Aesthetics and Ordering in Stacked Area Charts

Steffen Strunge Mathiesen[0000−0002−0499−9566] and Hans-Jörg Schulz[0000−0001−9974−535X]

Aarhus University, Dept. of Computer Science, Åbogade 34, 8200 Aarhus, Denmark
`st.st.ma@hotmail.com, hjschulz@cs.au.dk`

Abstract. Stacked area charts are a common visualisation type for sets of time series. Yet, they are also known to be challenging to read, in particular if the time series exhibit much fluctuation or even abrupt changes. In this paper, we introduce a novel approach to improving the layout of stacked area charts by means of reordering the time series in the stack. This approach breaks down into two parts: First, we gather aesthetic criteria and define associated quality metrics for stacked area charts. Second, we use these quality metrics together with a new algorithm called `UpwardsOpt` to find orderings of the stacked time series that optimise a chart's aesthetic properties. The produced orderings guarantee optimality in the sense that no better result can be obtained by moving any individual time series to a different position in the stack. In a benchmark study, we show that our algorithm can increase the layout quality up to 25%-50% over the state-of-the-art approach at the expense of longer runtimes. All datasets and an open source implementation of our algorithm are provided to facilitate their reuse and the reproducibility of our results.

Keywords: Time series visualisation, stacked graphs, layout algorithm

1 Introduction

Stacked area charts are a common means to show the aggregate of numerical quantities over time and available in any serious spreadsheet or dashboard application. Yet academic literature on them is sparse. In most cases, stacked area charts are merely discussed as a by-product of their bigger brother, the stream graph. This paper aims to change this by providing a broader view on the aesthetic criteria of stacked area charts and how to algorithmically fulfil them.

Stacked area charts (sometimes also called *layer charts*, *strata charts*, or *band curves* [10]) show a set of n individual time series $f_0 \dots f_{n-1}$ as horizontal layers that are piled on top of each other. As a result, the top contour of this stack of layers – the so-called *silhouette* g_n – conveys the sum of all time series: $g_n = \sum_{i=0}^{n-1} f_i$. In this stacked area charts differ from stream graphs, which do not stack on top of the horizontal time axis, but instead place layers above and below a curved baseline. Both chart types should only be used if the sum of the time series is a meaningful quantity and the silhouette carries useful information – e.g., for counts, percentages, or fractions [16, p.27].

The expressiveness and effectiveness of stacked area charts are debated in the literature. Prominent studies have looked mainly into their usefulness for various analytic tasks as compared to other chart types [6, 8, 12, 13]. One important observation is that any task performed with stacked area charts will be hindered by the accumulation of fluctuations across the layers – i.e., fluctuations of the lower layers affecting the look of the layers placed above them. In terms of expressiveness, this lets rather stable layers potentially look as if they are much more volatile than they actually are. In terms of effectiveness, this makes it very hard to read off concrete values for any time series but f_0 at the bottom of the stack and the overall silhouette g_n.

These apparent shortcomings stand in contrast to the unbroken popularity of stacked area charts for data journalism and reporting, with recent examples ranging from Donald Trump's tax returns[1] to the infection numbers of respiratory diseases including the COVID-19 virus[2]. These charts are made possible, as a clever ordering of the layers can reduce the readability problem of stacked area chart to some degree. This is done by finding an ordering of layers that minimises fluctuations, so that it eases the estimation of their thickness and does not convey a volatility that is not supported by the underlying data.

To achieve such an ordering that flattens the layers for better readability, existing algorithms focus on a quality metric called "wiggle" [3]. Generally speaking, wiggle is a measure of the fluctuations of a layer – i.e., its ups and downs. Different approaches for how to quantify a layer's wiggle have been proposed. They all have in common that they are based on the first derivative of a layer. In combination with an optimisation algorithm that aims to order the layers, so that the sum of their wiggle becomes minimal, the literature offers already good solutions to find suitable layer orderings for stacked area charts.

The first contribution of this paper is to show that there is more to the layout of stacked area charts than wiggle. We capture a set of aesthetic criteria for stacked area charts including means to quantify and combine them in one objective function: flatness, straightness, continuity, and significance. In their combination, these criteria yield a more balanced visual appearance for stacked area charts than it can be achieved by the mere optimisation of a chart's wiggle.

The second contribution of this paper is to give a novel optimisation algorithm with a stricter optimality guarantee than the state-of-the-art approach, which in turn yields better orderings for most charts. Where the state-of-the-art approach guarantees that no individual layer can be *swapped with a neighbouring layer* to improve the layout [5], our algorithm guarantees that no individual layer can be *moved to any other position* to improve the layout. This enables us to generate layouts that cannot be produced with the state-of-the-art algorithm – for example, where moving a layer to a better position would involve a

[1] `https://www.nytimes.com/interactive/2020/09/27/us/donald-trump-taxes-timeline.html`

[2] `https://publichealthinsider.com/2020/09/10/alongside-the-ongoing-transmission-of-covid-19-common-colds-are-on-the-rise-in-seattle-and-king-county/`

number of intermediate swaps that do not improve the overall layout. Together with other supplementary material, a Python implementation of our algorithm is available at `https://vis-au.github.io/stackedcharts/`.

