Vision, Modeling, and Visualization (2024)
L. Linsen and J. Thies (Eds.)

A Framework for Axis Breaks in Charts

Rasmus Thorsøe[1], Peter Locher[1], Harith Rathish[1], and Hans-Jörg Schulz[1]

[1]Department of Computer Science, Aarhus University, Denmark

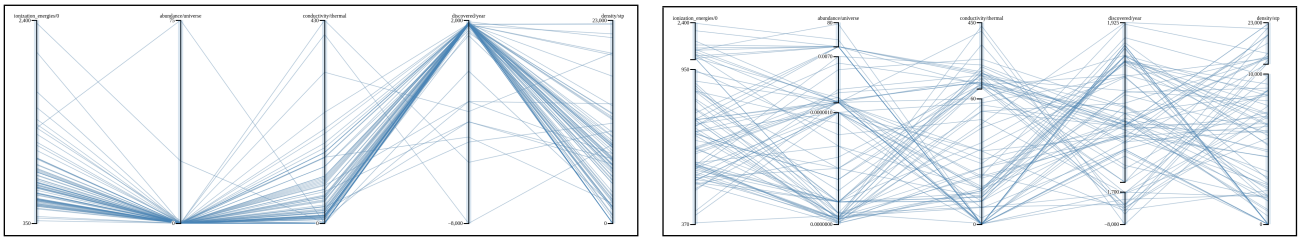

Figure 1: *A parallel coordinates plot without axis breaks (left) and with axis breaks as computed by our framework (right). The difference shows how axis breaks help to better distribute the highly skewed data across the axes, so that more details and nuances become visible. Shown dataset: Properties of chemical elements [Wea22].*

Abstract

Axis breaks are used in charts, for example, to reduce whitespace, to accommodate outliers, or to show data at different scales. Proposed in the 1980s, axis breaks have not gotten much attention since then in terms of what characterizes "good" breaks, how many of them to introduce, and where to best place them? To answer these questions, we propose a five-step framework that specifies (1) the number of breaks, (2) their position, (3) the scaling of the resulting subaxes, (4) the "niceness" of the breaks, and (5) the formatting of the breaks. To apply this framework, we introduce a new metric, called skew*, to quantify how unevenly distributed points are along an axis. Skew is then used as a cost function to formulate the search for optimal axis breaks as a clustering problem, which we solve by applying a dynamic k-means algorithm. We apply our framework specifically to Parallel Coordinate Plots and compare our algorithmic solution to established methods like percentile breaks and Jenks natural breaks. An interactive testbed to try our framework as well as its source code are made freely available.*

CCS Concepts

• ***Human-centered computing*** → ***Visualization techniques; Information visualization;***

1. Introduction

The axes of a chart play a hugely important role as they enable the reader to read off absolute values and to gauge relative distances with respect to data items' positions. It is hence no surprise that for many aspects of axis design, sophisticated algorithms have been proposed in the past – including, but not limited to the scaling [WF02] and rescaling [FCW21] of axes, their labeling [TLH11], and their ordering [LHZ16].

One aspect of axis design that is not as common are axis breaks. Axis breaks are deliberately introduced subdivisions of an axis into two or more subaxes for various reasons. For example, when visualizing highly skewed data, data points can tend to group so closely together that they are indistinguishable to the naked eye (see Fig. 1 left). Introducing axis breaks can "un-skew" the data and provide a view into the otherwise very condensed regions of the data space. Other reasons can include the accommodation of outliers within the same view, the reduction of unused whitespace, the display of data at different scales, or the handling of data at different orders of magnitude (cf. Fig. 2).

While having been formally introduced by Cleveland as early as 1984 [Cle84], to this date only a few specialized approaches exist that introduce axis breaks automatically according to given constraints, but otherwise without user input or intervention. Among them, notable mentions are Hao et al.'s *variable binning* approach for scatter plots [HDS*10] and Andrienko & Andrienko's *quartile-based axis normalization* for parallel coordinate plots [AA01]. Apart from these, the literature on this topic is scarce. Isenberg et al. conducted a user study on *dual-scale charts* [IBDF11] that compares line charts in which the horizontal axes are distorted and broken in various ways (see Fig. 2 middle). For bar charts,

DOI: 10.2312/vmv.20241198

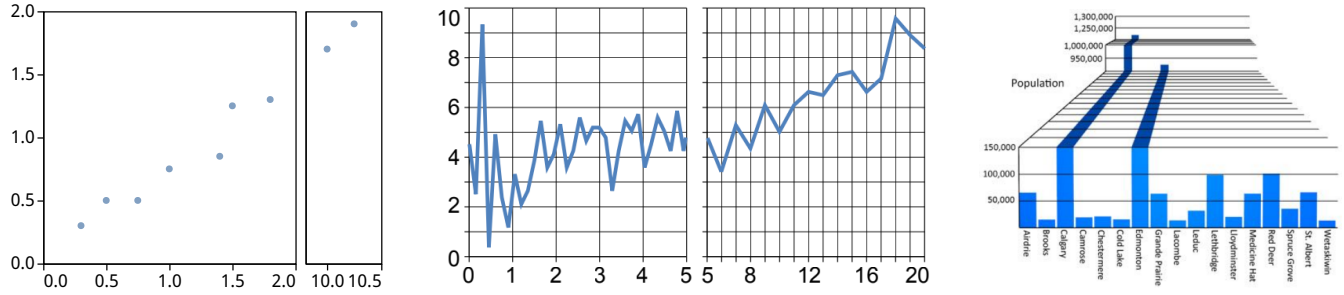

Figure 2: *Axis breaks used in three different charts for three different reasons. Left: An axis break in a scatterplot that removes unused whitespace, but keeps the scaling of both subaxes the same. (Recreated from [Cle84] using Vega-Lite). Middle: A broken horizontal axis of a line chart to use different scaling on each resulting subaxes. (Reproduced with permission from [IBDF11], ©IEEE 2011). Right: A perspective bar chart with two axis breaks along the vertical axis to accommodate values at different orders of magnitude. (Reproduced with permission from [MESW21], ©MacTavish et al. 2021)*

which are widely known not to lend themselves to axis breaks, inventive representations like MacTavish et al.'s *Stepped Perspective Charts* [MESW21] still find ways to make axis breaks work (see Fig. 2 right). Yet none of these approaches give any heuristic or algorithm to automatically determine the breaks.

