Extending Chartist

Group 2

Georg Lausegger, Maximilian Weiss Reinthaler, Roman Purgstaller, and Lukas Breitwieser

Institute for Information Systems and Computer Media (IICM),
Graz University of Technology
A-8010 Graz, Austria

Nov 2014

https://github.com/breitwieser/chartist-js.git

Abstract

This report documents our enhancements on the Chartist library for simple responsive charts. It will give a short overview on what we did and our motivation.

The first part will explain our implementation for the 'axis ticks labels rotation' extension. This feature allows the front end developer to set an option which rotates label by 45 degrees. The second part of the report reviews the implementation process of the 'vertical/horizontal' feature, which gives the front end developer the possibility to create a vertical bar chart which is able to transform into a horizontal bar chart on smaller screens. This feature could be used to display wide range bar charts on both, landscape and portrait mode of a smart phone and of course desktop PC's, tablets.
The last section covers parallel coordinates, which were not part of the Chartist library before and had to be implemented by us as new chart type. The first part of the section explains what parallel coordinates are and what the are used for. The second part of the section gives an overview of the implementation process and its challenges.
This report furthermore includes a front end developer guide which gives the developer an idea on how to use the mentioned features.

Contents

Chapter 1

Introduction

Today, more and more people around the world are accessing the internet by their smart phones and tablets. Mobile devices play a major role for nearly every popular web application. Therefore, Responsive Web Design is one of the most important concepts when creating a website. The wide range of different devices makes it essential to not only provide one resolution for all different kinds of screen sizes. The main idea behind this approach is to provide tools and paradigms for creating screen size independent web applications. An important aspect of Responsive Web Design is making data visualization including charts and graphics responsive.

The chartist library provides a lot of nice features to create simple responsive bar charts, which is quite important for the stated reasons. But there are still pieces missing, we thought might be useful to developers. So our goal for this assignment was to make chartist a little better. We thought about enhancements to the library which would be interesting to implement.

The first thing that came to our minds was that is is quite complicated to deal with labels once the size of a chart decreases. Especially when labels hold long strings it is difficult to find a compromise between size(equals readability) and available space. So we implemented an additional option that gives the developer the possibility to rotate labels by 45 degrees, which saves a lot of space and increase readability.
Another idea of ours was that the responsiveness of vertical bar charts is somehow limited in Chartist, because of the aspect ratio feature. This feature allows the developer to define a aspect ratio to which the chart sticks, no matter on what screen size it is displayed. It was our intention vertical bar charts, as they are provided by Chartist now, are quite good for desktop screens and smart phones screens in landscape mode, but are getting too small to gather any information out of them, once they are viewed on a mobile device in portrait mode. With our new 'vertical/horizontal' feature the front end developer is able to create bar charts which are both horizontal and vertical. By setting the new rotate option the developer can define a screen width on which he wants the bars to change their alignment. Once the screen width is lower than the defined value, bars are displayed horizontal and change to vertical again as soon as the screen width is higher than the value.
A common way to represent complex datasets with more than two parameters, are parallel coordinates. This chart type is provided by several established charting libraries and is used to display n-dimensional data on n vertical, parallel lines. Since Chartist don't features parallel coordinates yet, we decided to face the challenge and implement it by our own.
The following chapters will give an overview on the implementation process of all the above mentioned features and its challenges. An added front end developers guide will give the reader instructions on how to use them.

Chapter 2

Rotate Labels

2.1 Motivation

Chartist offers the possibility to create all basic chart types including line charts, bar charts and pie charts. When considering axis ticks labels, the question of how to improve these in terms of an responsive behaviour can be considered. By comparing Chartist with another big library used for the creation of responsive charts, Highcharts, it turns out that the fronted-developer could specify an angle for the axis ticks labels to be rotated (e.g. 45 degrees) to save space and therefore improve responsiveness. Especially on smaller screen sizes like smartphones, the rotation of the axis ticks labels make sense.

2.2 Implementation

At first, the decision that the axis ticks labels on the X and the Y axis can be rotated separatly was chosen. Thus, it should be possible to rotate for example the labels on the X axis, while the labels on the y axis stay fixed. So the first thing to do was to specify an option to enable the axis ticks labels to be rotated. For this purpose, the **rotateLabel** option have been introduced:

```
1 var options = {
2   axisX: {
3     rotateLabels: true
4   },
5   axisY: {
6     rotateLabels: true
7   }
8 };
```

Figure 2.1: rotateLabel Option in Chartist

Code listing 2.1 shows a simple example of how the option structure in Chartist look like. In this case, the axis ticks labels on the X as well as the Y axis would be rotated, cause the **rotateLabel** option is set to true on both axis. In case the **rotateLabel** option is not specified by the frontend developer, Chartist holds a data structure with default options. So the **rotateLabel** option is set to **false** by default.

Since the **rotateLabel** feature should be available not only for bar charts, but also line charts, it make sense to make the code reusable as good as possible. The code changes are therefore done in the core.js file, which is used by both chart types to draw the basic Chartist components like the Chartist rect (place where the chart is drawn) as well as the axis themselves and according to them, the axis ticks labels. So the implementation of

the rotateLabel feature take place in the functions *createXAxis* and the *createYAxis*. This is the place where the axis are drawn and the axis ticks labels are generated.

Chartist uses foreign objects with embedded HTML span tags to display labels. This guarantees a better text handling possibility than with standard SVG text element. However, there is also a fallback for browsers that doesn't support the foreign object element such as the Internet Explorer. In this case, a standard SVG text element is used for display the labels. Code listing 2.2 shows how a label is generated in Chartist.

```
1 <foreignObject x="45" y="269" width="86.6" height="30" style="overflow:
      visible;">
2 <span class="ct-label ct-horizontal" xmlns="http://www.w3.org/1999/xhtml">
     Monday</span>
3 </foreignObject>
```

Figure 2.2: Foreign Object Element for Displaying a Label

Our first intention was therefore to manipulate the **transform** attribute of the foreign object directly to make the labels rotate. This results in problems with multiline labels (long labels that need to be separated on more than one line) and very long labels, because the bounding box of the labels is calculated from the width of one gridline to the next one and could therefore be very large.

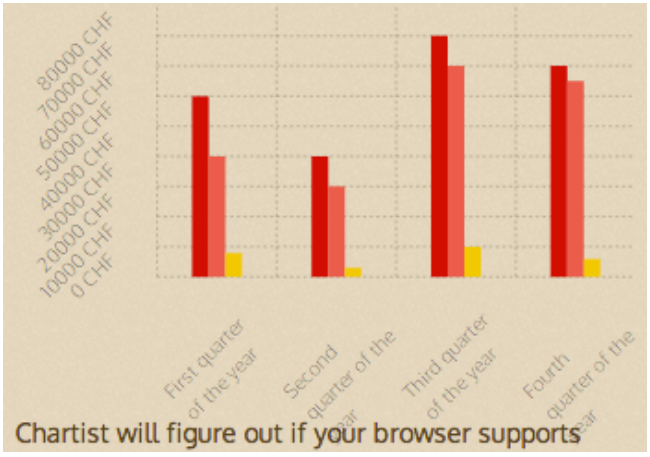


Figure 2.3: Problem with Multiline Labels

Figure 2.3 shows a common problem on multiline labels with the direct rotation.

In order to avoid those problems, we came up to the decision to reimplement this feature and use CSS3 transformations instead. This has several advantages, beside the small amount of code need, CSS3 transformations ensure:

- GPU acceleration - The browser can take advantage of delegating computations to the GPU rather than the CPU (although the rotation of the axis ticks labels doesn't need heavy computations)
- It fullfills the concept of **separation of concerns**

Code listing 2.4 shows how the rotation of the labels is done via CSS3 transformation.