2 Related work

Stacked area charts have been around for a long time, making it impossible to pinpoint their concrete origin or inventor. Already Willard Briton mentions them in his 1914 book "Graphic Methods for Presenting Facts" under the heading *component parts shown by curves* [2, ch.8]. In 1938, the American Standards Association approved design guidelines for stacked area charts [1, pp.64–65]. These guidelines were prepared by the Committee on Standards for Graphic Presentation of the American Society of Mechanical Engineers and they included advice on layout, gridding, scales, shading, and labelling. Some of these aspects still receive attention these days, as it is underlined by a novel labelling algorithm proposed in 2012 that seeks unused space to label layers [14]. Interestingly, the guidelines contain a list of circumstances under which stacked area charts should not be used:

– if accurate reading of values is desired, in the case of more than one layer,
– if irregular layers will unduly distort the contours of the others above it, or
– if changes in the series are abrupt, causing distortion of the layers' width.

In particular the latter two restrictions have ever since tickled the imagination of visualization designers, and a series of alternatives and improvements have been suggested to work around them. This resulted in an entirely new understanding of when and how stacked area charts can be used, as it is illustrated in Fig. 1. The most prominent of these layout alternatives is certainly the *ThemeRiver*™ technique [7] or its modern instance: the *stream graph* [3]. Though using different underlying algorithms, both yield very similar and appealing,

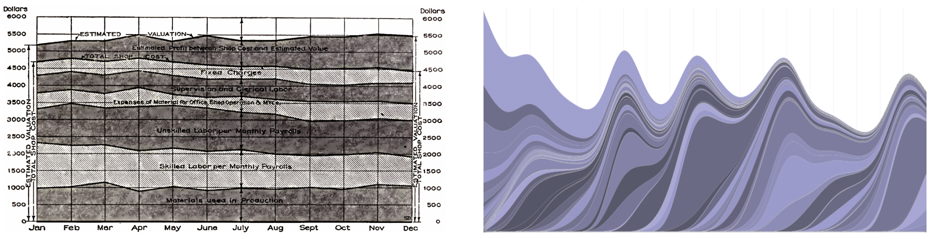


Fig. 1. The classic vs. the modern use of stacked area charts: On the left an example from 1914 that follows the then existing understanding that stacked area charts should only be used for mostly regular layers without any abrupt changes [2, p.147]. On the right an example from 2016 that shows the much more relaxed understanding prevalent today that, given the proper aesthetic optimisations, stacked area charts can also be used for very irregular bands that do not even persist across the entire chart [13].

flow-like stacked diagrams for irregular layers. Using the terminology from the stream graphs algorithm, these techniques reduce the distortion introduced by irregular layers by means of a curved baseline instead of a flat x-axis. This curve allows for a better distribution of the irregularities by vertically stacking layers not only above it, but also below it.

An important consideration for stream graphs has always been the layout of grouped (e.g., clustered) layers [15, 17, 4]. In particular the idea of accommodating the display of "complementary evolution" of multiple layers has received some attention in the literature. Complementarity means in this case that the fluctuations of two or more layers cancel each other out at least in some part – i.e., when one layer gets thicker, another get thinner and vice versa. It is assumed that complementary layers are somehow related to each other – i.e., forming a group or cluster – and thus one wants to place them side-by-side in the chart. One way to address this layout challenge is to add a term capturing complementarity to the evaluation function of the chart's quality, so that it will be taken into account by the ordering algorithm [11].

The work most related to ours is the research on the ordering of layers to reduce distortions due to irregularities. Byron and Wattenberg [3] were the first to point out the ordering problem and to address it by introducing the notion of wiggle. They proposed to order the layers by a heuristic called `OnSet` – i.e., the first occurrence of a non-zero value for a time series f_i determined the vertical position of its layer in the stack. Bartolomeo and Hu [5] later argued that not all datasets exhibit this property of time series appearing at different time points, which limits the usefulness of `OnSet` for other data. They propose two new heuristics: `BestFirst` for finding an initial ordering and `TwoOpt` for iteratively improving that ordering. `BestFirst` uses a greedy approach, where the next layer to be added to a stack is the one which adds the least wiggle to the chart. Its output is then improved by `TwoOpt`, which iteratively swaps adjacent layers if this will improve the wiggle. Their algorithm is the current state of the art.

While both `OnSet` and the combination `BestFirst`+`TwoOpt` have been proposed for stream graphs, their principal approach as well as the notion of wiggle are likewise applicable to "plain-old" stacked area charts. So, even though we are not focusing on stream graphs, but on stacked area charts, our contributions detailed in the following sections build very much on top of these two approaches.

3 Aesthetic criteria for stacked area charts

It is not easy to pinpoint what makes a stacked area chart look aesthetically pleasing and why. So far, having minimal wiggle is usually considered as the only criterion in this regard. Yet in our layout experiments with different datasets,[3] we encountered a number of curious glitches and imperfections in the resulting charts that apparently are not captured (well) by wiggle alone. These observations have sparked our investigation into a broader set of aesthetic criteria to

[3] See `https://vis-au.github.io/stackedcharts/` for a list of datasets used in this work.

create more balanced charts that trade some of that reduced wiggle for being a little less disagreeable in some other regards. In the following, we describe the four aesthetic criteria we found most useful in optimising the layout of stacked area charts.

As a layer can be geometrically captured in a variety of ways – e.g., through its centre line [3] or through its top and bottom outlines [5] – we introduce the quality metrics associated with the aesthetic criteria for a line l_i that is representative of layer f_i. Different options for choosing representative lines are discussed at the end of this section.