We propose a framework for automatically introducing breaks in axes. Through its general nature, our framework is able to cater to a variety of different scenarios and constraints for axis breaks:

- It can observe a predetermined, manually provided number of breaks or automatically determine an optimal number itself.
- It can scale the resulting subaxes to the same size or make their size proportional to the number of contained data items/values.
- It can accommodate different styles of breaking the data range (tights vs. nice) and its visual representation (concatenated vs. broken).

Our framework is introduced in the following section and specific improvements to its steps are discussed in the sections thereafter. We exemplify our framework for different axis-based chart types throughout the paper and specifically introduce, evaluate, and discuss axis breaks for Parallel Coordinate Plots.

2. A General Framework for Axis Breaks

To capture the whole process of generating axis breaks, we propose a five-step framework in which each step deals with one specific consideration. Each of these steps has its own set of currently used methods that can be combined in various ways to achieve different outcomes. For the first two steps (in combination) as well as for the fourth step, we will furthermore propose improved algorithms in Sections 3 and 4, respectively. In case of multiple axes, the framework is applied to each axis individually.

Step 1: Determine Break Number. This step determines the number of subaxes k into which an axis will be broken by introducing $k-1$ axis breaks. Current methods either rely on a manually adjusted or fixed choice for k. An example of the former are Hao et al's variable binned scatterplots [HDS*10] that use rather high numbers of k (between 6 and 10) to create a fine-grained bin structure. An example for the latter is Andrienko & Andrienko's quartile-based normalization using fixed $k = 4$ to yield one subaxis per quartile [AA01]. This approach is exemplified together with a quintile-based normalization in Figures 3(c+d).

Step 2: Determine Break Positions. Given the number k of subaxes, this step determines for each data point which subaxis it belongs to. This is done by assigning the dataset X to k disjoint segments $C_1, ..., C_k \in \mathcal{C}$ each representing a section of the broken axis – i.e., a subaxis. A straightforward approach is to use, for example, k-means for finding the $k-1$ largest gaps among the data values and break the axis there (see Figure 3(e)). Other approaches are those used for *classification* in thematic mapping, such as the Jenks / Fisher Natural Breaks method [Jen77, Fis58]. Interactive, user-driven classification as suggested by Andrienko et al. [AAS01] is also possible – effectively constituting a manual positioning of the breaks that is informed by the underlying data distribution. Another alternative proposed by Unger and Schumann [US09] is the hierarchical merging of data values into intervals until the given number of k intervals is reached.

Step 3: Scaling of Subaxes. This step determines how much visual space is assigned to each of the subaxes $C_1, ..., C_k$. For example, Hao et al.'s variable binned scatterplots [HDS*10] use *equal size scaling* of all subaxes on the x-axis, and *data size scaling* of all subaxes on the y-axis. The former scales all subaxes to the same size regardless of how many data items each shows, while the latter scales all subaxes to a size proportional to the number of data items shown per axis. Andrienko & Andrienko's quartile-based axis subdivision uses a fixed scaling, but allows users to interactively adjust the scaling of the subaxes to their needs. It is also possible to preserve the original linear scaling of the subaxes. This is used for cutting empty whitespace (i.e., gaps in the data value distribution) using the trimming in the next step, while keeping the subaxes on par. Cleveland [Cle84] uses this method in his original paper on axis breaks in some figures.

Step 4: Trimming of Subaxes. This step deals with the interval between two consecutive subaxes. One option is removing the range between the maximum of one subaxis C_n and the minimum

of the subsequent subaxis C_{n+1} (*tight ranges*) as Cleveland [Cle84] does or to round ranges off to more readable numbers with fewer digits (*nice ranges*). Another option is to keep subaxes continuous, as in the approach by Andrienko & Andrienko [AA01].

Step 5: Formatting of Subaxes. This final step governs the appearance of the subaxes with their scaling and range bounds. For this, a number of practical details must be considered that also depend on the axis breaking process. They include density and style of the tick marks of each axis section, and visual representation of the breaks. Representational choices include, for example, the use of full scale breaks (i.e., small gaps between subaxes) vs. concatenation of axes with an adjustment of the spacing of the tick marks to indicate the different granularities of different subaxes, similar to Isenberg et al.'s use of grid lines [IBDF11].