Putting it together, depending on how the axis ticks labels should be rotated or not, a CSS class containing the necessary code to rotate the labels is appended to the labels. Again, there are two different CSS classes,

```
1              transform-origin: 0% 50%;<br>
2              -ms-transform: rotate(-45deg); // IE 9 <br>
3              -webkit-transform: rotate(-45deg); //Chrome, Safari, Opera<br>
4              transform: translateY($ct-translate-label-y) rotate(-45deg);
```

Figure 2.4: CSS3 Transformation (Rotate Labels by 45 Degrees)

ct-rotatedlabels-xaxis and *ct-rotatedlabels-yaxis* that are appended. This has the advantage that the code can be modified for the X and the Y axis, without making the same change to the axis ticks labels that lie on the other axis. So all that's needed to rotate the labels is to add the given class to the label if the option **rotateLabels** is set to true:

```
1          // default case (ct-label ct-label-horizontal)
2          var classnames = [options.classNames.label, options.classNames.
               horizontal].join(' ');
3
4          if(options.axisX.rotateLabels) {
5            // add rotation for labels by adding css class
6            classnames = [options.classNames.label, options.classNames.
                 horizontal,
7                          options.classNames.labelRotationXAxis].join(' ');
8
9          }
```

Figure 2.5: Add Class for Rotation (Here for the X Axis)

What's left is the problem with very long and multiline labels - e.g. unwanted cuttings of labels. For that purpose, some kind of a hack is used to determine whenever the label should be trimmed. To determine if the label should be shortened, a hidden inline (no line break) span tag is appended to the HTML body. The label content is then copied to this new hidden inline span tag. Then the width of the original span is compared with the new span element. If the new span element's width is greater than the original one, there's a line break in it and we cut of the label with the CSS property *text-overflow: ellipsis*. Finally, the hidden span tag can be removed from the DOM. Every label that's rotated are checked for line breaks. The function *trimAxisLabels* which is shown in code listing 2.6 does exactly the above mentioned hack.

Figure 2.7 shows how the rotation feature looks like in action.

```
1  Chartist.trimAxisLabels = function (label_classname) {
2
3    $(label_classname).each(function(i) {
4      // clone element to non visible inline span
5      var element = $(this)
6          .clone()
7          .css({display: 'inline', width: 'auto', visibility: 'hidden'})
8          .appendTo('body');
9
10     // do not wrap the label and cut it if it's overflowing
11     if( element.width() > $(this).width() ) {
12       $(this).css({'display':'block','white-space':'nowrap','overflow':'
             hidden','text-overflow':'ellipsis'});
13     }
14
15     element.remove();
16   });
17 } ;
```

Figure 2.6: Trim Multiline Labels if Rotated

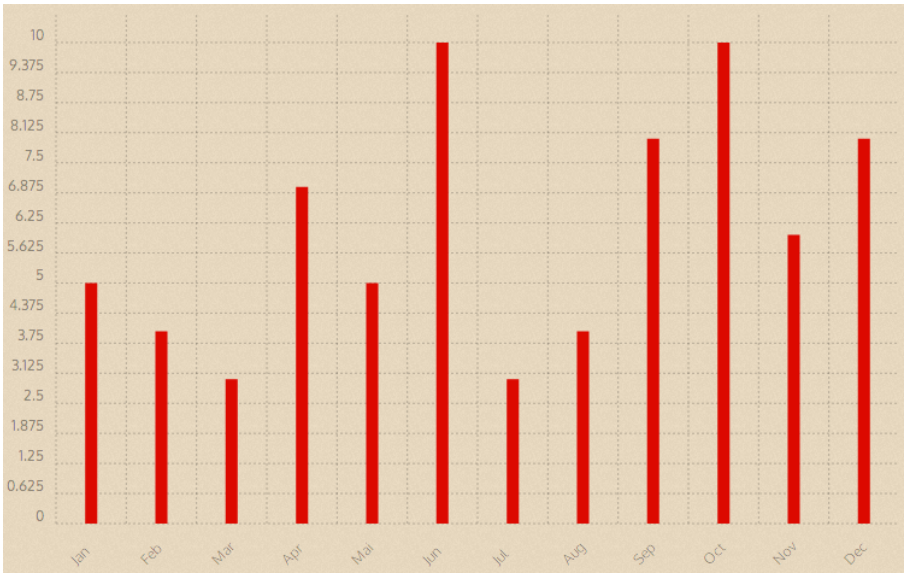


Figure 2.7: Simple Bar Chart with Rotated Labels on the X Axis

Chapter 3

Bar Chart

3.1 What is a bar chart?

Alternatively referred to as a bar graph, a bar chart is a graphic representation of data. Bar charts show horizontal or vertical bars going across the chart horizontally, with the values axis often displayed on the bottom of the chart. [**Computerhope**]

Bar Charts are useful for comparing classes or groups of data. In bar charts, a class or group can have a single category of data, or they can be broken down further into multiple categories for greater depth of analysis. [**Skymark**]

Since this paper discusses responsive charts, there is one characteristic of bar charts that is very important for the following discourse and that would be that every bars length is proportional to its value. So changing the proportions of the bars or the scale is not allowed in any circumstances.

3.2 Responsive bar chart

Making a bar chart responsive is quite a challenge. Since one of the main requirements of a bar chart is to display the relations between datasets. If someone wants to show that dataset A is three times bigger then dataset B, the bar chart has to reflect that fact even on a small screen. So the most important aspect on resizing a bar chart is to keep all information that the bars provide. What other information is kept or thrown away differs from use case to use case and depends very strongly on what data is visualised.

Scaling

A good approach to this issue is to treat the whole graph like other media blocks on web pages, which are dependent on their dimensions, like images or videos. The **chartist** framework provides a quite simple solution for this use case:

When a designer talks to a developer about the images being 320x240 on this page and 300x200 on that element, he actually just translated his idea of using 4:3 and 3:2 images into pixels. With Chartist you can specify those ratios directly on containers without the need to calculate any fixed dimensions.[**ChartistGettingStarted**]

As shown in figure 5.1, the whole chart is getting smaller and smaller with decreasing screen size, but that's one of the compromises that have to be accepted. Exact data and values are much more difficult to read, probably not without zooming into the chart. Anyway, this still could be a good way to display that chart if it fits the requirements of the information provided and the needs of (most) of the users. For example the use case

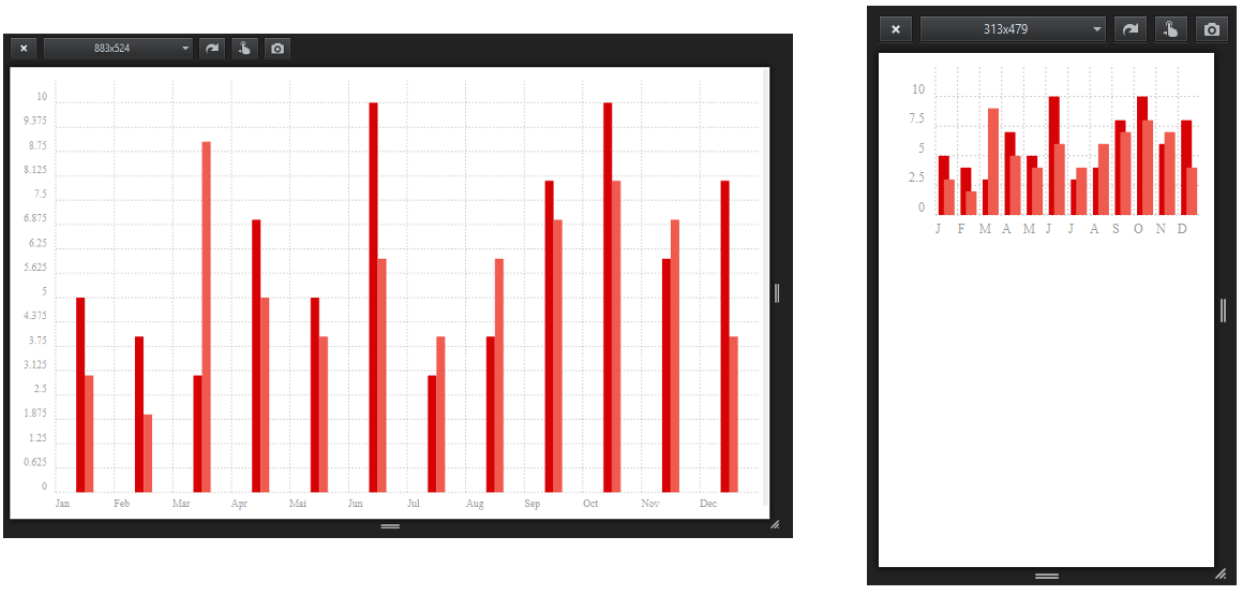


Figure 3.1: Example of a responsive bar chart created with Chartist.js. The X-axis is labelled with values. The Y-axis names the data sets. With decreasing screen size the bars start to overlap, so they won't use too much space. The code is found here.

of a web page which wants to inform its users on the results of an election. Even with the smallest screen size, one quick look on the bar chart tells the user who won the elections and how the votes are distributed. For the exact percentage of a candidate he might have to zoom into the chart.

Modify X Axis

Another way to make a **horizontal** bar chart responsive is to resize just the X-axis. Although a bar chart is a two dimensional medium, the information that is provided is one dimensional. So it is allowed to keep the information on the Y-axis the way it is want and the way it makes the chart readable. This whole approach could be used the other way around if the chart is a vertical one. If the designer is able to choose between a horizontal and a vertical bar chart, it would be wise to choose horizontal one to generate a chart that is readable on smart phones, which have become the number one device to access the web these days. Only if someone can be sure that the grand of the users are accessing the information from an desktop PC, a vertical bar chart would be the better choice. In figure 5.2 a good example of a responsive horizontal bar chart is shown.