3.1 Flatness: Minimising wiggle

The idea behind *flatness* is that a layer looks nicer and is easier to judge in terms of its vertical span if it is as horizontal as possible. The latter is rooted in the *line width illusion* [9], which is a bias that leads us to perceive sloped layers as thinner than they actually are. The smaller the absolute slope of a layer is, the more this perceptual bias will be reduced. Minimising the slope is exactly what Byron and Wattenberg [3] proposed with their wiggle metric: the less wiggle a layer exhibits, the flatter it is. Hence, we can capture flatness in the same way by measuring the absolute slope between time points t_{j-1} and t_j for a layer's representative line l_i:

$$wiggle_{i,j} = |l'_i(t_j)| \cdot (t_j - t_{j-1}) \quad \text{with} \quad l'_i(t_j) = \frac{l_i(t_j) - l_i(t_{j-1})}{t_j - t_{j-1}} \tag{1}$$

The effect of optimising a whole chart for flatness is illustrated in Fig. 2b, which reorders the layers from the chart in Fig. 2a to minimise the wiggle of all shown layers, flattening them out as best as possible. As one can see, the overall course of the layers is more horizontal with a less ragged and more orderly look to them as compared to the random order in Fig. 2a. This does not only help in tracing them from left to right, but also in estimating and comparing their respective vertical spans, which we call the *thickness* of a layer.

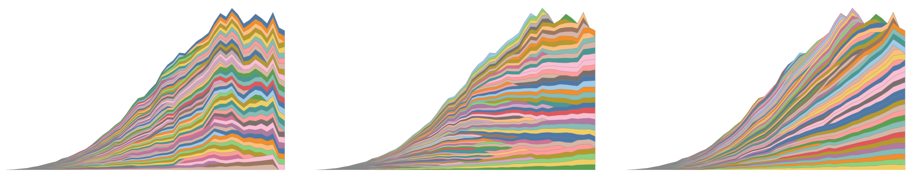
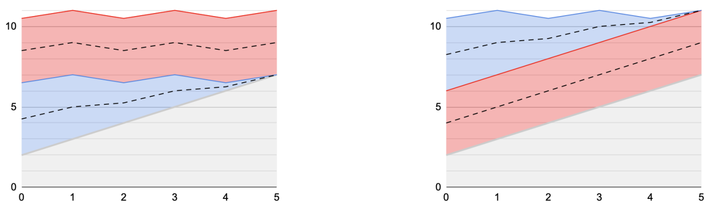

(a) Chart without any optimisation (random order) (b) Same chart as 2a, but optimised for flatness (c) Same chart as 2a, but optimised for straightness

Fig. 2. The visual effect of ordering the layers for flatness or straightness. Note that the colouring of layers is consistent across all three examples.



(a) $wiggle = 16.19$, $bump = 20.5$ (b) $wiggle = 26.19$, $bump = 4.5$

Fig. 3. Ordering the coloured layers for flatness (a) or straightness (b).

3.2 Straightness: Minimising bumps

Merely optimising for flatness can create charts that are made to be as horizontal as possible, no matter what. Yet in particular for charts that exhibit a large overall increase or decrease, that means for the optimisation to work against that overall trend of the chart. It does so by ordering the layers in a way that they meander between up and down to keep them as horizontal as possible. This results in a bumpy or wavy look for the individual layers that also shows in Fig. 2b. This bumpy look is an artefact of ordering the layers for flatness (i.e., minimal wiggle) and not inherent in the data.

To counter this visual effect, we introduce *straightness*, which is a criterion that aims to reduce those bumps and to keep the layers as steady and even as possible. This is schematically exemplified in Fig. 3b, where the order of the red and blue layers from Fig. 3a is reversed to maximise their straightness. As a result, the horizontal but bumpy look achieved by ordering for flatness is traded for a sloped but straight look – i.e., fewer, smaller bumps. The bumpiness of a layer can be captured using its second derivative as measure for how concave/convex it is. For a time point t_j and a representative line l_i, the second derivative can be computed using the first derivative from Eqn. 1:

$$bump_{i,j} = |l_i''(t_j)| \quad \text{with} \quad l_i''(t_j) = l_i'(t_{j+1}) - l_i'(t_j) \tag{2}$$

Fig. 2c shows the effects of maximising the straightness for an entire chart – the same chart that was optimised for flatness in Fig. 2b. By disregarding their flatness, the layers of this ordering look much calmer and steadier. They also better support the overall increasing trend of the silhouette, which remains somewhat of a baffling sight in Fig. 2b with its mostly horizontal layers.

3.3 Continuity: Minimising broken layers

In particular when layers appear or disappear suddenly, large singular jumps can sometimes occur in the chart which may “break” layers – i.e., have them end at time point t_j at a y-coordinate that is very different from the y-coordinate at which they continue at time point t_{j+1}. This is illustrated in Fig. 4, showing how this problem depends very much on the aspect ratio of the chart (left chart vs.

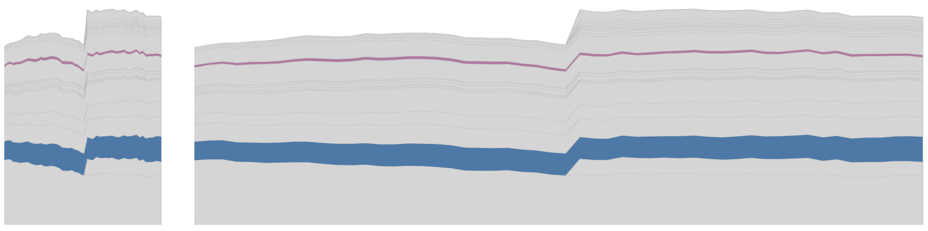


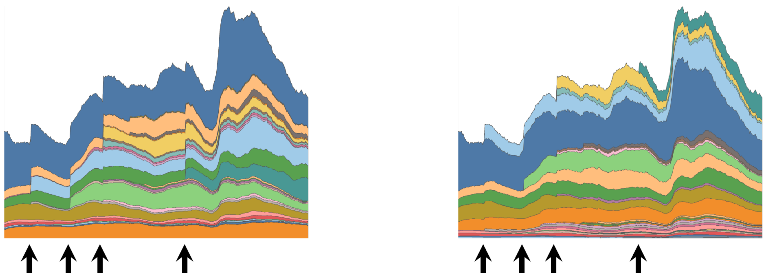
Fig. 4. Illustration of the effect of layer thickness and aspect ratio on the appearance of broken layers caused by sudden vertical shifts. Data: subset of *unempl*