Breaking up the problem into these five steps allows us to mix & match different methods with each other depending on the data, the plot type, and the usage scenario. An example of this is given in Figure 4, which combines the look&feel of Hao et al.'s variable binned scatterplots with the quartile-based axis breaks from Andrienko & Andrienko and explores two different scaling options side-by-side. To try some of these combinations interactively, one can use our interactive web-based testbed at https://vis-au.github.io/axisbreaks/

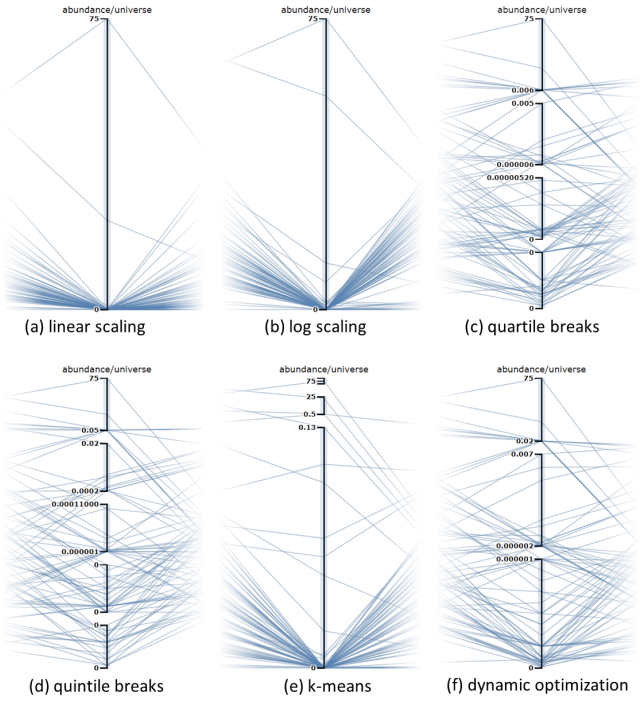

Figure 3: *Different options for showing the highly skewed distribution of chemical elements in the universe along a parallel coordinates axis: (a) linear scaling shows merely that two elements (hydrogen and helium) dwarf all the other elements; (b) log-scaling [Fie17] brings out the next two elements (oxygen and carbon); (c) quartile-based axis breaks [AA01] distribute the data more uniformly and show more details of the lower quartiles; (d) generalizing the quartile-based breaks into percentile-based axis breaks – here showing quintiles; (e) breaking the axis at the largest gaps using a k-means clustering to reclaim some empty space brings out the long tail of the distribution, but still condenses the bulk of data at the bottom of the axis; (f) axis breaks produced by our novel dynamic optimization approach.*

In addition, this stepped approach to axis breaks also enables us to selectively improve methods for some of these steps. In the following two sections, we do so for the combination of step 1 and 2 by introducing an optimization for computing the number and position of axis breaks, as well as for step 4 by extending Talbot et al.'s tick mark algorithm [TLH11] to produce human-readable, "nice" cutoffs for the subaxes.

3. A Method to Compute Break Number (Step 1) and Positions (Step 2)

There are two main issues addressed by axis breaks: One are highly skewed value distributions along an axis where a few extreme values compress all other data values into a very small axis interval. The other are value distributions across multiple orders of magnitude, which can be understood as a generalization of the skewed distributions: Where for the skewed distributions, the axis is broken into a subaxis containing the extreme values and another subaxis containing the remaining bulk of the data, doing this for the orders of magnitude will just lead to the same problem reoccurring at the next lower level of magnitude. This is illustrated in Fig. 5, showing why simple solutions for determining axis breaks, such as seeking the largest gaps do not necessarily lead to better visualizations.

So, instead of seeking local features like gaps in the data to place axis breaks, we advocate for a more holistic view of the axis. What we would like from an axis break is that the axis is afterwards less skewed and more uniformly distributed than before. Hence the fundamental idea of our computational method for axis breaks is simple: increase the number of subaxes k until the skew of an axis no longer sufficiently decreases. In the following, we describe

- how we formally capture this notion of skew and use it in a cost function,
- how we use this cost function to define a stopping condition for this process – i.e., what it means to *no longer sufficiently decrease*, and
- how we use both in a dynamic optimization algorithm to find suitable axis breaks.

3.1. Cost Function: Skew

We use the term *skew* γ for describing how much the data distribution differs from a uniform distribution when applying a given mapping to the data. This is different from statistical skewness that describes how asymmetric a distribution is around its average, but has the same property that a uniform distribution of points has skewness and skew of 0.

Concretely we look at the distance of a point to its uniform position as defined by the following mapping:

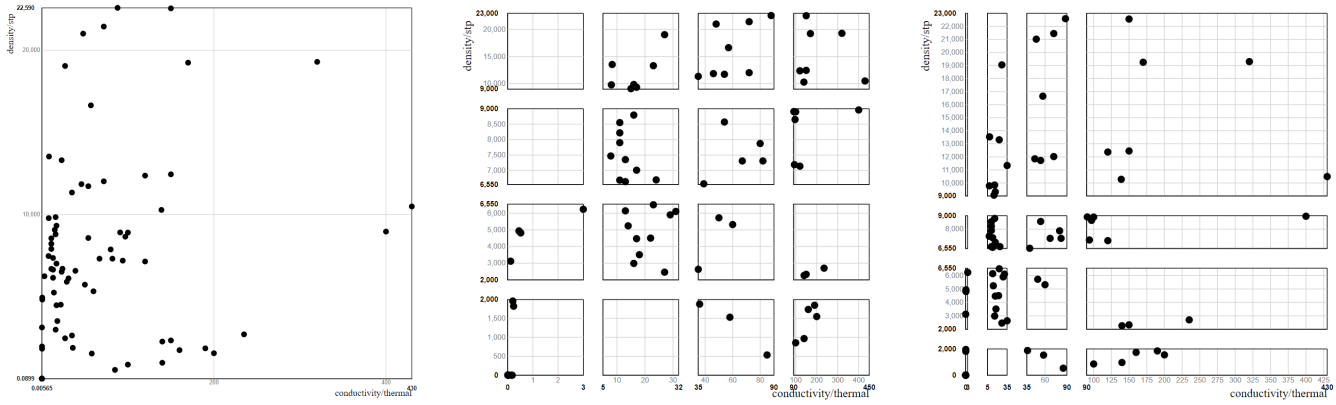

Figure 4: *Two variable binned scatterplots (middle, right) created using the quartile-based breaks from Andrienko & Andrienko's approach [AA01] on the original scatterplot (left). The plot in the middle uses subaxes of equal size, which is sensible for quartiles, but hides the "variable" property of the binning. The plot on the right does not rescale the subaxes, but maintains the original linear overall scale. This shows much better how the quartiles are distributed across the value ranges with the long tail for both attributes resulting in the big bin at the top right, but it also makes it harder to identify and interact with the data points in the lower quartiles.*