The bar chart is expressive and readable to the user on a wide screen and is not loosing any important information while the screen size is decreasing. At first only the length of the bars is getting smaller, but when decreasing the screen size reaches a screen width below 460px the designer decided to delete less necessary information from his chart. This scaling can be done without a special framework. The designer of this page created his own stylesheet, which can be combined with the JavaScript event **onresize**. If someone is controlling the responsiveness of bar charts by his own stylesheets and JavaScript functions, it is clearly more work than using a simple library like Google Charts, but also gives him the opportunity to decide, what is important information and what can be left out if space is getting rare, by his own. So, for a very specific bar chart with essential and complex information, a customized bar chart containing an existing library and self-created stylesheets, would be a wise choice.

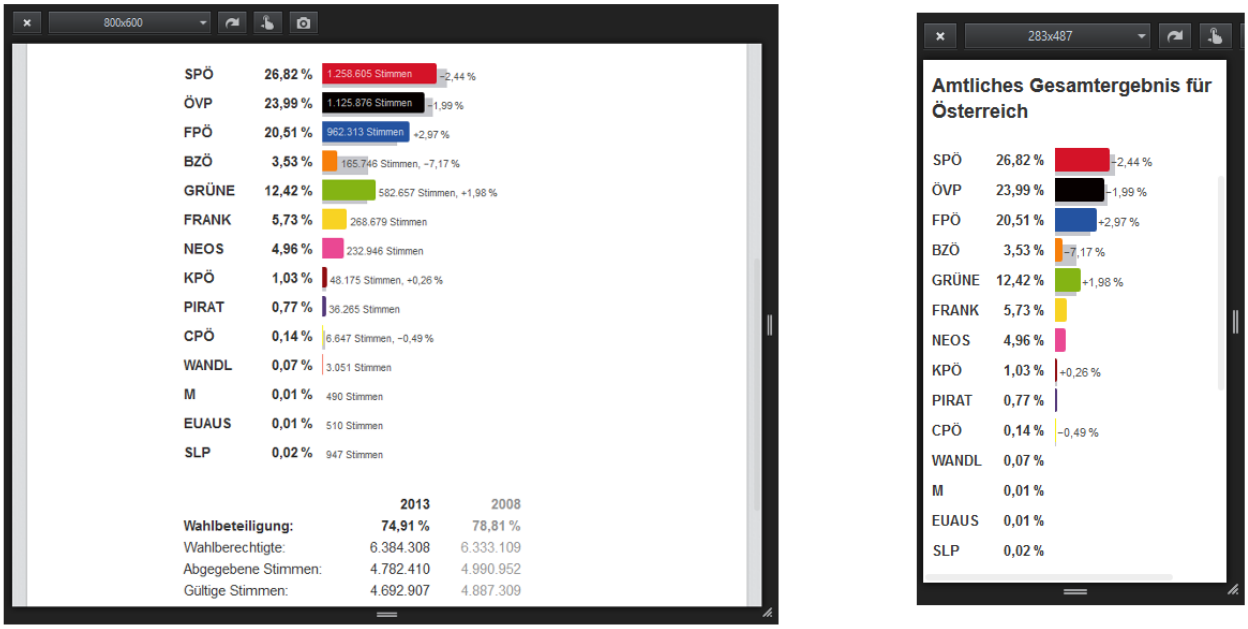


Figure 3.2: Horizontal bar chart from orf.at showing the results from the national elections.

3.3 Rotate Bar Chart Implementation

The Motivation for this feature was the fact that more and more people are accessing the internet with their smartphones or tablets. Those two devices, both have a portrait and a landscape mode and are very likely to get turned around by their users. A vertical bar chart may not by the ideal chart for portrait mode. One may very soon reach the point were a chart is to small to be expressive or even readable to it's viewer, because it has to fit to the width of smartphone screen. Even if responsive it's ability to shrink and keep its dimensions is limited. Consider the following scenario. A bar chart has lots of different data sets but their values are not vary so much. The result may be a very wide an flat bar chart. This bar chart would be much more readable on a smartphone in portrait if it was displayed wit horizontal bars. So our idea was to add a feature to chartist to create bar charts which are able to switch from vertical to horizontal when screen width is getting rare. To do so we had to introduce a new member to Chartists responsive options section simply called 'rotate'. With this option the front end developer can define on which screen width he wants the bar chart to be a horizontal one.

```
1    var responsiveOptions = [
2        ['screen and (max-width: 28em)', {
3            rotate: true
4        }],
5        ['screen and (min-width: 28em)', {
6            rotate: false
7        }]
8    ];
```

As shown in code snippet above the developer can set the 'rotate' responsive option true or false for a certain screen width.

To implement this feature we had to go through the whole process of the chart creation, because Chartist doesn't provide horizontal bar charts. Even if discussed in the github issue section, it's not implemented yet. Also it wasn't our intention to create a new type of chart, for Chartist called 'Horizontal Bar Charts', but create a kind of an hybrid version of the already existing bar chart version. Besides the naming problem (a bar chart

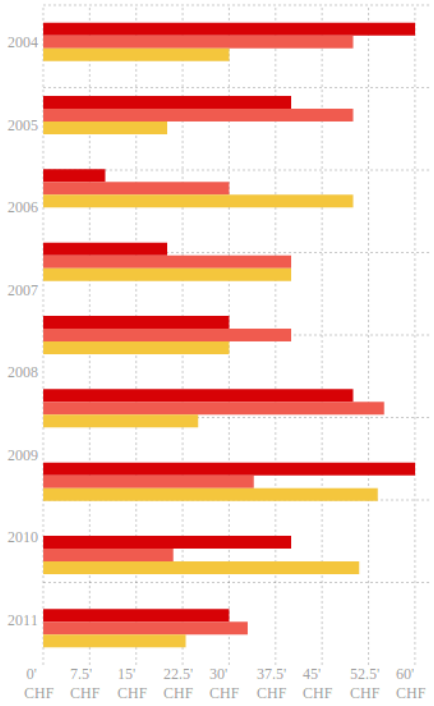


Figure 3.3: Horizontal bar chart created with Chartist.

exactly has horizontal bars, a chart with vertical bars is called column chart) the motivation here was that the front end developer doesn't need to choose between horizontal bars and vertical bars, but can create simple bar chart as usual and only if he decides in the end that this option would be useful he is able to add it with a few lines of code in the responsive options section to his existing chart.
The first problem we had while implementing this feature were caused by the labels. First of all we had to interchange the labels of the data sets with the bounds, because a horizontal bar chart needs the bounds displayed on the bottom of the chart and the labels for the data sets on the left side where the bars start. For the vertical bar chart provided by Chartist, labels are displayed beneath the related bar, starting right if the grid line that bounds the bars on the left. To achieve the same result for labels on Y-axis, like they are needed for the horizontal bar chart, we had to place them exactly between the two limiting grid lines of a data set. There for we divided the total height of the chart and divided it through the number of data sets. With this value and the index of the data set we calculated the exact y-axis position of the label. On the x-axis position the labels now behave similar to what bounds behave on vertical bar charts.

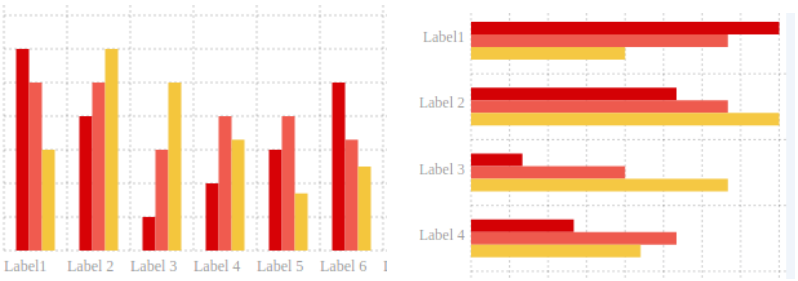


Figure 3.4: Labels for vertical and horizontal data sets.