right chart) and on the width of the layer (red layer vs. blue layer). This is not only a cosmetic problem, but also a problem for a chart's effectiveness: breaks can make it tricky if not impossible to follow a layer – especially if there are many thin layers and the layers are shifted by many multiples of their thickness.

Hence, we introduce *continuity* as an additional aesthetic criterion. We capture continuity by measuring the vertical displacement of a layer in relation to its average thickness. We call this measure *break* and we define it for a time point t_j, a layer f_i, and a representative line l_i as:

$$break_{i,j} = \left| \frac{l'_i(t_j)}{f_i(t_j) + f_i(t_{j-1})} \right|^2 \cdot (t_j - t_{j-1}) \tag{3}$$

Using continuity by itself to generate an optimised layer ordering does not yield aesthetic results – it just removes the breaks. It is rather useful as an "add-on" to ordering by flatness or straightness to prevent breaks from appearing as optimal solutions when optimising for these criteria. This can be seen in Fig. 5, where we optimise solely for flatness in Fig. 5a, which introduces four breaks in the chart. Fig. 5b shows the same chart, but this time also optimised to minimise breaks. It has less breaks, but they are not entirely gone – for example, the light blue layer still exhibits a break at the position of the second arrow from the left.



(a) Ordering solely for flatness. (b) Ordering for flatness and continuity.

Fig. 5. The effect of introducing continuity as an additional criterion to reduce the sudden vertical shifts visible at the indicated time points. Data: *unempl*

This could be fixed by moving the green layer starting at that position atop the light blue layer. Yet, this would apparently introduce so much additional *wiggle* for the green layer, that the algorithm took this as the better trade-off.

3.4 Significance: Minimising the influence of thin layers

Treating all layers equally results in layouts that are close to optimal in a mathematical sense, but not by their visual appearance. The reason is that thin layers are optimised for their aesthetics in the same way as thick layers, even though thicker layers are visually much more salient. Sacrificing flatness or straightness of a prominently visible thick layer to improve the aesthetics of a barely visible thin layer is not a good visual trade-off. Whereas, getting the layout right for those thick layers contributes greatly towards an overall aesthetic chart and can make up for a number of thin layers which are not placed quite as optimally.

We propose the criterion of *significance*, which prioritises thicker layers in the layout process – or rather: thicker parts of the layers as thickness may vary over time and thus a layer maybe more important to the layout during one time interval than during another. Additionally, parts of the layers with zero thickness (i.e., a layer disappearing from the chart for some time) should not be counted at all at those time points, since they do not impact the aesthetics of the chart while they are not visible. To capture this, we weight each layer's aesthetic scores by its thickness. Already Byron and Wattenberg used a quadratic wiggle function to emphasise the importance of the thicker layers in the chart [3], and we extend this practice to all aesthetic measures in this section. Yet often a mere linear weighting does not give the thicker layers the prominence in the optimisation that they have in the visual appearance of a chart. Hence, we further introduce variable exponents to these weights to increase or decrease the effect of the weighting as needed. These exponents can be varied to adjust the weighting from chart to chart and from one aesthetic criterion to the next. This procedure yields the following three formulas for *wiggle*, *bump*, and *break* values as they are aggregated over all m time points for a layer f_i using line l_i and exponent s:

$$\begin{aligned}
wiggle_i &= \sum_{j=1}^{m-1} \left(\frac{f_i(t_j) + f_i(t_{j-1})}{2} \right)^s \cdot wiggle_{i,j} \\
bump_i &= \sum_{j=1}^{m-2} f_i(t_j)^s \cdot bump_{i,j} \\
break_i &= \sum_{j=1}^{m-1} \left(\frac{f_i(t_j) + f_i(t_{j-1})}{2} \right)^s \cdot break_{i,j}
\end{aligned} \tag{4}$$

Note that the weight for $bump_i$ is not an average, but depends on the thickness of a layer at the time point $f_i(t_j)$ where the bump is expressed. In other terms: $wiggle_i$ and $break_i$ use the first derivative defined between two time points t_{j-1} and t_j and we thus use the average thickness between these two time points.

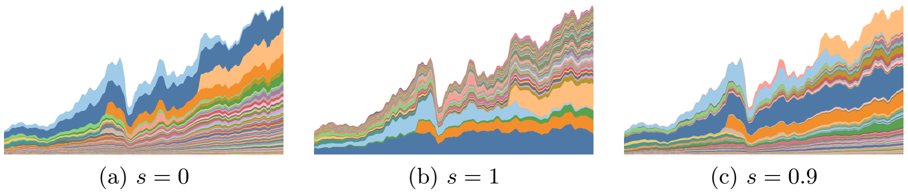


Fig. 6. Using different exponents for weighting the layout aesthetics – in this case straightness – by the layers' thickness. (a) employs no weighting and thus all thinner layers are placed at the bottom to "straighten out" as many layers as possible to reduce the overall *bump* measure. (b) employs linear weighting, which places all the thicker layers at the bottom as their straightness is prioritised. (c) uses an exponent in between 0 and 1 to get a compromise solution between (a) and (b). Data: *stocks*

But $bump_i$ uses the second derivative defined between three time points t_{j-1}, t_j, and t_{j+1} and we thus use the thickness $f_i(t_j)$ at t_j directly.