Definition 1 The uniform mapping of the i-th point of X sorted by increasing value, $x_i \in X$ with respect to X is

$$\pi_u(X, x_i) = \frac{i}{|X|}$$

The uniform mapping π_u maps $x_i \in X$ to its location in $[0,1]$ when X is distributed uniformly. This is done simply by mapping points to their sorted-order indices and normalizing. Skew can then be defined w.r.t. the uniform mapping in the following way:

Definition 2 The skew of mapping $\pi : X \to \mathbb{R}$ on $x \in X$ is

$$\gamma_\pi(X, x) = |\pi(x) - \pi_u(X, x)|$$

For point sets $Y \subseteq X$ we define

$$\gamma_\pi(X, Y) = \sum_{y \in Y} \gamma_\pi(X, y)$$

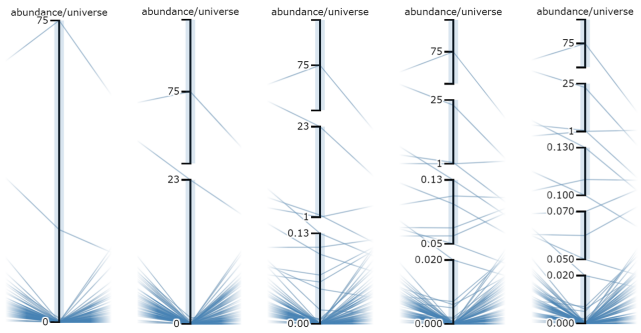

Figure 5: *Introducing axis breaks ($1 \leq k \leq 5$) for data across orders of magnitude will recursively put extreme values on their own subaxis with new extremes appearing at the next lower level of magnitude that still condense the bulk of data at the bottom.*

After analyzing different ways in which to use skew in our approach, we ended up using the squared skew:

Definition 3 The squared skew of mapping $\pi : X \to \mathbb{R}$ on $x \in X$ is

$$\gamma^2_\pi(X) = \sum_{x \in X} (\pi(x) - \pi_u(X, x))^2 = \sum_{x_i \in X} \left(\pi(x_i) - \frac{i}{|X|} \right)^2$$

We use the squared skew as a cost function *cost* as it better balances skew reduction with the incurred displacement from the unbroken, linearly scaled axis. For example, when deciding between two possible axis breaks – one that displaces a single data point by 10 pixels and one that displaces five data points by 2 pixels each to get them closer to their position on the uniformly distributed axis – the linear skew would weigh both options equally, but the squared skew would clearly favor the second option with many smaller changes instead of one big change. This is also intuitively what we want: to create a better distribution of data points without displacing them too far from their original positions.

3.2. Stopping Condition: Skew vs. Fragmentation

We can then use the squared skew as a cost function to measure the "goodness" of an axis – i.e., how well it distributes the data values along the axis – regardless of whether it is broken and rescaled or not. This leads to the following stopping condition in which the costs are normalized by the number of data items n, so that the same threshold θ can be applied regardless of dataset sizes:

$$\frac{\text{cost}(k-1) - \text{cost}(k)}{n} \leq \theta \tag{1}$$

We set $\theta = (1-\phi)^2 + 0.001 \cdot k$, where ϕ is the degree of fragmentation ranging from 0 (no breaks) to 1 (many breaks). This makes the threshold θ dependent on the number of breaks, as each break introduces additional visual clutter, which is only warranted if it yields in return a large enough skew improvement. How large that improvement has to be is adjusted through the parameter ϕ.

3.3. Dynamic Optimization: Axis Breaks through Clustering

Our optimization algorithm places the axis breaks by minimizing the cost function of the squared skew (Section 3.1) and determines the number of axis breaks through the stopping condition weighing improved skew vs. incurred axis fragmentation (Section 3.2). It thus carries out steps 1 & 2 of our framework in one go. If a predefined number of $k-1$ breaks are to be placed – i.e., carrying out only step 2 – this is simply done by setting the stopping condition accordingly.

To carry out this optimization, our algorithm is based on a dynamic algorithm for the k-means problem developed independently by different researchers [WS11, Wu91]. An overview of the dynamic approaches was given by Grønlund et al. in 2017 [GLMN17]. Our algorithm is based on the $O(n^2k)$ version that uses prefix sums to improve the recursive computation of the basic $O(n^3k)$ dynamic algorithm.

Using a dynamic algorithm requires the use of induction to upgrade a clustering for n points with k clusters to a clustering with more points or more clusters without recomputing everything. This is done by dynamically computing the optimal clustering w.r.t. skew of the first i points X_i into m clusters using solutions of subproblems. To find the optimal solution for i and m, the subproblems with the first $j < i$ points and $m-1$ clusters are used.

Such optimal clustering of X_i restricted by its last cluster starting in $j+1$ is denoted by us as $F_m[j,i]$. It is computed by combining the skew of the new cluster m with the optimal cost of the $m-1$ clustering of X_j.

$$F_m[j,i] = (i-j)^2 \cdot \gamma^2_{\pi_0}(X_{j+1,i}) + j^2 \cdot D[m-1,j]$$

With this we can compute the cost of the m,i-sub problem $D[m,i]$:

$$D[m,i] = \min_{\mathcal{C}} \gamma_{\pi_{\mathcal{C}}}(X_i) = \frac{1}{i^2} \cdot \min_{1 \le j < i} F_m[j,i]$$

As a base case $D[1,i] = \gamma^2_{\pi_0}(X_i)$ is computed directly for all i in $O(n^2)$ time. This gives us the ingredients for a dynamic algorithm of time complexity $O(n^3k)$ that simply fills out a $n \times k$ table C using the recursion formula. The algorithm is upgraded to $O(n^2k)$ time complexity by using prefix sums to compute $\gamma^2_{\pi_0}(X_{j+1,i})$ in constant time for each $F_m[j,i]$. This is captured in Algorithm 1, which proceeds just like the dynamic 1D k-means algorithm presented in [GLMN17, WS11].

