too. The three scaling modes we have shown before, require additional attention when dealing with brushes. A single brush created in full height by dragging a rectangle, can consist of multiple clusters that are not visually connected, if we switch to one of the other modes. Similarly, if we brush in normal or scaled mode, we can create a brush, that would be impossible to create in relative mode. Another challenge when using brushes on set dimensions is the fact, that not all visual elements have to lie inside the brush. If a data record is brushed because of one of its elements, it may still have other elements, that were not brushed, but have to be highlighted as if they were. This leads to the situation, that there are brushed elements outside of the actual brush geometry.

MouseOver Effect The mouseover effect is a simple feature aimed to make data exploration of sets easier. As mentioned before, there are bars and blocks in a profile histogram. Blocks with a width smaller than the bar's width represent data records that have additional elements. However, there is no way to see, which elements are also part of these records. The mouseover feature provides this information, by highlighting parts of all blocks that contain related elements. By simply moving the mouse cursor over a block in the histogram, each block in the histogram, that represents the same data record, is partly highlighted. This feature is like a simple brush that works only with single blocks, and does not highlight selected data records in other views. But it works in realtime even for large datasets, and does not destroy currently active brushes.

Scale Sliders As mentioned in the state of the art report, zooming and panning is an important topic in information visualization. If there are too many visual elements in a profile histogram, their size can become very small. It can be hard to brush them or even see them. The scale sliders provide an easy solution to this problem, by allowing the user to zoom in, and move the zoomed viewport around. This functionality is common to most views in ComVis, so it was also used in the profile histogram.

Data Conversion To visualize datasets that contain set-typed data as multiple categorical dimensions, a converter is needed. There are two types of conversions: binary conversions and general conversions. A binary conversion can be performed, if a categorical dimension contains only two values, like, e.g., "true" and "false". If a data record contains the value "true" in this dimension, it receives an element in its set, otherwise it doesn't. A data record containing only "false" values in its categorical dimensions, contains only the empty set in the set-typed dimension after the conversion.
General conversions can use categorical dimensions with arbitrary numbers of elements as an input. The user can select categories to be added to the set-typed dimension, and each data record containing such a category receives the respective element in its set. If

is also possible to map different categories to one and the same element, or add multiple elements, if a data record contains a special category.

Adaption of Existing Views Since ComVis already supports multiple views, some of them have been extended to allow the use of sets. This includes changes in the underlying data structure to allow multiple elements per data records, as well as adapting the brushing mechanisms. To emphasize the empty set, its visual representation has been altered.
The histogram is the most simple view to display sets. Each element is represented by a bar, and a data record contains multiple elements, it is represented by multiple bars. When brushing is used, there can be brushed bars outside the brush rectangle, due to the nature of sets.
The scatterplot has been extended to allow using sets in both dimensions. Elements are treated similar to categories, and are aligned along the axes. A data record containing multiple elements in its set, is represented by multiple points. Since they are not connected, it is not easy for the user to spot related elements. If a brush is applied to the scatterplot, all data records containing a brushed element are selected. These data records can contain additional elements that have not been brushed, but have to be highlighted as well, even if they lie outside the brush geometry.
The parallel coordinates view has also been adapted to display set-typed dimensions. The main difference to traditional dimensions in parallel coordinates is that a line representing a data record has to intersect a set-typed axis at multiple positions. If a data record contains multiple elements in its set, there has to be one intersection for each element. If a line should intersect an axis at multiple positions, it has to split up at the axis before into multiple lines. These lines intersect the set-typed axis at multiple positions and are joined at the next axis. If there are two consecutive set-typed axes, one line segment has to be drawn from each intersection at the first axis to each intersection of the second one. To reduce the overrepresentation of data records containing many elements, the opacity of these lines can be reduced.

6.4 Demonstration

To demonstrate the usefulness of set-typed visualization, two datasets containing set-typed data as multiple categorical dimensions have been prepared.