Using the exponent s, we can either remove the weighting by setting $s = 0$, use a linear weighting by setting $s = 1$, or increase s further to emphasise thicker layers even more. This can be seen in Fig. 6, where we optimise for straightness using different exponents for weighting the layers' thickness.

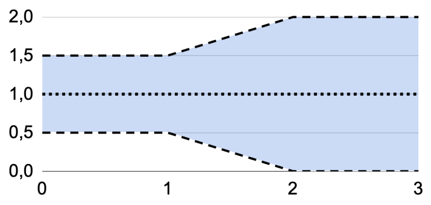
3.5 Choosing one or more representative lines

So far, Secs. 3.1- 3.4 have not specified for which concrete line l_i of a layer to compute the aesthetics. Byron and Wattenberg [3] advocate the use of the centre line c_i of a layer – i.e., the line halfway between the bottom and the top of a layer. This line c_i can easily be determined by averaging bottom and top y-values of a layer at each time point t_j. Another option given by Bartolomeo and Hu [5] is to compute the aesthetics separately for the bottom line g_i and for the top line g_{i+1} (the top line of f_i being the bottom line of f_{i+1}) and then to average the resulting values to yield a single aesthetic quantity per layer. The three lines $l_i \in \{g_i, g_{i+1}, c_i\}$ of a layer f_i can be computed in a straightforward manner:

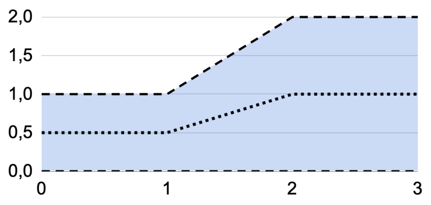
$$g_i = \sum_{k=0}^{i-1} f_k \qquad g_{i+1} = g_i + f_i \qquad c_i = \frac{g_i + g_{i+1}}{2} \tag{5}$$

The main argument that speaks for using the outer lines g_i and g_{i+1} is that they are actually visible to the viewer. I.e., while the centre line is an "imaginary" line running in the middle of each layer, the outer lines are the borders to the respective next layers and thus shown in the chart. Hence, they are the ones that potentially produce a layer's wiggly or bumpy look, so it makes sense to optimise the layout with respect to them.

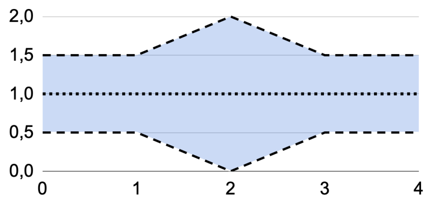
The main argument that speaks for using the centre line is its sensitivity to certain changes in a layer's thickness. This is illustrated in Fig. 7, where the centre line is able to capture the difference between the examples shown on the left and on the right. For example, in Fig. 7a, the *wiggle* values (i.e.,



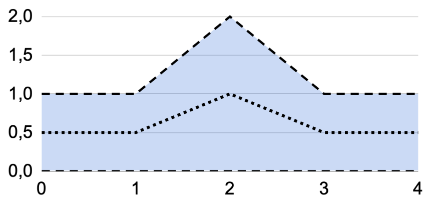
(a) $wiggle_c(t_2) = 0.0$, $wiggle_o(t_2) = 0.5$



(b) $wiggle_c(t_2) = 0.5$, $wiggle_o(t_2) = 0.5$



(c) $bump_c(t_2) = 0$, $bump_o(t_2) = 1$



(d) $bump_c(t_2) = 1$, $bump_o(t_2) = 1$

Fig. 7. Computing *wiggle* and *bump* values for centre line (dotted) and outer lines (dashed) of a layer. With the centre line, one can distinguish between left and right.

absolute slopes) between t_1 and t_2 are 0.5 for both top and bottom line, so their average *wiggle* is 0.5. In Fig. 7b, outer lines again yield a *wiggle* value of 0.5 (the average of 0 for the bottom line and 1 for the top line). So, measured by their outer lines, left and right side would be considered equally good, even though the left example distributes the increase in thickness in a much nicer and more even way. Though, the *wiggle* values of the centre line reflect this nicer look of the left variant through a smaller value (0 on the left vs. 0.5 on the right). The same is true in Figs. 7c and 7d, which show a similar example for straightness and its associated *bump* values.

Hence, none of the two options is clearly superior to the other. Therefore, we propose a convex combination of all three lines with the weights α, β, and γ summing up to 1 for an aesthetic criterion $crit_i \in \{wiggle_i, bump_i, break_i\}$:

$$crit_i = \alpha \cdot crit_i(g_i) + \beta \cdot crit_i(c_i) + \gamma \cdot crit_i(g_{i+1}) \tag{6}$$

Setting $\alpha = 0, \beta = 1, \gamma = 0$ yields Byron and Wattenberg's procedure, while setting $\alpha = 0.5, \beta = 0, \gamma = 0.5$ yields Bartolomeo and Hu's procedure. Any other setting can be used to go beyond these two approaches, even combining all three lines with each other if desired. In the latter case, it has to be noted that the outer line at the top of one layer is at the same time the outer line at the bottom of the next layer and will thus be taken into account twice. Hence, weighing $\alpha = \beta = \gamma$ does not actually put all lines on par with each other, but will instead emphasise the outer lines over the centre line. Another more intricate option follows from the observations in Fig. 7 and would dynamically shift the weights so that when a layer's thickness changes the middle line gets higher weights, and when it is steady the outer lines get weighted more.