The D table provides the skew of the optimal solution. To yield the clustering $\mathcal{C}$, we also maintain a size $n \times k$ table T containing all the break points j for which $F_m[j,i]$ is minimal. This can be traversed to extract the clustering (see Algorithm 2). The stopping conditions can be integrated by computing D row by row and for each row m use $D[n,m]$ to decide whether to continue or to output the clustering for $D[n,m-1]$.

The result of our algorithm is a balanced, density-based positioning of axis breaks, as it is exemplified in Figure 3(f). A benchmark comparing its output quality for Parallel Coordinate Plots is given in Section 5.1.

Algorithm 1 Optimal Clustering by Skew

```
Init D and T as arrays of length n × k with 0 in all cells
Init length n array A with cumulative sums of x_i
Init length n array A^2 with cumulative sums of x_i^2
Init length n array H with cumulative sums of i · x_i
for i = 1 to n do
    D[1,i] = γ²_π0(X_i)
for m = 2 to k do
    for i = 1 to n do
        best_so_far = ∞
        for j = m to i do
            cost_X_j = D[m−1, j]
            cost_X_{j+1,i} = γ²_π0(X_{j+1,i})
            cost = j²· cost_X_j + (i−j)² · cost_X_{j+1,i}
            if cost < best_so_far then
                best_so_far = cost
                break_index = j+1
        D[m,i] = 1/i² · best_so_far
        T[m,i] = break_index
return C, T
```

Algorithm 2 Report Clusters(T)

```
row = k
col = n
while col > 0 do
    j = T[row, col] + 1
    report cluster [x_j, x_col]
    col = T[row, col]
    row = k − 1
```

4. A Method to Yield Sensible Subaxis Boundaries (Step 4)

Having established the number and position of the break points for the axis breaks does not yet yield the start- and endpoints of the corresponding subaxes. Naïvely, one can trim subaxes to start with the lowest value contained and end with the highest, but that usually results in very awkward interval boundaries that are not easy to label, read, and remember. To solve this issue, we propose a new method for determining simple and readable boundaries for subaxes. The difference between tight and "nice" subaxis boundaries is exemplified in Figure 6 for a beeswarm plot.

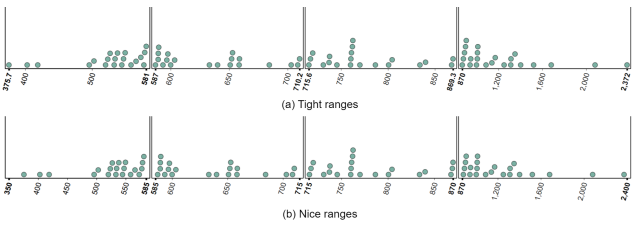

Figure 6: *A beeswarm plot showing the ionization energies of the chemical elements on four differently scaled subaxes. (a) shows axis breaks with tight ranges – i.e., exact boundaries; (b) shows axis breaks with "nice" ranges – i.e., slightly rounded boundaries to make them more human-readable.*

Our method to compute sensible subaxis boundaries takes inspiration from the *Extended Wilkinson's algorithm* proposed by Talbot et al. [TLH11] for the placement of tick marks along axes. Nice values for range bounds are slightly simpler to define than for tick marks, as there are no dependencies between one range bound on other range bounds, as there is for tick mark numbers. Our only constraints are that all points of a cluster must fall within its range bounds and that the range bounds cannot overlap the bounds of the neighboring cluster ranges. Wilkinson's density objective and system of generating nice values from offsets and skips are therefore not necessary in our case.

We generate our candidates independently for each range bound using a list of divisors $Q = [1,2,5]$. With these, we divide decreasing powers of 10 starting with the nearest power of 10 below the bound. For example, the number 363 would have 100 as its nearest power of 10 followed by 10, 1, which gives the list of numbers $[100,50,20,10,5,2,1]$. Each of these numbers is then turned into a candidate by repeating it until it hits the range bound: $[300,350,360,360,360,362,363]$ – that is the closest multiples of $[100,100/2,100/5,10,10/2,10/5,1]$ to 363.

More formally, we denote a range bound C with $\min(C) = r_1, \max(C) = r_2$ so that its data range is $[r_1,r_2]$ and its length $||C|| = r_2 - r_1$. We define its magnitude to be $m = \lfloor \log_{10} ||C|| \rfloor$. The list of candidate nice numbers C is generated by dividing 10^i by the elements of $Q = \{1,2,5\}$ for i descending from m. This produces a list of step sizes

$$G = 10^m, \frac{10^m}{2}, \frac{10^m}{5}, 10^{m-1}, \frac{10^{m-1}}{2}, \frac{10^{m-1}}{5}, \ldots \tag{2}$$

For each of these step sizes $g_1,\ldots,g_j,\ldots$, we have a sequence $0, g_j, 2g_j, 3g_j, \ldots$. From g_j, we produce a candidate nice lower bound l_j of r_1. l_j will be the first number in that sequence to the left of r_1. Similarly for r_2, we have a candidate nice upper bound u_j that is the first number in the sequence of g_j that is to the right of r_2. They are computed fast with floor and ceiling division.