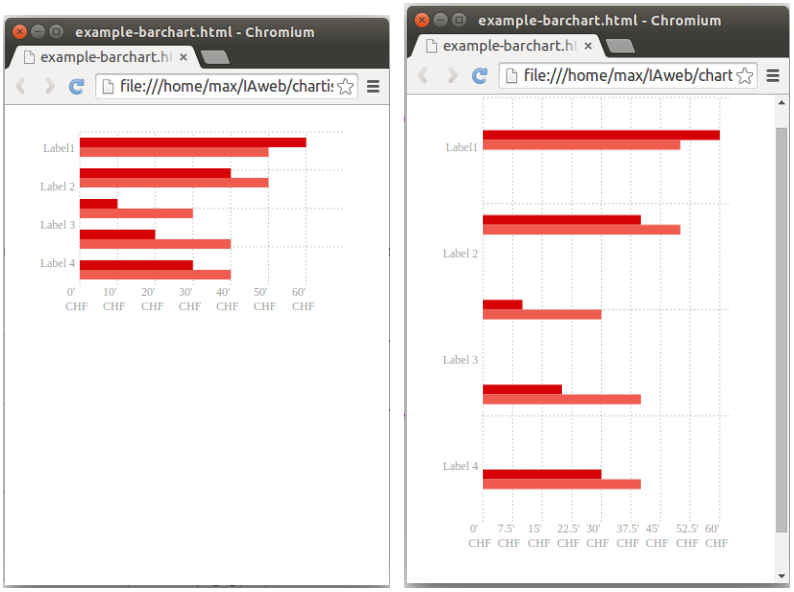
For the bounds we discovered other problems. Since they were displayed one below the other on for the vertical bars, Chartist calculates to much bounds for the horizontal bars, where they are displayed next to each other and space is rare of course. So we had to make sure that Chartist algorithm calculates fewer bounds for the horizontal bar charts, then for the vertical. **Figure 2.6**

(a) Originally calculated bounds

(b) Reduced bounds

Figure 3.5: If displayed on the x-axis, fewer bounds are needed, since they get displayed next to each other.

Another issue comes width the defined aspect ratio of a chart. Chartist let's the front end developer define an aspect ration for the created chart. So the chart sticks on that aspect ratio no matter on what screen or device it is displayed. Creating a horizontal bar chart which sticks to an aspect ratio of e.g. 3:4 or 4:5 on portrait mode doesn't save any space, because the chart just uses half of the screen. For this reason we defined new values for aspet ratios in the 'chartist-settings.scss' file. For every possible apect ration we defined an inverse complement. So when ever the a bar chart is created and 'rotate' option is true, we now look up the aspect ratio class of the chart and use its complement to create a horizontal version of it.



(a) In this example the horizontal bar chart uses the same aspect ratio as the verticals one.

(b) In this example an inverse aspect ratio is used, so that the chart can use the whole screen.

Figure 3.6: The same chart with different aspect ratios.

```
// Scales for responsive SVG containers
(1),      - ct-square,        /* no inverse here */
(15/16), - ct-minor-second,     - (16/15),      - ct-minor-second-inverse,
(8/9),    - ct-major-second,    - (9/8),     - ct-major-second-inverse,
(5/6),    - ct-minor-third,     - (6/5),     - ct-minor-third-inverse,
(4/5),    - ct-major-third,     - (5/4),     - ct-major-third-inverse,
(3/4),    - ct-perfect-fourth, - (4/3),     - ct-perfect-fourth-inverse,
(2/3),    - ct-perfect-fifth,  - (3/2),     - ct-perfect-fifth-inverse,
(5/8),    - ct-minor-sixth,     - (8/5),     - ct-minor-sixth-inverse,
(1/1.618), - ct-golden-section, - (1.618/1) ,   - ct-golden-section-inverse,
(3/5),    - ct-major-sixth,     - (5/3),     - ct-major-sixth-inverse,
(9/16),  - ct-minor-seventh,  - (16/9),      - ct-minor-seventh-inverse,
(8/15),  - ct-major-seventh,  - (15/8),      - ct-major-seventh-inverse,
(1/2),    - ct-octave,        - (2/1),     - ct-octave-inverse,
(2/5),    - ct-major-tenth,     - (5/2),     - ct-major-tenth-inverse,
(3/8),    - ct-major-eleventh, - (8/3),     - ct-major-eleventh-inverse,
(1/3),    - ct-major-twelfth,     - (3/1),     - ct-major-twelfth-inverse,
(1/4),    - ct-double-octave,       - (4/1))     - ct-double-octave-inverse
```

Figure 3.7: All aspect ratios provided by Chartist and their complements.

Chapter 4

Parallel Coordinates

4.1 What are Parallel Coordinates

Parallel coordinates is a technique for analysing data. Given a set of points in n-dimensional space and n parallel vertical lines. A point is represented as a polyline with vertices on the vertical axis. A much more intuitive description would be to talk about the representation of a data table. Each column is mapped onto a vertical axis where each data value would end up somewhere along the line, between the minimum at the bottom and the maximum at the top. The points belonging to the same record (row) are connected with lines. That creates the characteristic jumble of lines parallel coordinates are famous for. **Kosara2010**

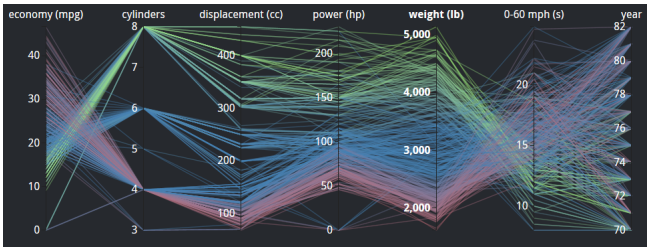
An advantage of parallel coordinates is that they reveal correlations between multiple variables. Therefore it is often helpful to separate specific records of similar data from the dataset. **Few2006**

Figure 4.1a shows an example of parallel coordinates with cars from the seventies and eighties. Draging around the axis makes it possible to filter the datasets and therefore show correlations between the data. In figure 4.1b the dataset is filtered for cars having 4 cylinders

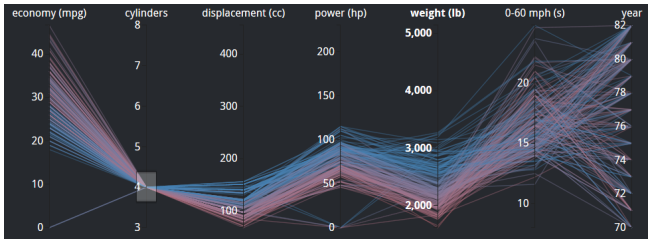
4.2 Responsive Parallel Coordinates

4.2.1 Scaling

There are various approaches that come into minds when thinking about responsive parallel coordinates. The most obvious approach would simply be scaling. Adjusting the size of the chart to the size of the element may not be the best solution. Especially large datasets would make a diagram unreadable on small screens.

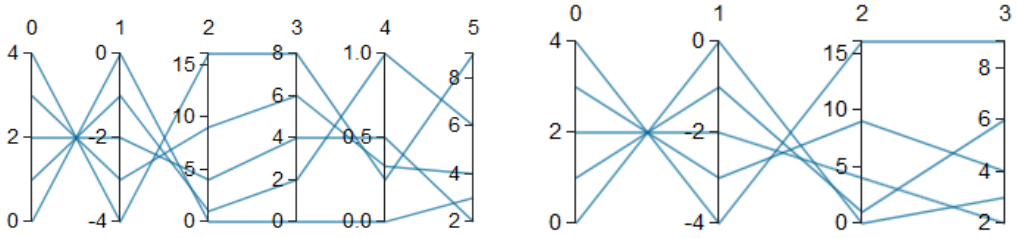


(a) Parallel Coordinates with Unfiltered Data



(b) Parallel Coordinates Showing Records from Cars with 4 Cylinders

Figure 4.1: Parallel Coordinates Using Car Data from the Seventies and Eighties Taken from **Evans**



(a) A simple example with parallel coordinates using 6 columns

(b) The Same Example but with Column 4 and 5 Removed

Figure 4.2: Removing Columns from Parallel Coordinates, the Full Example is Taken From **Chang**

4.2.2 Removing Columns

Another possibility is to simply clean up the data and remove some columns from the dataset and therefore also from the chart. The problem with this approach is that the new chart is different from the original as shown in figure 4.2

Chapter 5

Parallel Coordinates Implementation

5.1 Parallel Coordinates Plot

Code listing 5.2 shows how to draw a parallel coordinate plot inside a container with `class="ct-chart"`. In this case default options and default responsive options have been used. The resulting plot can be seen in figure 5.1.

Parallel Coordinates (PC) are usually used with many data records. If many of them have the same value on one dimension it is impossible to tell which line segments belong to one data record. To solve this issue the user can move the mouse over a line segment and the corresponding data record is highlighted (see Figure 5.5). This behaviour is achieved using the CSS `hover` selector.