4 Ordering layers

Being able to quantify a chart's aesthetics does not automatically yield a better chart. To that end, we need to combine the different criteria into an objective function and then search the space of all possible layer orderings for one that minimises that function.

4.1 Objective function

Integrating the individual aesthetic criteria into one objective function is not quite straightforward due to the very different value ranges their respective quality measures produce. We chose an approach that multiplies the individual metrics with each other. To weight their influence on the overall outcome, we introduce the following weight function:

$$w(x, w_{metric}) = 1 - w_{metric} + x \cdot w_{metric} \tag{7}$$

This function takes a value $x \in \mathbb{R}$ and a weight $w_{metric} \in [0 \ldots 1]$ with $metric \in \{wiggle, bump, break\}$. It produces a weighted output of x, returning 1 if $w_{metric} = 0$ and x if $w_{metric} = 1$, so it will have no effect on the multiplication in the objective function if set to 0 and full effect if set to 1.

Given that we aim to solve a minimisation problem, we formulate the objective function as a cost function, so as to reduce the values of *wiggle*, *bump*, and *break*. Its output depends on i, which specifies the layer and thus the time series f_i for which it shall be computed, as well as on its bottom line g_i produced by the underlying stack of layers on which layer i is placed and whose fluctuations add onto those of layer i itself:

$$\begin{aligned} &cost_{layer}(i, g_i) = \\ &\quad w\left(\frac{wiggle_i}{\sum chart}, w_{wiggle}\right) \cdot w\left(\frac{bump_i}{\sum chart}, w_{bump}\right) \cdot w\left(\frac{break_i}{\sum chart}, w_{break}\right) \end{aligned} \tag{8}$$

The normalisation is done with $\sum chart = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} f_i(t_j)$. Without loss of generality, all other parameters (e.g., the significance exponents s or the weights α, β, γ for the convex combination of lines) are assumed as globally defined to keep the equations as well as the following algorithms readable. For all practical purposes, a reasonable default parameterisation is to set s, β, w_{wiggle}, w_{bump} to 1 and α, γ, w_{break} to 0. The resulting chart can then be used as a starting point to fine-tune individual parameters, like increasing the weight w_{break} if indeed breaks occur and need to be taken care of.

The cost for the whole chart is derived by summing its layers' costs:

$$cost_{chart}(stack) = \sum_{i=0}^{n-1} cost_{layer}(i, g_i) \tag{9}$$

Both variants, the cost per layer and the cost of the whole chart, are being used in the following when describing our optimisation algorithm.

4.2 Optimisation procedure

Our ordering algorithm – called `UpwardsOpt` – improves the ordering of an initial stack of layers w.r.t. a given cost function $cost_{chart}$. `UpwardsOpt` iterates over all layers starting from the bottom and stopping at the top, hence its name. At each iteration i, `UpwardsOpt` (1) removes layer i from the stack, (2) calls a function $FindBestPosition$ to determine the best position of layer i in the remaining stack, and (3) reinserts layer i there. After having gone through all layers, `UpwardsOpt` compares the cost values of the initial stack and of the resulting stack. If the cost improved by more than a given threshold, `UpwardsOpt` is run again on the resulting stack. Otherwise the algorithm terminates.

As for a suitable initial ordering, we recommend sorting the layers by their average thickness starting with the thickest layer at the bottom of the stack and then decreasing towards the top. Beginning with our incremental ordering algorithm from this sorted initial order can speed-up its runtime drastically, cutting down on the number of necessary executions of `UpwardsOpt` before passing the termination threshold. This is due to the fact that this particular ordering ensures an early consideration of the thick layers, whose impact on the overall layout is usually substantial. Once the thicker layers are moved to suitable positions, all the thinner layers can then fall in place around them. If the thicker layers were considered at a later point, many of the already well-positioned thinner layers would need to be repositioned again, requiring more executions of `UpwardsOpt` to converge on the final order. In our experiments, such an initial ordering by average thickness combined with a threshold of a minimum improvement of 1% resulted in 2 to 4 executions of `UpwardsOpt` before termination.

A naïve implementation of the algorithm described above would result in a cubic runtime complexity: For each execution of `UpwardsOpt`, we iterate over n layers, testing for each layer n different positions in the stack, and for each tested position we compute $cost_{chart}$ by summing the $cost_{layer}$ function for all n layers in the stack. Yet our implementation of the $FindBestPosition$ function (shown as Algorithm 1) brings the computation down to quadratic complexity by eliminating the need to recompute $cost_{chart}$ each time a new position is tested for a layer. This is done by preprocessing and storing three types of costs:

- $costBelow$ (line 5) holds the layer cost under the assumption that the layer in question is below the layer i that is to be positioned – i.e., layer i does not add into the bottom line $gBelow$ of that layer.
- $costAbove$ (line 6) holds the layer cost under the assumption that the layer in question is above the layer i that is to be positioned – i.e., layer i adds into the bottom line $gAbove$ of that layer.
- $costLayer$ (line 7) holds the cost of layer i if being moved to position pos. This means at index pos, $costLayer[pos]$ contains the cost of layer i sitting on the stack of layers 0 through $pos - 1$. The cost of layer i being positioned all the way at the top is added in line 11.