$$l_j = \lfloor r_1/g_j \rfloor \qquad u_j = \lceil r_2/g_j \rceil \tag{3}$$

To pick the best boundary from the candidates, we minimize two objectives dependent on the choice of candidate. First, we prioritize the candidates with the simplicity objective, which is defined in the same way as in Wilkinson's algorithm. Let $|C|$ be the number of candidates and j be the index of candidate b_j in the list. Then the "simplicity" of a number can be introduced as follows:

Definition 4

$$simplicity(b_j) = 1 - \frac{j}{|C|}$$

It is clear from the example that the higher simplicity a candidate has, the larger the gap between the range bound and the closest data point. Such a gap is introducing white space into the display space dedicated to the cluster, thereby pushing the points in the cluster closer together. To counterbalance this, we also use a simpler version of Wilkinson's coverage objective that we call "tightness" as it rewards choosing a bound that fits the data tightly. As we are now optimizing the bounds independently, tightness does not need the squared difference to balance Wilkinson's coverage on both sides of the range, as it was reasoned for by Talbot et al. in their Extended Wilkinson's algorithm [TLH11].

Definition 5 Denote the length of the range of cluster C by $||C||$. The tightness of a nice bound b_j for a range bound r_1 or r_2 is then

$$tightness(b_j) = \left(1 - \frac{|b_j - r_1|}{||C||}\right) \cdot \frac{|C|}{|X|}$$

We can optimize these two objectives together by balancing them with a tightness weight w. Our objective becomes the following:

$$(1-w) \cdot simplicity(b_j) + w \cdot tightness(b_j) \tag{4}$$

Since the number of candidates is limited, the objective can be optimized naïvely by computing the score from Equation 4 for each candidate and picking the one with the highest score. The choice of $w = 0.9$ worked well for the datasets tested in this project.

5. Use Case: Axis Breaks for Parallel Coordinates

Parallel Coordinate Plots represent data points as lines [Ins09, HW13]. When there are no axis breaks, a skewed data distribution may cause multiple lines to converge in a small space, as shown in the abundance and discovered year axes in Figure 7 (top-left). Such overplotting makes it difficult to trace an individual line across multiple axes and to observe patterns hidden in those regions.

While the problem of overplotting in Parallel Coordinate Plots has been addressed in various ways in the literature – from sampling [EBD05] to aggregation [RSL*19], axis breaks are rather un-

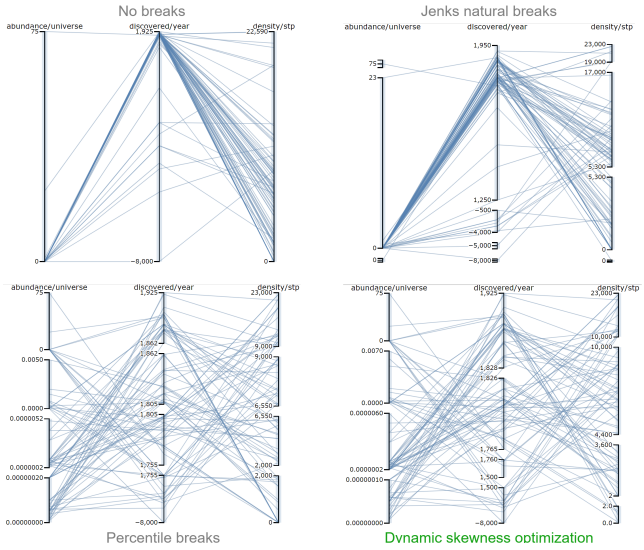

Figure 7: *Introducing axis breaks into Parallel Coordinates. In the plot with no axis breaks (top-left) there is significant overplotting on the "abundance/universe" and "discovered/year" axes. Jenks natural breaks (top-right) cuts the largest gap on the abundance axis – between 23 and 75, but also introduces multiple breaks on the year-discovered axis. Percentile breaks (bottom-left) assign each quartile its own subaxis. However, we can still see overplotted lines going from "discovered/year" to the lower subaxis in "density/stp". Our dynamic optimization (bottom-right) separates these lines even further, while keeping the same number of breaks.*

common for Parallel Coordinates. To the best of our knowledge, the only published approach is by Andrienko & Andrienko [AA01] who proposed to center the median on the axis and to position the lower and the upper quartiles equidistantly below and above this center point, respectively. Then each of the resulting four subaxes is scaled linearly to the minimum and maximum values contained in each. This has the effect that outliers will only compress the first and fourth quartile, while the interquartile range occupies a guaranteed space in the middle of the axis.

By using our dynamic programming approach for positioning axis breaks, we extend beyond Andrienko & Andrienko's approach by allowing a flexible number of breaks at flexible positions so as to minimize the number of breaks, while maximizing their effect. We call the resulting plots *Split Parallel Coordinates (SPC)*. The following section compares them to Jenks natural breaks and to Percentile breaks – i.e., Andrienko & Andrienko's approach.

5.1. Benchmarking

To gauge the results w.r.t. the trade-off between improved data distribution and introduced distortion (as compared to the original linear scaling without breaks), we measure two characteristics:

- **Overplotting:** The overplotting metric was introduced by Dasgupta and Kosara in their work on *pargnostics* [DK10, Sec.3.8]. It basically measures how many lines are drawn on top of each other in the space between the axis in question and its neighbor(s). The lower this metric, the better the plot.
- **Distortion:** The distortion metric quantifies how much the axis breaks displace the data points on the axes respective to their original position. The lower this metric, the better the plot as it requires less mental effort to use it.

We applied both metrics to each axis of the Parallel Coordinate Plot independently and averaged their results. All plots had a maximum height of 1400 pixels, with axis breaks having a gap of 30 pixels. The data used were the properties of chemical elements [Wea22] that are also shown in Figure 7. We compared against Jenks natural breaks and Andrienko & Andrienko's approach (percentile breaks at 25%, 50%, and 75%), since both approaches introduce axis breaks and re-scale the resulting subaxes at the same time, making them most similar to our approach.