This paragraph points out the similarities and differences between parallel coordinates and line charts. Figure 5.4 shows the basic structure of the former chart type. The outer rectangle represents the entire svg element with the point of origin in the upper left corner. The smaller inscribed box is the `chartRect`. Its height and width is reduced due to padding and area for the labels. This is also the first difference: In PC the labels area is above the `chartRect`, while it is below for line charts. Furthermore, PC has a slightly modified x-axis. There is no horizontal line – only grid lines are drawn. The next difference concerns the y-axis – instead of one PC has N of them (N being the number of dimensions). This means that there are no continuous grid lines for the y-axis.

As the name suggests, the method `createChart` is responsible for generating the final svg element. It uses functions from the core module that have already been implemented (e.g. `Chartist.createSvg(...)`. The first action after the svg element has been created is to create the bounds objects. The bounds object stores all necessary information for one y-axis – including: `min, max, range, step, numberOfSteps, ...`. Due to the fact that there are more than one y-axis the private function `getBoundsArray` deals with this modification. Also the position of the chart rectangle needs to be modified afterwards (call to the core function is decorated with a private function). The next task is to draw the actual data records. This is done before any axis or grid is drawn. This solved the problem of readability of the axis labels, if many data records are drawn over them. This is possible because the characteristics of the axis are already known (`bounds`). The function iterates over all data records and dimensions and draws the line segments. Therefore the y position of the line has to be calculated. Thereby it should be noted that the minimum of the dimension is always located at the bottom of the chart rectangle. The maximum on the other side has no fixed location and as a result cannot be used to transform a value into pixels. The last step is to add axis, grids and labels. Thereby, modified versions of `createXAxis/createYAxis` of the core modules are used.

5.2 Responsive Behaviour

Like other chart types that have already been implemented in this library, parallel coordinates is also able to adjust its behaviour/appearance based on the screen dimensions or used device. The front end developer can

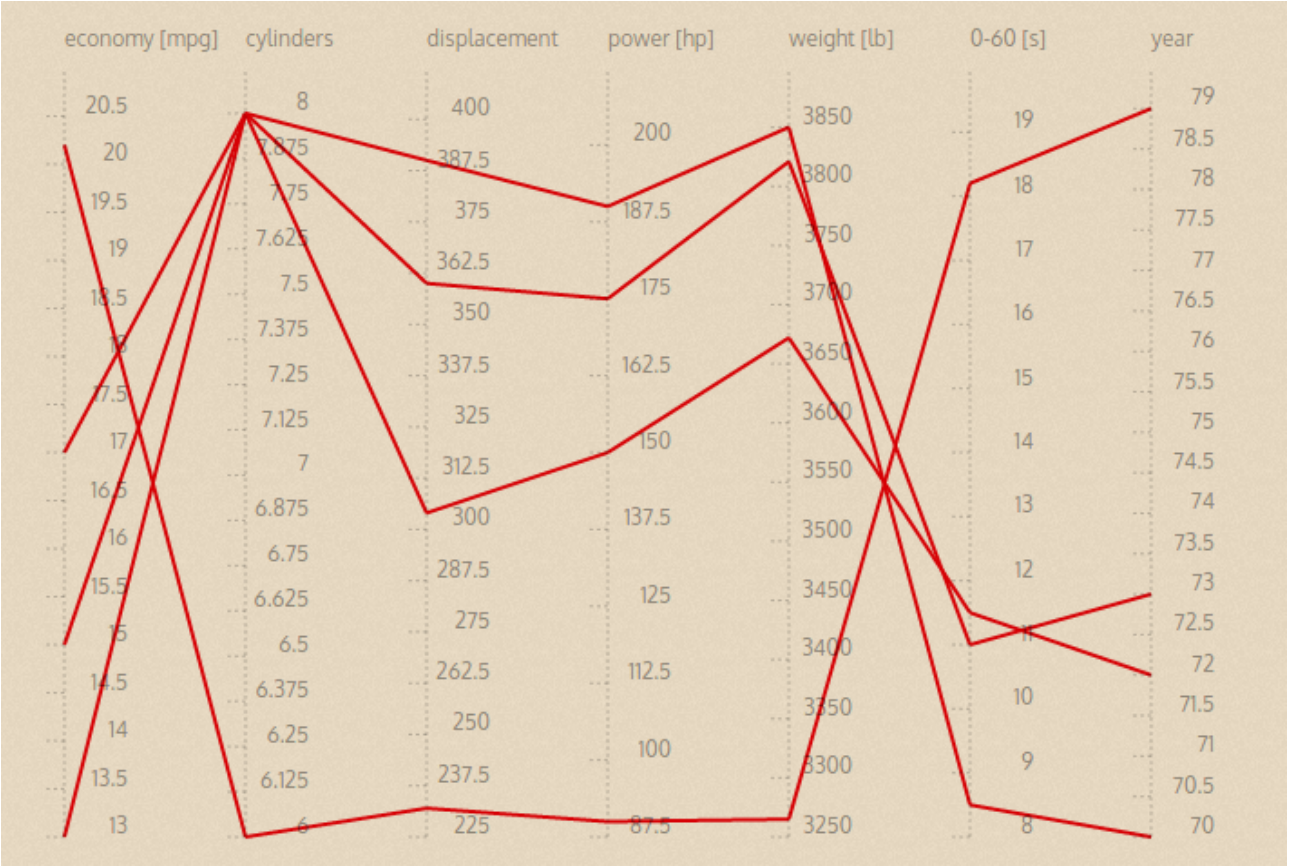


Figure 5.1: Parallel Coordinate Example

```
new Chartist.ParallelCoordinates('.ct-chart', {
    labels: ['economy [mpg]', 'cylinders', 'displacement', 'power [hp]', '
        weight [lb]', '0-60 [s]', 'year'],
    series: [
        [13,8,360,175,3821,11,73],
        [15,8,390,190,3850,8.5,70],
        [17,8,304,150,3672,11.5,72],
        [20.2,6,232,90,3265,18.2,79]
    ]
});
```

Figure 5.2: Code Snippet that Shows How To Embed this Chart Type on a Web Page

```
1   function createChart(options) {
2     var seriesGroups = [],
3       bounds,
4       normalizedData = Chartist.normalizeDataArray(Chartist.getDataArray(this.
            data), this.data.labels.length);
5
6     // Create new svg object
7     this.svg = Chartist.createSvg(this.container, options.width, options.
          height, options.classNames.chart);
8
9     // initialize bounds
10    bounds = getBoundsArray.call(this, normalizedData, options);
11
12    // create chart rect
13    var chartRect = createChartRect.call(this, options);
14
15    //draw all data records
16    drawDataRecords.call(this, seriesGroups, options, chartRect, bounds,
          normalizedData);
17
18    //draw grid, axis and labels
19    drawAxisGridsLabels.call(this, options, chartRect, bounds);
20
21  }
```