During testing, we then use these costs to determine the chart's overall cost if layer i is moved to position pos in the stack. This is done by combining the

Algorithm 1 FindBestPosition

```
 1: procedure FindBestPosition($order, f_i$)
 2:     $gBelow \leftarrow 0; gAbove \leftarrow f_i$                         ▷ Preprocessing Stage
 3:     $costBelow, costAbove, costLayer \leftarrow []$
 4:     for $pos = 0$ to $length(order) - 1$ do
 5:         $costBelow.add(cost_{layer}(order[pos], gBelow))$
 6:         $costAbove.add(cost_{layer}(order[pos], gAbove))$
 7:         $costLayer.add(cost_{layer}(f_i, gBelow))$
 8:         $gBelow \leftarrow gBelow + order[pos]$
 9:         $gAbove \leftarrow gAbove + order[pos]$
10:     end for
11:     $costLayer.add(cost_{layer}(f_i, gBelow))$
12:
13:     $currentCost \leftarrow costLayer[0] + \sum_{l=0}^{j-2} costAbove[l]$        ▷ Testing Stage
14:     $bestIndex \leftarrow 0, bestCost \leftarrow currentCost$
15:     for $pos = 1$ to $length(order) - 1$ do
16:         $currentCost$ += $costBelow[pos-1] - costAbove[pos-1]$
17:         $currentCost$ += $costLayer[pos] - costLayer[pos-1]$
18:         if $currentCost < bestCost$ then
19:             $bestIndex \leftarrow pos$, $bestCost \leftarrow currentCost$
20:         end if
21:     end for
22:     return $bestIndex$
23: end procedure
```

overall cost from *costBelow* for all layers below *pos*, from *costLayer* for the costs of layer i being placed at *pos*, and from *costAbove* for all layers above *pos*:

$$cost_{chart}(g_0) = \sum_{l=0}^{pos-1} costBelow[l] + costLayer[pos] + \sum_{l=pos}^{n-2} costAbove[l] \quad (10)$$

While this saves us from re-computing the cost function for all layers each time we try a new position for layer i, we can even get rid of the summations in Eqn. 10. This is done by testing new positions for layer i from bottom to top, moving it up one position at a time. When moving layer i up to a new position *pos*, we do not need to recompute the cost for the whole stack, but only to adjust the cost value computed for the last position $pos - 1$ that we tested:

$$\begin{aligned} newCost = \; & currentCost \\ & + costBelow[pos-1] - costAbove[pos-1] \\ & + costLayer[pos] - costLayer[pos-1] \end{aligned} \quad (11)$$

This exact procedure can be found in Algorithm 1 in lines 16 and 17. This way, we only need to sum over the full stack once in the beginning for $pos = 0$ (line 16), i.e., testing the very bottom position for layer i. Afterwards, the above procedure can make use of the preprocessed cost values without having to run the cost function and without iterating over the full stack again.

5 Benchmarking

We implemented our ordering algorithm (`UpwardsOpt`) as well as the state-of-the-art algorithm (`BestFirst+TwoOpt`) as a Python 3.6 backends to a Tableau v.2019 chart. All benchmarks were run on a 2017 27" iMac 5K with a 3.4 GHz Intel Core i5 processor and 40 GB RAM. The datasets used in our benchmarks were chosen to span the different possibilities from only a few time series with many time points, all the way to many time series with only a few time points. They can be found at `https://vis-au.github.io/stackedcharts/`.

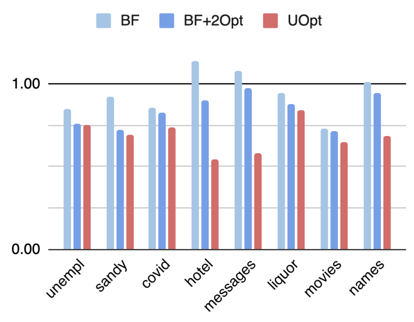
We restricted the benchmark to two cases: optimising only for flatness (minimising wiggle) and optimising only for straightness (minimising bumps). The significance exponent was set to $s = 1$. We used only outer lines – i.e., $\alpha = 0.5, \beta = 0.0, \gamma = 0.5$ – and a 1% threshold of minimum improvement. As a neutral reference point, we generated $100,000$ randomly ordered stacks for each dataset and averaged their $cost_{chart}$ values. We then computed the optimised orderings using `BestFirst` only, the combination of `BestFirst+TwoOpt`, as well as our algorithm `UpwardsOpt`. Their $cost_{chart}$ values were then set in relation to the averaged values to see how much each improves over the average random order. We further logged the runtimes of `BestFirst+TwoOpt` and of `UpwardsOpt`.

The results are documented in Table 1 and Fig. 8. In terms of quality and speed, we can observe that all trends persist for both, flatness and straightness. We can also observe that `UpwardsOpt` produces better, but slower outputs than `BestFirst+TwoOpt` throughout all datasets. The use of the `BestFirst` heuristic by itself produces very mixed results, from close to optimal orderings (e.g., for *movies*) to worse than the average random ordering (e.g., for *hotel*).

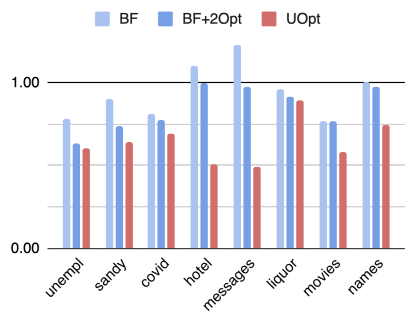
Quality-wise, `UpwardsOpt` performs only slightly better than `BestFirst+` texttt TwoOpt for datasets with only few layers (e.g., for *unempl* or *sandy*), as well as for rather similar layers that do not exhibit much individual traits (e.g., for *liquor*). In both cases, the search space is not that large for both algorithms to find much different solutions – either because there are only a few layers to reorder in the first place, or because no reordering would much affect the outcome. Yet, for some datasets that have a mix of longer and shorter layers (e.g., *messages*), `UpwardsOpt` improves significantly over `BestFirst+TwoOpt`.