The results shown in Figure 8 indicate that at the cost of introducing slightly less distortion than percentile breaks, our algorithm reduces the number of overplotted lines by the largest margin, followed by percentile breaks and Jenks natural breaks. This observation holds also for other tested datasets [Raj18, Num22] with the difference being more pronounced in highly skewed data attributes and less so in more uniformly distributed ones.

5.2. Discussion/Limitations

Our algorithm improved the overplotting score of Parallel Coordinates, making it easier to trace individual data points and reveal patterns in overplotted regions. The per-axis computation of the breaks makes the found breaks indifferent to the order of the axes as well as to how often individual axes are shown. Thus, common basic interactions with Parallel Coordinates are expected to translate well

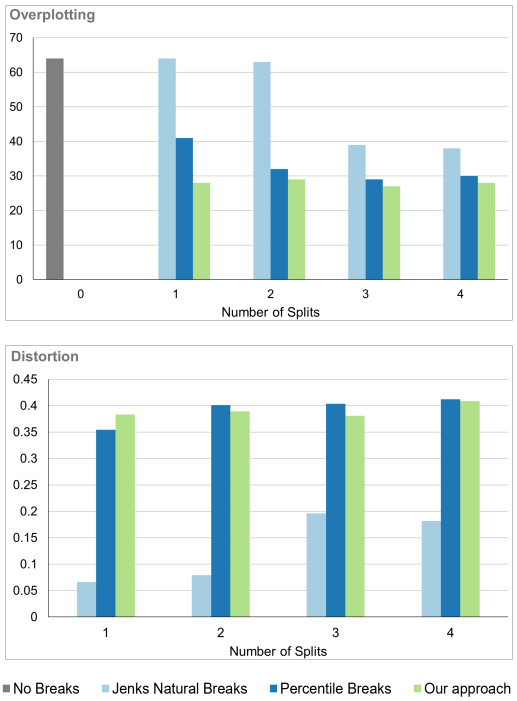

Figure 8: *Benchmark results for Parallel Coordinates with different axis break strategies. (top) Overplotting scores showing how much different approaches and different numbers of breaks reduce overplotting and thus increase the readability and traceability of the data points. (bottom) Distortion scores showing how much different approaches and different numbers of breaks increase the inevitable distortion introduced by them.*

to Split Parallel Coordinates, as the breaks do not have to be recomputed. Interactions such as brushing lines along a part of an axis are likewise expected to carry over to Split Parallel Coordinates, as the brushing can easily span across axis breaks.

However, interactions that rely on the slopes of the lines, such as *angular brushing* [HLD02, SGMS21], are challenging on Split Parallel Coordinates. This is because the correlation denoted by the slope of lines is lost when axis breaks are introduced. Thus, for tasks dependent on the slope of the lines, we recommend using axis scaling methods instead of axis breaks. Our Split Parallel Coordinates have this limitation in common with other parallel coordinate variants that do not use continuous, linearly scaled axes that run strictly parallel to each other – like Weidele's *Conditional Parallel Coordinates* [Wei19], which also use disjoint axes, but for different dimensions altogether.

In addition, an early study (N=147) by Carvalho and McMillan [CM92] indicates that larger distortions can impede reading of charts with broken axes. While the study did not include Parallel Coordinates, they observed that different chart types (line charts vs. bar charts) have different effects on how axis breaks affect reading behavior. Thus, further studies are needed to find thresholds where acceptable distortions become unacceptable, so as to provide upper bounds to the degree of fragmentation ϕ used in the cost function.

6. Conclusion

We have introduced a five step framework for automatically introducing axis-breaks in charts. We have argued that the complexity of the problem warrants such a framework, and we have shown how several existing approaches fit within this framework. We have demonstrated the usefulness of this framework by benchmarking a configuration of the framework against a standard linear scaling.

We have also introduced a new algorithm, Dynamic Optimization by Skew, for automatically determining the number and position of axis breaks, and an algorithm to determine readable sub-axis boundaries. First benchmarks indicate a slightly better performance in terms of reducing overplotting and distortion than the existing approaches. The whole framework has been implemented in a proof-of-concept application where the different techniques for each step can be tried out on some selected datasets.

Acknowledgements

We gratefully acknowledge partial funding support of this work through Aarhus University Research Foundation (AUFF) project VADE – Visual Analytcis of Data Errors.

References

[AA01] Andrienko G., Andrienko N.: Exploring spatial data with dominant attribute map and parallel coordinates. *Computers, Environment and Urban Systems 25*, 1 (2001), 5–15. doi:10.1016/S0198-9715(00)00037-5. 1, 2, 3, 4, 7

[AAS01] Andrienko G., Andrienko N., Savinov A.: Choropleth maps: Classification revisited. In *Proceedings of the International Cartographic Conference* (2001). 2

[Cle84] Cleveland W. S.: Graphical methods for data presentation: Full scale breaks, dot charts, and multibased logging. *The American Statistician 38*, 4 (1984), 270–280. doi:10.1080/00031305.1984.10483224. 1, 2

[CM92] Carvalho C. R., McMillan M. D.: *Graphic Representation in Managerial Decision Making: The Effect of Scale Break on the Dependent Axis*. Master's thesis, Air Force Institute of Technology, Air University, 1992. URL: https://apps.dtic.mil/sti/pdfs/ADA258989.pdf. 7

[DK10] Dasgupta A., Kosara R.: Pargnostics: Screen-space metrics for parallel coordinates. *IEEE TVCG 16*, 6 (2010), 1017–1026. doi:10.1109/TVCG.2010.184. 7