Figure 5.3: `createChart` Method – Advanced Features Removed

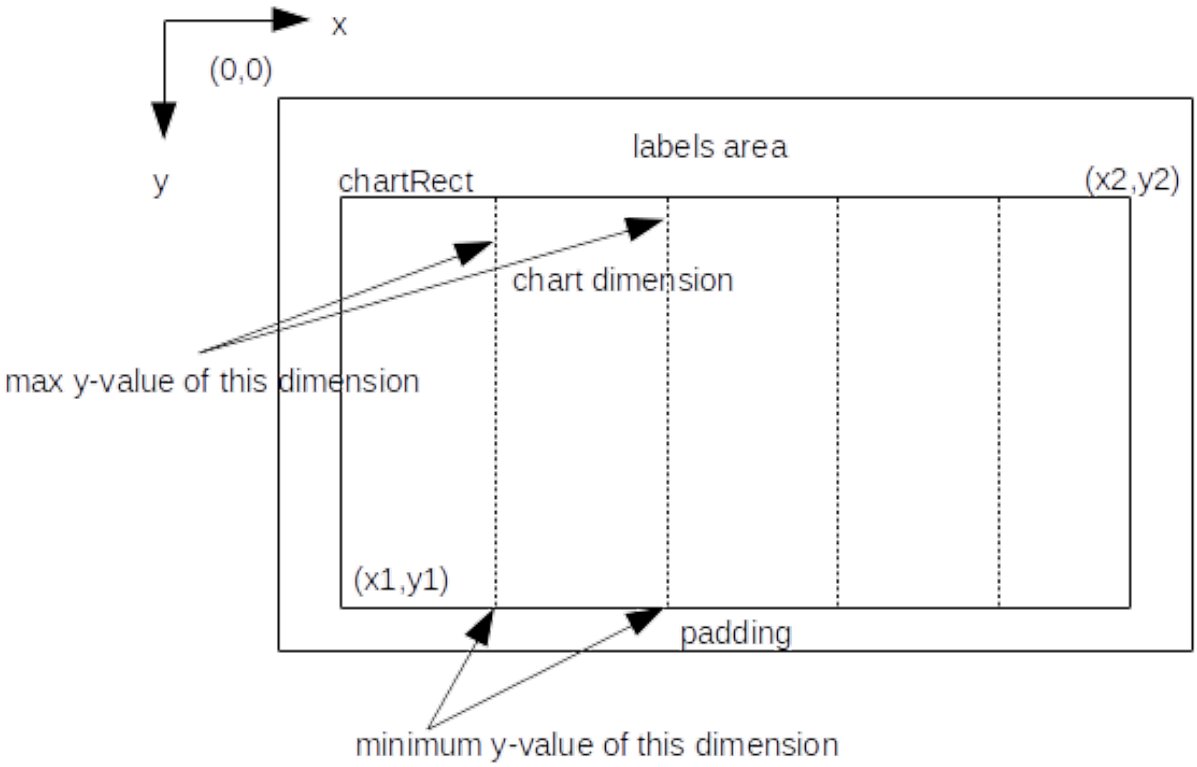


Figure 5.4: Parallel Coordinate Schema

Figure 5.5: Highlighted Data Record on Mouse Over

define options that are applied only if the corresponding media query is true. The most powerful option is the definition of the `labelInterpolationFnc` for the x-axis. This means, that the front end developer can hide dimensions on smaller screens that are less important. Code listing 5.6 shows how to hide all dimensions except "economy", "power" and "year" on screens that are less than 40em wide, while all dimensions are displayed on larger screens. The results can be viewed in figure 5.7. During the creation of the chart this function is called for all values specified in `labels`.

The current solution has two limitations:

- `labelInterpolationFnc` for the y-axis is ignored (as described earlier this chart type has more than one y-axis)
- alterations of the label value are also ignored, because it caused problems in combination with user selection of displayed dimensions (see section 5.4.1). Example: In line charts it is possible to shorten the labels by returning for example `value.substring(0,9)+"..."` inside the `labelInterpolationFnc` - The original label "Very long label on the x-axis" would be changed to "Very long...")

5.3 Styling with SASS

The CSS for chartist are created with SASS. **SASS** Therefore front end developer can customize charts by either removing the CSS fully and write their own selectors by using SASS mixins or simply customize the CSS file. **ChartistGettingStarted**

For the purpose of implementing parallel coordinates with its own style the current implementation of the SASS settings are extended. To extend the default style in chartist one has to consider 2 files:

- `/src/styles/settings/_chartist-settings.scss`
- `/src/styles/chartist.scss`

The former file contains default values and class names which are used to generate the CSS. See figure 5.8 for properties used styling parallel coordinates. Most of the above mentioned variable names are not directly referenced to parallel coordinates. The purpose of this design is that the properties can easily be used for other chart types.

The file `chartist.scss` contains all SASS mixins. The chartist extension also includes mixins for parallel coordinates as illustrated in figure 5.9. Again, the mixins are kept as generic as possible. For example a line chart can be extended with a mean using the existing style.

```
var responsiveOptions = [
      ['screen and (max-width: 40em)', {
          axisX: {
              labelInterpolationFnc: function(value, index) {
                  return ["economy [mpg]", "power [hp]", "year"].indexOf(
                     value) != -1 ? value : null;
              }
          }
      }],
      ['screen and (min-width: 40em)', {
          axisX: {
              labelInterpolationFnc: function(value, index) {
                  return value;
              }
          }
      }]
  ];
  ...
  new Chartist.ParallelCoordinates('.ct-chart', data, options,
     responsiveOptions);
```

Figure 5.6: Responsive Behaviour Definition

(a) Large Screen **(b)** Small Screen

Figure 5.7: Responsive Behaviour

```
// parallel coordinates properties
$ct-parallelCoordinates-line-width: 2px !default;
$ct-parallelCoordinates-dasharray: false !default;
$ct-point-size: 10px !default;
// Line chart point, can be either round or square
$ct-point-shape: round !default;
// Area fill transparency between 0 and 1
$ct-area-opacity: 0.1 !default;
// Mean properties
$ct-mean-stroke: #006400 !default;
$ct-mean-line-width: 4px !default;
// Histogram properties
$ct-histogram-stroke:#000000 !default;
$ct-histogram-width:2px !default;
$ct-histogram-opacity:0.7 !default;
$ct-histogram-fill:#A9A9A9 !default;
// Mono series
$ct-series-mono-color: #d70206 !default;
$ct-series-mono-line-width: 2px !default;
```

Figure 5.8: SASS Default Properties for Parallel Coordinates

```
@mixin ct-chart-parallelCoordinates($ct-parallelCoordinates-line-width: ct-
    parallelCoordinates-line-width, $ct-parallelCoordinates-dasharray: ct-
    parallelCoordinates-dasharray) {
fill: none;
stroke-width: $ct-parallelCoordinates-line-width;
@if ($ct-parallelCoordinates-dasharray) {
stroke-dasharray: $ct-parallelCoordinates-dasharray;
}
}
@mixin ct-mean($ct-mean-line-width: $ct-mean-line-width, $ct-mean-stroke: $ct-
    mean-stroke) {
fill: none;
stroke-width: $ct-mean-line-width;
stroke: $ct-mean-stroke;
}
@mixin ct-histogram($ct-histogram-stroke: $ct-histogram-stroke, $ct-histogram-
    width: $ct-histogram-width, $ct-histogram-fill: $ct-histogram-fill) {
fill: $ct-histogram-fill;
fill-opacity: 0.2;
stroke: $ct-histogram-stroke;
stroke-width: $ct-histogram-width;
stroke-opacity: 0.3;
}
```

Figure 5.9: SASS Default Properties for Parallel Coordinates

5.4 User Interaction

So far the user was only able to passively consume the rendered chart. This section deals with features that give the user the opportunity to influence the appearance.

5.4.1 Select Displayed Dimensions

Chapter 5.2 described the possibility to hide dimensions under certain circumstances. This decision is made by the front end developer and can't be changed later by the user. It might well be actually the case that the user and front end developer have a different understanding about the importance of a dimension. As a result a feature has been implemented that let the user choose which dimensions are displayed. This means she can also override the decision of the front end developer.

During the design of this functionality it was found advantageous to render the menu in a separate container outside the `svg` element due to the following reasons:

- Simplicity
- Browser compatibility
  If the menu should be drawn in the `svg` element, the best option would be to use foreign object. This means that html tags can be included inside an svg element. Unfortunately this feature is not supported on all browsers **mozillaForeignObject**
- Flexibility
  The front end developer can adjust the appearance according to his own needs using CSS and the position of the container in the DOM tree.

The option that enables this features is `selectDisplayedDimContainer` – see code snippet 5.11. The front end developer specifies the selector – in this case it is `".ct-select-dimensions"`. The example in 5.12 shows how the generated HTML code looks like. It is an unordered list with one list item per dimension. Inside the list item there is an `input` element that contains the checked attribute if the dimension is shown. Additionally, there is an event handler attached to each input box that applies the changes to the corresponding chart. As you can see in in figure 5.10 the resulting menu has no bullet points and no line break after each dimensions. This illustrates the flexibility that was mentioned earlier and is achieved through CSS code in listing 5.13

This solution integrates well with the already existing responsive behaviour. For example let us assume that there is a chart made for mobile devices with the option to hold the display in portrait or landscape mode. In the smaller portrait view the front end developer decided to hide the dimensions "economy", "power" and "year". The letters in the following enumeration and in figure 5.10 belong together.

(a) User starts in portrait mode
Cylinder, displacement, weight and acceleration have been filtered out according to the definition of the front end developer.

(b) User hides the "power" dimension and shows "0-60"
He overrides the front end developers definition, because the dimension "0–60" is more important for him.

(c) User changes the view to landscape
Note that the dimension "power" is also hidden, because the user deselected it in step 2.