Primary Biliary Cirrhosis This dataset contains blood measurements and symptoms of about 400 patients. Many of them were suffering from liver diseases. While blood measurements are classic numeric values, symptoms are binary dimensions and can be converted to a single set-typed dimension. This dimension can be analyzed using the profile histogram, to find correlations between specific symptoms and other attributes provided in the dataset.
The profile histogram provides some interesting details on the distribution of elements in the sets. Figure 6.1 shows an overview of all patients' symptoms. Without brushing we can already see, which symptoms occur often, and which do not. We can also see, that about 50% of all patients with a tumor do not show any other symptoms. When using a second view displaying the patients' gender information, we can look for differences in symptoms between male and female patients. While we could already use a traditional

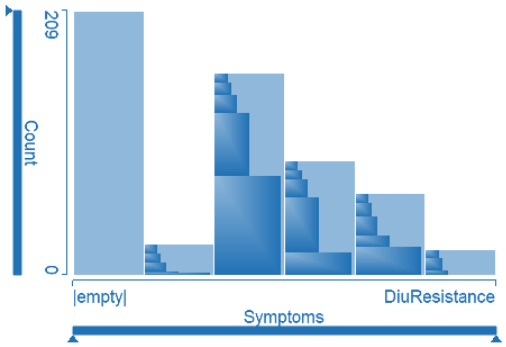


Figure 6.1: This profile histogram shows the distribution of the elements of the set-typed dimension *Symptoms*. We can easily estimate the probabilities of symptoms. The height of the full-width block in the third bar (which represents patients suffering from hepatomas) tells us, that about 50% of all patients with liver tumors do not show any other symptoms in this dataset.

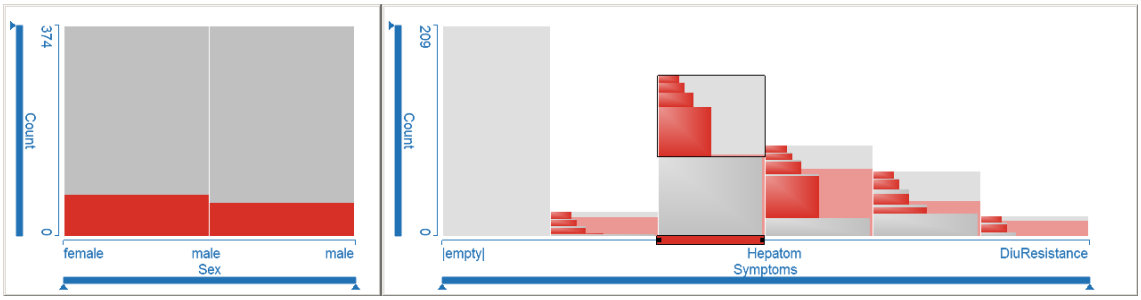


Figure 6.2: Patients suffering from hepatoma and at least one other symptom have a slightly higher probability to be female (in this dataset).

histogram to find out, if men or women are more likely to suffer from a certain symptom, we can now analyze combinations of symptoms. Figure 6.2 shows, that hepatoma patients suffering from multiple symptoms are a bit more likely to be female (which may not be significant), while suffering from hepatoma without showing other symptoms is a male domain.

The Questionnaire Dataset This dataset contains information from a questionnaire filled out by about 65000 people. The questions cover topics like income and education, as well as shopping habits and the use of technical devices and services. This dataset contains several groups of binary columns, that can be converted to set dimensions. This time we do not simply put all binary dimensions into one set, but create several separate dimensions for education, technical devices, etc.
A very interesting point in data exploration is the search for erroneous data. Especially data provided voluntarily by people can contain many errors or missing values. As an example we can show, that there are two people, that pretend to buy all different kinds of dog and cat food, but deny even having a cat or a dog (see figure 6.4). Such data records can be removed from the dataset, because there is a high risk, that other dimensions

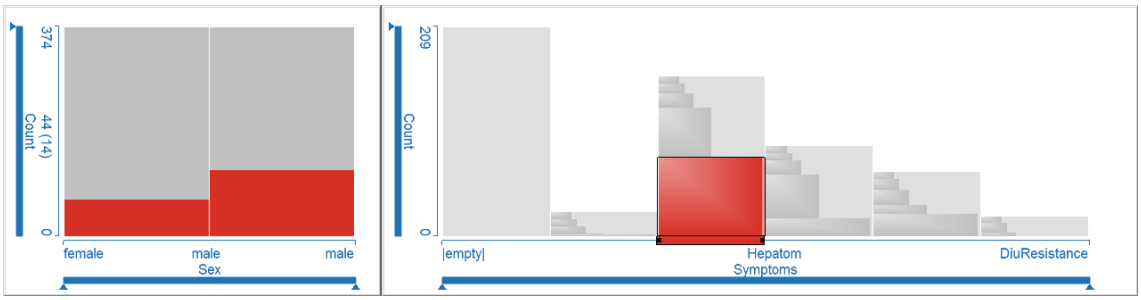


Figure 6.3: Patients suffering from a hepatoma without showing any other symptoms have a higher probability to be male.

contain less obvious errors.