[EBD05] Ellis G., Bertini E., Dix A.: The sampling lens: making sense of saturated visualisations. In *CHI'05 Extended Abstracts on Human Factors in Computing Systems* (2005), ACM, pp. 1351–1354. doi:10.1145/1056808.1056914. 6

[FCW21] Fisher J., Chang R., Wu E.: Automatic Y-axis rescaling in dynamic visualizations. In *Proceedings of the IEEE Visualization Conference (VIS'21)* (2021), IEEE, pp. 116–120. doi:10.1109/VIS49827.2021.9623319. 1

[Fie17] Field J. A.: Some advantages of the logarithmic scale in statistical diagrams. *Journal of Political Economy 25*, 8 (1917), 805–841. doi:10.1086/253026. 3

[Fis58] Fisher W. D.: On grouping for maximum homogeneity. *Journal of the American Statistical Association 53*, 284 (1958), 789–798. doi:10.1080/01621459.1958.10501479. 2

[GLMN17] Grønlund A., Larsen K. G., Mathiasen A., Nielsen J. S.: Fast exact k-means, k-medians and Bregman divergence clustering in 1D. *CoRR abs/1701.07204* (2017). doi:10.48550/arXiv.1701.07204. 5

[HDS*10] Hao M., Dayal U., Sharma R., Keim D., Janetzko H.: Variable binned scatter plots. *Information Visualization 9*, 3 (2010), 194–203. doi:10.1057/ivs.2010.4. 1, 2

[HLD02] Hauser H., Ledermann F., Doleisch H.: Angular brushing of extended parallel coordinates. In *Proc. of the IEEE Symposium on Information Visualization (InfoVis'02)* (2002), IEEE, pp. 127–130. doi:10.1109/INFVIS.2002.1173157. 7

[HW13] Heinrich J., Weiskopf D.: State of the art of parallel coordinates. In *Eurographics 2013 – State of the Art Reports* (2013), Sbert M., Szirmay-Kalos L., (Eds.), Eurographics, pp. 95–116. doi:10.2312/conf/EG2013/stars/095-116. 6

[IBDF11] Isenberg P., Bezerianos A., Dragicevic P., Fekete J.-D.: A study on dual-scale data charts. *IEEE TVCG 17*, 12 (2011), 2469–2478. doi:10.1109/TVCG.2011.160. 1, 2, 3

[Ins09] Inselberg A.: *Parallel Coordinates: Visual Multidimensional Geometry and Its Applications*. Springer, 2009. doi:10.1007/978-0-387-68628-8. 6

[Jen77] Jenks G. F.: *Optimal data classification for choropleth maps. Occasional Paper No.2*. Tech. rep., University of Kansas, Department of Geography, Lawrence, KS, 1977. 2

[LHZ16] Lu L. F., Huang M. L., Zhang J.: Two axes re-ordering methods in parallel coordinates plots. *Journal of Visual Languages & Computing 33* (2016), 3–12. doi:10.1016/j.jvlc.2015.12.001. 1

[MESW21] MacTavish M., Etemad K., Samavati F., Willett W.: Perspective charts. In *Proceedings of the Graphics Interface 2021* (2021), Canadian Information Processing Society, pp. 246–255. doi:10.20380/GI2021.28. 2

[Num22] Numbeo: Dataset: Quality of life and cost of living. retrieved 01-JUN-2023. URL: https://www.numbeo.com. 7

[Raj18] Rajkumar S.: Dataset: UN country data. retrieved 01-JUN-2023. URL: https://www.kaggle.com/datasets/sudalairajkumar/undata-country-profiles. 7

[RSL*19] Richer G., Sansen J., Lalanne F., Auber D., Bourqui R.: HiePaCo: Scalable hierarchical exploration in abstract parallel coordinates under budget constraints. *Big Data Research 17* (2019), 1–17. doi:10.1016/j.bdr.2019.07.001. 6

[SGMS21] Sahann R., Gajic I., Moeller T., Schmidt J.: Selective angular brushing of parallel coordinate plots. In *Short Paper Proc. of EuroVis 2021* (2021), Eurographics, pp. 109–113. doi:10.2312/evs.20211064. 7

[TLH11] Talbot J., Lin S., Hanrahan P.: An extension of Wilkinson's algorithm for positioning tick labels on axes. *IEEE TVCG 16*, 6 (2011), 1036–1043. doi:10.1109/TVCG.2010.130. 1, 3, 5, 6

[US09] Unger A., Schumann H.: Visual support for the understanding of simulation processes. In *IEEE Pacific Visualization Symposium* (2009), IEEE, pp. 57–64. doi:10.1109/PACIFICVIS.2009.4906838. 2

[Wea22] Weaver S.: Dataset: Periodic table of chemical elements. retrieved 01-JUN-2023. URL: https://github.com/sweaver2112/periodic-table-data. 1, 7

[Wei19] Weidele D. K. I.: Conditional parallel coordinates. In *Short Paper Proc. of the IEEE Visualization Conference (VIS'19)* (2019), IEEE, pp. 221–225. doi:10.1109/VISUAL.2019.8933632. 7

[WF02] Wu L., Faloutsos C.: Making every bit count: Fast nonlinear axis scaling. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2002), ACM, pp. 664–669. doi:10.1145/775047.775146. 1

[WS11] Wang H., Song M.: Ckmeans.1d.dp: Optimal k-means clustering in one dimension by dynamic programming. *The R Journal 3* (12 2011), 29–33. doi:10.32614/RJ-2011-015. 5

[Wu91] Wu X.: Optimal quantization by matrix searching. *Journal of Algorithms 12*, 4 (1991), 663–673. doi:10.1016/0196-6774(91)90039-2. 5