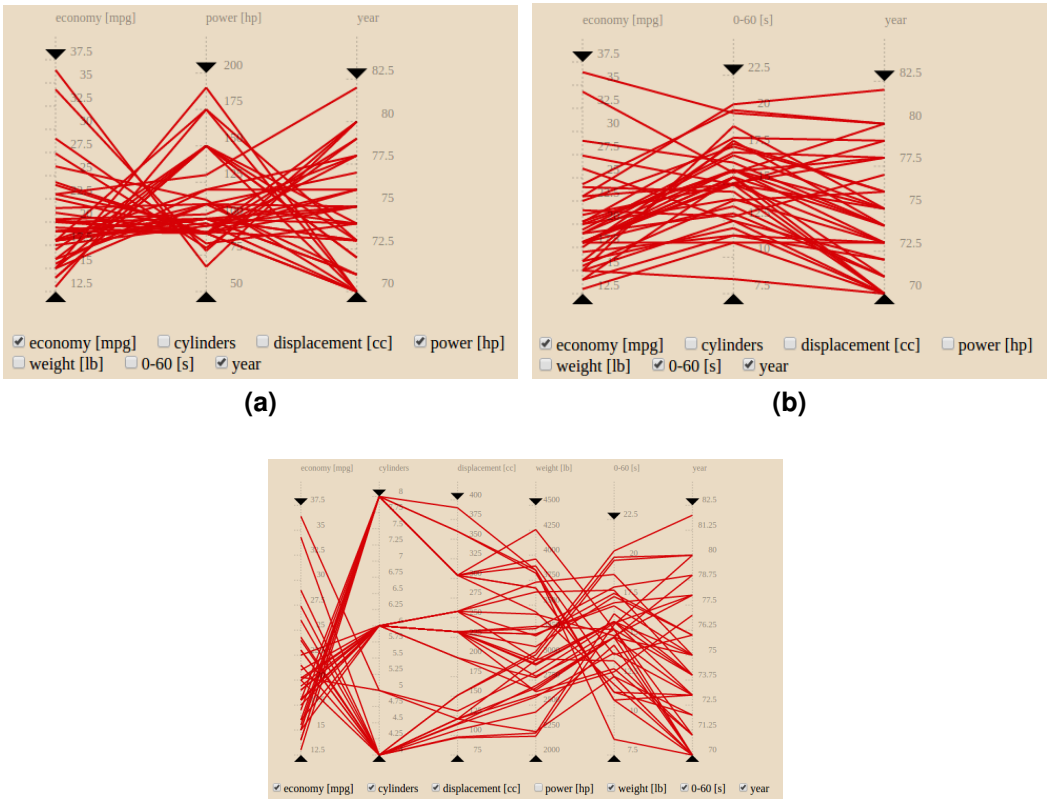


Figure 5.10: Combination of Responsive Behaviour and Interactive Selection of Displayed Dimensions

```
<div class="chart">
      <div class="ct-chart ct-golden-section"></div>
</div>
...
<section>
    <div class="ct-select-dimensions"></div>
</section>

...

var options = {
      selectDisplayedDimContainer: ".ct-select-dimensions",
      ...
  };
  new Chartist.ParallelCoordinates('.ct-chart', data, options,
     responsiveOptions);
```

Figure 5.11: Usage of Select Displayed Dimension Feature

```
<ul>
  <li><input id="..." type="checkbox" checked>Dimension Name</li>
  <li><input id="..." type="checkbox">Dimension Name 2</li>
  ...
</ul>
```

Figure 5.12: Generated HTML to Select Displayed Dimensions

```
.ct-select-dimensions ul
{
  margin: 0;
  padding: 0;
  list-style-type: none;
}

.ct-select-dimensions ul li {
  display: inline;
  padding-right: 1em
}
```

Figure 5.13: Possible CSS Definition

5.4.2 Select Values of Interest

Parallel Coordinate charts often contain a lot of data records cluttering the chart area. The goal of this feature is to provide UI elements to give the user the opportunity to define value ranges on each dimension. Data records that are outside this interval are filtered out (see figure 5.14).

This functionality required the ability to drag elements across the chart area. As a result the function `makeDraggable` was added to the core module. It is a generalization of the dragging requirement that is needed for the "select value of interest" function. Beside the element that should be draggable it has three more arguments: the enclosing svg element and two event handlers: `onMoveAction(element, dx, dy)` and `onReleaseAction(element)`. This means that this function can be used to easily add dragging behaviour to any element on the chart and defining event handlers that are tailored to the underlying use case. During the development of this function, two problems arose that had to be solved:

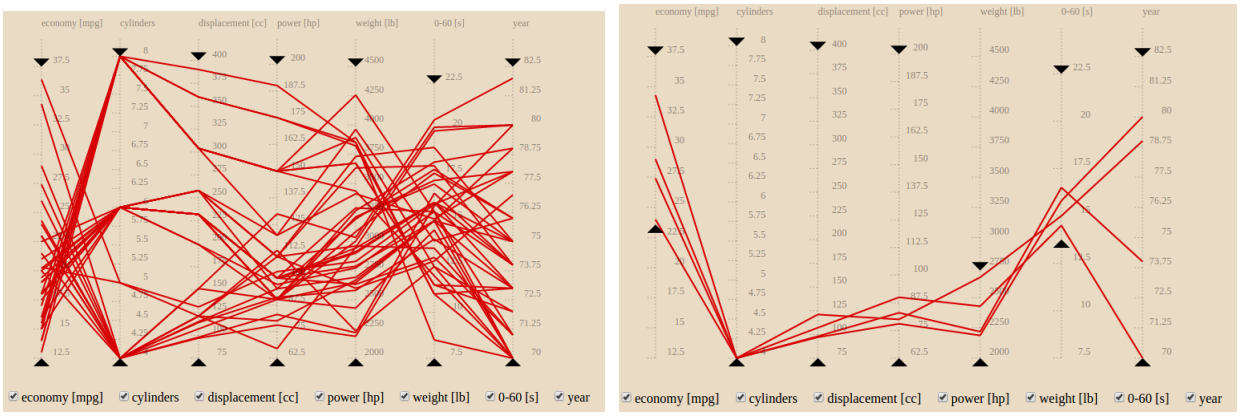
- dragging an element conflicted with text selection of labels inside the chart
- if the `onMouseOut` event was attached to the element itself, dragging was often prematurely interrupted if the user moved the mouse too fast.

The first issue was solved with CSS by temporarily deactivating text selection during the drag event (see code listing 5.16). The other one by attaching the event handler not to the element itself, but to the enclosing svg node.

The next step was to determine the value on the dimension based on the pixel position of the ruler. Therefore it was necessary to invert the Core function `projectPoint`.

As already mentioned in the chapter 5.1, the maximum value of a dimension is not always on the same position. Therefore the minimum which is located at `chartRect.y1` is the anchor point for this calculation. This result is stored in the dimensions object to recover the position after a chart redraw triggered through a resize or an alteration of the displayed dimensions.

The last step was to filter all data records that fall outside the selected intervals. This is done without redrawing the entire chart. The function `filterDataRecords` (see code listing 5.15) iterates over all dimensions and data records and sets the style attribute accordingly. Thereby, only displayed dimensions are



(a) Unfiltered **(b)** Filtered

Figure 5.14: Select Values of Interest

```
1   function filterDataRecords(data, bounds) {
2
3     data.series.forEach(function(dataRecord, index) {
4       var hide = false;
5
6       //check all dimensions that are displayed currently
7       dimensions.dimensionIndex.forEach(function(dimIndex) {
8         var min = dimensions.minValues[dimIndex];
9         var max = dimensions.maxValues[dimIndex];
10        if(dataRecord[dimIndex] < min || dataRecord[dimIndex] > max) {
11          hide = true;
12        }
13      });
14
15      var style = hide ? "display:none" : "";
16      document.getElementById("dr"+index).setAttribute("style", style);
17    });
18  }
```

Figure 5.15: `filterDataRecords` Function

taken into account. This means if one filters out data records based on one dimension and hides this dimension in the next step, all data records will again be displayed.

5.5 Additional Features

The library provides additional features for parallel coordinates in order to make it easier to analyse data.

5.5.1 Mean visualization

It is possible to show the arithmetic mean. The arithmetic mean, denoted as $\overline{x}$ is the sum of all data records $x_1, x_2, ..., x_n$, of each dimension divided by the number of records, denoted by n. Therefore, the mean of each dimension is calculated by:

$$\overline{x} = \frac{x_1, x_2, ..., x_n}{n}$$

```
.ct-unselectable {
  -webkit-touch-callout: none;
  -webkit-user-select: none;
  -khtml-user-select: none;
  -moz-user-select: none;
  -ms-user-select: none;
  user-select: none;
}
```

Figure 5.16: CSS Options to Prevent Text Selection During Dragging

The value is displayed as a slightly thicker green line. As already mentioned in section 5.3 the style of the mean visualization in the chart is defined in the SASS settings file. The CSS class used is called `ct-mean`. The library uses the following configuration value for showing and hiding the mean:
`showMean : Boolean`.

Figure 5.17 shows a simple example of a parallel coordinates chart including the arithmetic mean.

5.5.2 Histogram visualization

The histogram is a graphical representation of the distributed data. Each dimension is divided into a series of small intervals, and each data point is assigned to the corresponding interval.

As in section 5.5.1 already mentioned the style of the histogram is defined in the SASS settings file. The CSS class is `ct-histogram`.