6.5 Implementation

Data Input and Output ComVis supports loading and saving of several file formats. In contrast to the traditional CSV format, that cannot handle sets, an extension of the CSV format, called the ComVis file format supports set dimensions. If a data record contains set elements, they are concatenated by the set separator "—". If another separator is desired, it can be defined by using the "#setseparator" directive. ComVis also supports a session format, which saves the current dataset, along with the current configuration of views and brushes. Saving another data structure in this file is straight forward, because this format can only be read by ComVis.

Data Manager The data manager is the central module designed to keep a dataset in system memory and provide data access methods for views and brushing functions. To make the handling of set data fast, it is organized in a redundant structure. A list of records contains lists of elements for each record. Each element also contains references to all records that it is assigned to. A list of elements also contains these elements. The list of elements provides quick access, if we are interested in element lists, for example in histogram views. In other cases, if we want to get a list of elements for a specific data record, we can use the record list.
A sorting algorithm has also been implemented. The user can reorder elements in a list, and the data structure is rebuilt. Reordering is performed on the element list only, and afterwards, the record list is recreated.

Profile Histogram The profile histogram has been implemented using a common superclass of the histogram and the scatterplot in ComVis. This superclass deals with features that are common to these views, like axes, scale sliders and mouse events. The rendering process of the profile histogram is also very similar to the traditional histogram. The blocks are rendered with a brighter vertex at the top left corner to make them look (kind of) round and make it easier to see the blocks' borders without separating lines.
In ComVis, each view has a dedicated data structure to store information in a way that is best suited for fast and efficient rendering. For profile histograms, this view draw data consists of a list of bars, each containing a list of blocks. The number of data records

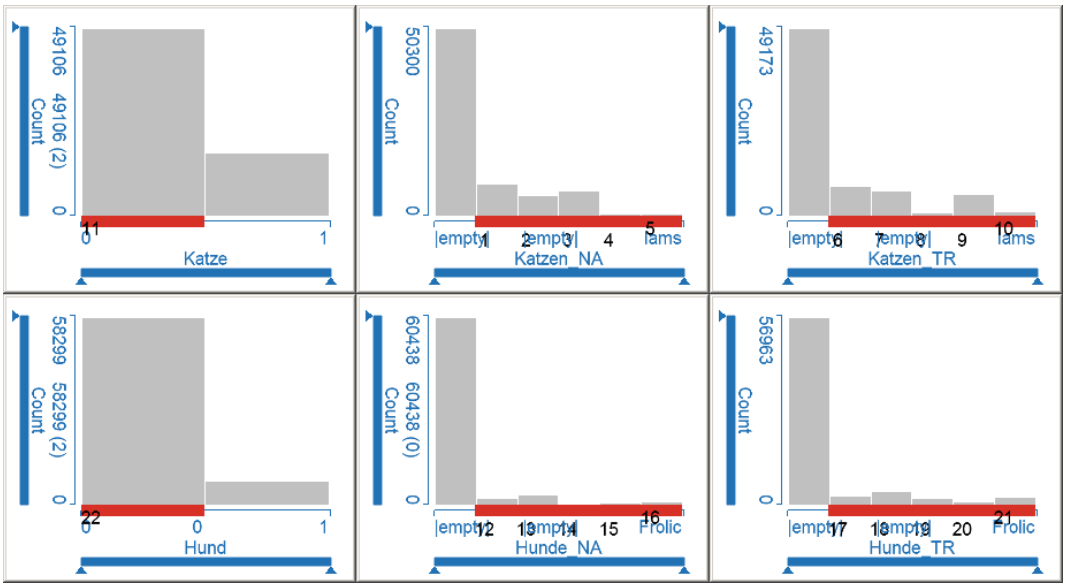


Figure 6.4: These six histograms display people that pretend to buy all different brands of pet food without having a pet. To accomplish this, every single brand of pet food has been brushed, and all brushes have been connected with the and-operator. Two persons are in this subset.

is stored for each block and each bar. A similar data structure is created for brushed elements as well.

Histogram Changes to the histogram view to support the visualization of sets, are minimal. When building the view draw data for sets, the data has to be acquired from the set item table. Although bars can not be partly brushed by the user, like in the profile histogram, there is still the possibility of partly brushed bars outside the brushing rectangle. To support this, the handling of sets in the brushing algorithm had to be modified.

Scatterplot Changes to the scatterplot view are similar to those in the histogram. The view draw data, which contains point in screen coordinates, has to support multiple points per data record, to represent multiple elements. If points are to be scaled, we can now add fractional points. A data record is then represented by multiple fractional points, if it contains multiple elements. After the points are summed up, the behaviour is the same as with other datasets. Brushing however is more complex, because all elements inside the rectangle have to be looked up in the set item table. All records containing at least one of these elements have to be brushed.

Parallel Coordinates The parallel coordinates view needs to store a lot of information on elements and records in its view draw data. Instead of saving one intersection point per axis for each data record, we have to store a list of such points in case of a set dimension. Instead of using polylines crossing all axes, we have to use single line segments, because lines can split up and join again.

Chapter 7

Conclusions

When summarizing the work on this topic, certain thoughts come to my mind. The datatype introduced to information visualization in this thesis can be very useful in data analysis because it helps to keep high-dimensional data manageable. This can be the foundation of many further new visualization techniques. Especially the vast amount of methods for the visualization of categorical data gives a hint on what could be done with sets. It would be very interesting to see which visualization techniques can be adapted to sets and how useful the resulting technique still is.
On the other hand, data analysis with sets does not necessarily become easier. While visualizing a single set-typed dimension is fairly easy, multidimensional views can be very challenging. They are not necessarily difficult to implement, but require the user to concentrate even more on what he or she is seeing to avoid wrong conclusions. Some mistakes in the visualization, especially in brushing algorithms, took a long time to be detected, because we first had to find out the correct behaviour. Some complex visualizations are hard to interpret, and require more time to understand. Interaction is an important key when it comes to supporting the user in data analysis.
Another problem is, that current datasets do not contain set-typed data. Many datasets currently available were collected to be visualized with categories, but only some of them can be converted. Groups of binary dimensions have often been avoided, because they were difficult to handle. If sets become more popular, this will probably change. As a consequence, we should not drop information from datasets, only because it cannot be visualized. If we can, e.g., store hierarchical information for a set-typed dimension, then we should do it, even if we cannot visualize it right now. Additional information like that may become valuable for future techniques.

Chapter 8

Acknowledgements

This work has been done in the scope of the basic research on visualization at the VRVis Research Center in Vienna, Austria, which is funded by an Austrian research program called Kplus.
I would like to thank Helwig Hauser and Krešimir Matković for their patience and their great support during my work.

Bibliography

[1] Comvis (http://www.comvis.at/).

[2] Gapminder (http://www.gapminder.org/).

[3] Many Eyes (http://services.alphaworks.ibm.com/manyeyes/home).

[4] Opengl (http://www.opengl.org/).

[5] Spotfire (http://spotfire.tibco.com/).

[6] TIBCO Software Inc (http://www.tibco.com/).

[7] Christopher Ahlberg. Spotfire: an information exploration environment. *SIGMOD Rec.*, 25(4):25–29, 1996.

[8] Christopher Ahlberg and Ben Shneiderman. The alphaslider: a compact and rapid selector. In *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 365–371, New York, NY, USA, 1994. ACM. ISBN 0-89791-650-6.

[9] Gennadi Andrienko and Natalia Andrienko. Parallel coordinates for exploring properties of subsets. In *Coordinated and Multiple Views in Exploratory Visualization, 2004. Proceedings. Second International Conference on*, pages 93–104, July 2004.

[10] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. Toolglass and magic lenses: the see-through interface. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 73–80, New York, NY, USA, 1993. ACM Press. ISBN 0-89791-601-8.

[11] Thorsten Buering, Jens Gerken, and Harald Reiterer. User interaction with scatterplots on small screens - a comparative evaluation of geometric-semantic zoom and fisheye distortion. *IEEE Transactions on Visualization and Computer Graphics*, 12 (5):829–836, 2006.

[12] Andreas Buja, John Alan McDonald, John Michalak, and Werner Stuetzle. Interactive data visualization using focusing and linking. In *VIS '91: Proceedings of the 2nd conference on Visualization '91*, pages 156–163, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press. ISBN 0-8186-2245-8.