The configuration values for the histogram are:
`showHistogram : Boolean`
`histogramPartition : Integer`
Where `histogramPartition` defines the number of intervals.

In figure 5.18 a parallel coordinates chart with a histogram is drawn.

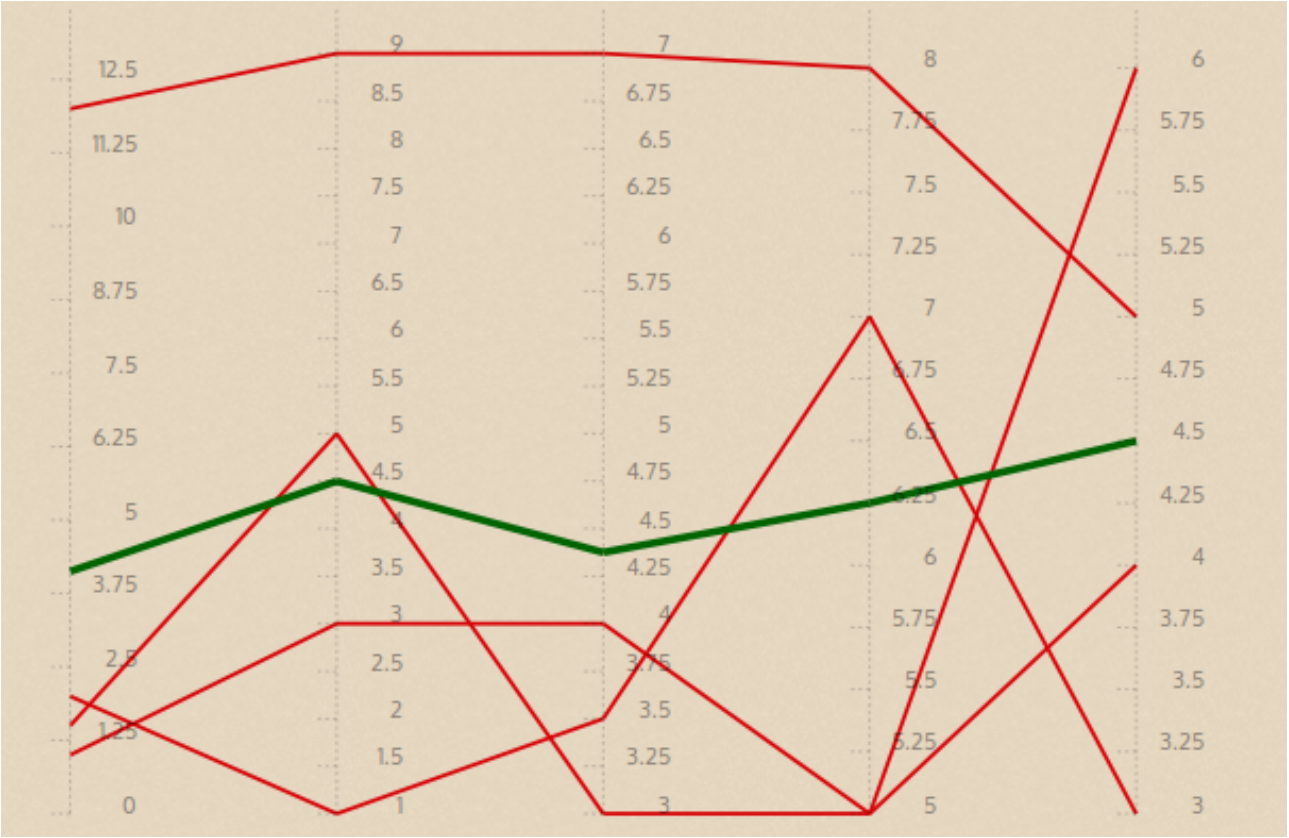


Figure 5.17: Parallel Coordinate with Arithmetic Mean

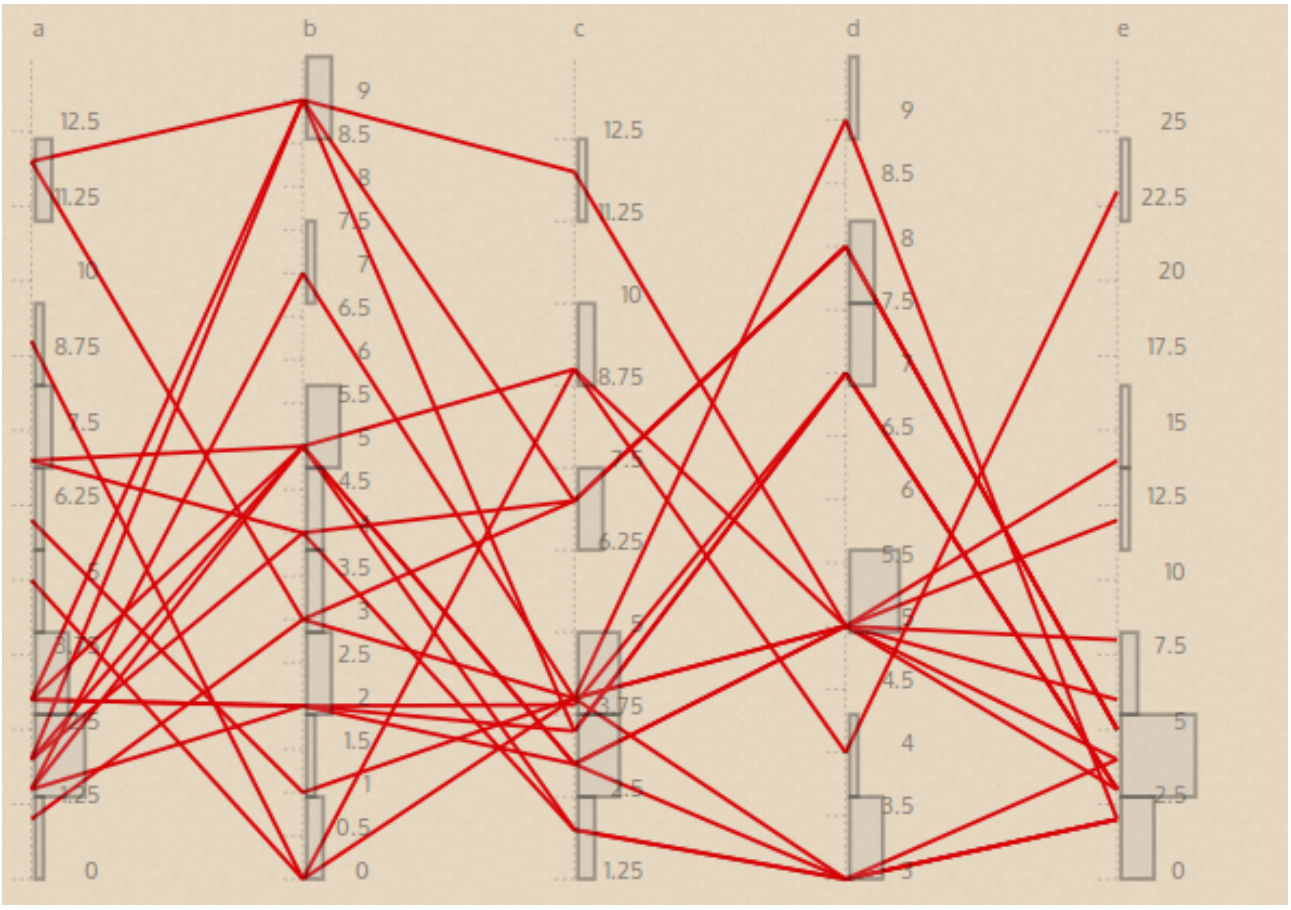


Figure 5.18: Parallel Coordinate with Histogram

Chapter 6

Conclusion

As a conclusion to this whole process of enhancing the Chartist library, we can say that the implementation of all those features was quite challenging. The 'axis ticks labels rotation' and 'vertical/horizontal' features were probably more effort than it seemed in the beginning, when we where collecting the ideas over what would be a nice enhancement to Chartist. Problems predicted in the beginning turned out to be harmless and quickly solved, while things we didn't even think about became a huge and time consuming challenge. Like expected, small adaptations could cause vast impact on the rest of the created chart. Finding these coherences and consider them in our implementation was one of main issues while this assignment.

The parallel coordinates were expected to be a lot of work since they had to be created from the scratch. While introducing this new chart type to Chartist with all responsive options typical for this charting library, we came up with more and more useful features for this chart type, which is rather complex even in its simplest form. After appending all this features to parallel coordinates we ended up with a very powerful and hopefully also useful, to all future Chartist users, new chart type.