[13] John M. Chambers, William S. Cleveland, Beat Kleiner, and Paul A. Tukey. Graphical methods for data analysis. pages 145–156, 1983.

[14] Jean-Daniel Fekete. The infovis toolkit. In *INFOVIS '04: Proceedings of the IEEE Symposium on Information Visualization*, pages 167–174, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7803-8779-3.

[15] Thomas R. Fleming and David P. Harrington. Primary biliary cirrhosis dataset (http://lib.stat.cmu.edu/datasets/pbc), 1994.

[16] George W. Furnas. Generalized fisheye views. In *CHI '86: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 16–23, New York, NY, USA, 1986. ACM Press. ISBN 0-89791-180-6.

[17] Jeffrey Heer and Maneesh Agrawala. Software design patterns for information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):853–860, September/October 2006.

[18] Heike Hofmann. Exploring categorical data: Interactive mosaic plots. *Metrika*, 51 (1):11–26, 2000.

[19] Chung-Chian Hsu. Generalizing self-organizing map for categorical data. *IEEE Transactions on Neural Networks*, 17(2):294–304, March 2006.

[20] Alfred Inselberg. The plane with parallel coordinates. *The Visual Computer*, 1(4): 69–91, 1985.

[21] Alfred Inselberg and Bernard Dimsdale. Parallel coordinates: a tool for visualizing multi-dimensional geometry. In *Visualization, 1990. Visualization '90., Proceedings of the First IEEE Conference on*, pages 361–378, San Francisco, CA, October 1990.

[22] Jarke J. van Wijk. Sequoiaview (http://www.win.tue.nl/sequoiaview), 1999.

[23] Jimmy Johansson, Patric Ljung, Mikael Jern, and Matthew Cooper. Revealing structure within clustered parallel coordinates displays. In *INFOVIS '05: Proceedings of the IEEE Symposium on Information Visualization*, pages 125–132, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7803-9464-x.

[24] Brian Johnson and Ben Shneiderman. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In *Visualization, 1991. Visualization '91, Proceedings., IEEE Conference on*, pages 284–291, San Diego, CA, October 1991.

[25] Daniel A. Keim. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):1–8, January/March 2002.

[26] Daniel A. Keim, Helmut Barro, Christian Panse, Jorn Schneidewind, and Mike Sips. Exploring and visualizing the history of infovis. In *INFOVIS '04: Proceedings of the IEEE Symposium on Information Visualization*, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7803-8779-3.

[27] Daniel A. Keim, Ming C. Hao, and Umeshwar Dayal. Hierarchical pixel bar charts. 8(3):255–269, July/September 2002.

[28] Daniel A. Keim, Ming C. Hao, Julian Ladisch, Meichun Hsu, and Umeshwar Dayal. Pixel bar charts: a new technique for visualizing large multi-attribute data sets without aggregation. *INFOVIS '01. IEEE Symposium on Information Visualization*, pages 113–120, 2001.

[29] Zoltán Konyha, Krešimir Matković, Denis Gracanin, Mario Jelović, and Helwig Hauser. Interactive visual analysis of families of function graphs. *Transactions on Visualization and Computer Graphics*, 12(6):1373–1385, 2006.

[30] Robert Kosara, Fabian Bendix, and Helwig Hauser. Parallel sets: interactive exploration and visual analysis of categorical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):558–568, July/August 2006.

[31] Robert Kosara, Silvia Miksch, and Helwig Hauser. Semantic depth of field. In *INFOVIS '01: Proceedings of the IEEE Symposium on Information Visualization 2001*, pages 97–104, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1342-5.

[32] Matthias Kreuseler, Norma Lopez, and Heidrun Schumann. A scalable framework for information visualization. In *INFOVIS 2000: IEEE Symposium on Information Visualization 2000*, pages 27–36, Salt Lake City, UT, October 2000.

[33] John Lamping, Ramana Rao, and Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 401–408, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-84705-1.

[34] Jeffrey LeBlanc, Matthew O. Ward, and Norman Wittels. Exploring n-dimensional databases. In *Visualization, 1990. Visualization '90., Proceedings of the First IEEE Conference on*, pages 230–237, San Francisco, CA, USA, October 1990.

[35] Marc Levoy. Display of surfaces from volume data. *Computer Graphics and Applications, IEEE*, 8(3):29–37, May 1988.

[36] Bruce H. McCormick, Thomas A. DeFanti, and Maxine D. Brown. Visualization in scientific computing. *IEEE Computer Graphics and Applications*, 7(10):69, October 1987.

[37] Hauser Novotný, Matej; Hauser. Outlier-preserving focus+context visualization in parallel coordinates. *Transactions on Visualization and Computer Graphics*, 12(5): 893–900, Sept.-Oct. 2006.

[38] Harald Piringer, Robert Kosara, and Helwig Hauser. Interactive focus+context visualization with linked 2d/3d scatterplots. In *Coordinated and Multiple Views in Exploratory Visualization, 2004. Proceedings. Second International Conference on*, pages 49–60, July 2004.

[39] Ramana Rao and Stuart K. Card. The table lens: merging graphical and symbolic representations in an interactive focus + context visualization for tabular information. In *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 318–322, New York, NY, USA, 1994. ACM Press. ISBN 0-89791-650-6.

[40] Kenneth Revett. A rough sets based classifier for primary biliary cirrhosis. In *Computer as a Tool, 2005. EUROCON 2005. The International Conference on*, volume 2, pages 1128–1131, Belgrade, Serbia and Montenegro, 2005.

[41] Kenneth Revett, Florin Gorunescu, Marina Gorunescu, and Marius Ene. Mining a primary biliary cirrhosis dataset using rough sets and a probabilistic neural network. In *Intelligent Systems, 2006 3rd International IEEE Conference on*, pages 284–289, London, September 2006.

[42] Theresa-Marie Rhyne, Melanie Tory, Tamara Munzner, Matt Ward, Chris Johnson, and David H. Laidlaw. Information and scientific visualization: Separate but equal or happy together at last. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 115, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-2030-8.

[43] Jinwook Seo and Ben Shneiderman. A rank-by-feature framework for unsupervised multidimensional data exploration using low dimensional projections. In *INFOVIS '04: Proceedings of the IEEE Symposium on Information Visualization*, pages 65–72, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7803-8779-3.

[44] Zeqian Shen, Michael Ogawa, Soon Tee Teoh, and Kwan-Liu Ma. Biblioviz: a system for visualizing bibliography information. In *APVis '06: Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation*, pages 93–102, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc. ISBN 1-920682-41-4.

[45] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *IEEE Visual Languages*, number UMCP-CSD CS-TR-3665, pages 336–343, College Park, Maryland 20742, U.S.A., 1996.

[46] Greg Smith, Mary Czerwinski, Brian Meyers, Daniel Robbins, George Robertson, and Desney S. Tan. Facetmap: A scalable search and browse visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):797–804, September/October 2006.

[47] Martin Theus and Adalbert Wilhelm. Modelling categorical data by interactive mosaic plots and tables. In *Proceedings of the 11th Workshop on Statistical Modelling*, pages 462–465, 1996.

[48] Melanie Tory and Torsten Möller. Human factors in visualization research. *IEEE Transactions on Visualization and Computer Graphics*, 10(1):72–84, 2004.

[49] Melanie Tory and Torsten Möller. Rethinking visualization: A high-level taxonomy. In *INFOVIS '04: Proceedings of the IEEE Symposium on Information Visualization*, pages 151–158, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7803-8779-3.

[50] Jarke J. van Wijk. Views on visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):421–432, July/August 2006.

[51] Jarke J. van Wijk and Wim A. A. Nuij. Smooth and efficient zooming and panning. In *INFOVIS '03: Proceedings of the IEEE Symposium on Information Visualization 2003*, pages 15–23, October 2003.

[52] Jarke J. van Wijk and Huub van de Wetering. Cushion treemaps: visualization of hierarchical information. In *INFOVIS '99: Proceedings of the IEEE Symposium on Information Visualization 1999*, pages 73–78, San Francisco, CA, October 1999.

[53] Matthew O. Ward. Xmdvtool: integrating multiple methods for visualizing multivariate data. In *VIS '94: Proceedings of the conference on Visualization '94*, pages 326–333, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press. ISBN 0-7803-2521-4.

[54] Martin Wattenberg and Jesse Kriss. Designing for social data analysis. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):549–557, 2006.

[55] Wattenberg, Martin. Baby Name Wizard (http://www.babynamewizard.com/namevoyager/lnv0105.html), 2005.