Table 1. Results of our benchmarking. Lower values are better. Costs are relative: $cost = 1.00$ denotes the quality of an average random order derived from $100,000$ random trials, $cost = 0.00$ denotes perfect quality with no wiggle or bumps.

			relative costs, flatness			relative costs, straightness			times (secs), flatness		times (secs), straightness	
dataset	n	m	BF	BF+2Opt	UOpt	BF	BF+2Opt	UOpt	BF+2Opt	UOpt	BF+2Opt	UOpt
unempl	28	443	1.06	0.82	0.81	1.04	0.73	0.67	3.79	3.58	2.05	4.29
sandy	183	33	0.92	0.73	0.69	0.89	0.74	0.65	2.59	15.67	1.77	11.81
covid	206	113	0.86	0.83	0.74	0.81	0.77	0.65	4.51	59.36	4.71	69.75
hotel	334	115	1.13	0.90	0.59	1.10	1.00	0.54	16.47	214.02	12.85	196.42
messages	604	135	1.08	0.98	0.58	1.23	0.97	0.49	45.50	640.39	58.64	1002.37
liquor	695	240	0.95	0.88	0.84	0.96	0.92	0.89	167.38	1014.19	180.45	1264.28
movies	881	51	0.73	0.71	0.64	0.77	0.76	0.60	28.55	504.54	31.11	589.37
names	1000	135	1.01	0.94	0.69	1.00	0.98	0.74	165.47	2500.90	178.67	2024.34



(a) Flatness



(b) Straightness

Fig. 8. Relative layout costs from Table 1. Lower is better.

`BestFirst` will pick the shorter layers first, because they barely increase the overall cost. But in the end, only longer layers remain and will be placed on top of the shorter ones, creating a far from optimal starting point for `TwoOpt`.

Runtime-wise, we see that `UpwardsOpt` takes roughly about one order of magnitude more time to complete than `BestFirst`+`TwoOpt`. The only exception is the smallest dataset *unempl*, for which no significant differences could be observed. The measured runtimes increase mainly with the number of layers, but they are also dependent on the structure of the dataset itself. An example is the *movies* dataset with 881 layers, but its runtime is well below the *liquor* dataset with only 695 layers. This is due to the fact that the layers in the *movies* dataset only span rather short time intervals. As a result, reordering them disturbs only a small part of the chart, so that fewer iterations of `UpwardsOpt` are needed.

6 Conclusion

We have presented aesthetic criteria that allow for a flexible configuration of layout properties. We have further introduced a novel ordering algorithm that yields results of higher quality. Specifically, it guarantees that no better ordering can be obtained by moving any *individual layer* to another position in the chart. It has to be noted though, that our algorithm does not necessarily find the global optimum, as moving *multiple layers* to different positions at once might still yield an even better ordering. First benchmarks of our algorithm show that in ideal situations (i.e., layers with high fluctuations) our algorithm can increase the layout quality up to $25\% - 50\%$ over the state-of-the-art approach. This improvement comes at the cost of longer runtimes, which we deem acceptable for two reasons: First, stacked area charts are hardly ever used for thousands of time series, so that runtimes usually remain tolerable. Second, stacked area charts are usually generated for presentation purposes. It is hence sensible to spend some computation time to yield ready-to-print charts that look their best.

References

1. American Standards Association: Time Series Charts: A Manual of Design and Construction. ASME (1938), `https://hdl.handle.net/2027/wu.89083916932`
2. Brinton, W.C.: Graphic Methods for Presenting Facts. The Ronald Press Company (1914), `https://archive.org/details/cu31924032626792`
3. Byron, L., Wattenberg, M.: Stacked graphs–geometry & aesthetics. IEEE TVCG **14**(6), 1245–1252 (2008). https://doi.org/10.1109/TVCG.2008.166
4. Cuenca, E., Sallaberry, A., Wang, F.Y., Poncelet, P.: MultiStream: A multiresolution streamgraph approach to explore hierarchical time series. IEEE TVCG **24**(12), 3160–3173 (2018). https://doi.org/10.1109/TVCG.2018.2796591
5. Di Bartolomeo, M., Hu, Y.: There is more to Streamgraphs than movies: Better aesthetics via ordering and lassoing. Computer Graphics Forum **35**(3), 341–350 (2016). https://doi.org/10.1111/cgf.12910
6. Harrison, L., Yang, F., Franconeri, S., Chang, R.: Ranking visualizations of correlation using Weber's law. IEEE TVCG **20**(12), 1943–1952 (2014). https://doi.org/10.1109/TVCG.2014.2346979
7. Havre, S., Hetzler, E., Whitney, P., Nowell, L.: ThemeRiver: Visualizing thematic changes in large document collections. IEEE TVCG **8**(1), 9–20 (2002). https://doi.org/10.1109/2945.981848
8. Heuer, H., Polizzotto, A., Marx, F., Breiter, A.: Visualization needs in computational social sciences. In: Proc. of MuC'19. pp. 463–468. ACM (2019). https://doi.org/10.1145/3340764.3344440
9. Hofmann, H., Vendettuoli, M.: Common angle plots as perception-true visualizations of categorical associations. IEEE TVCG **19**(12), 2297–2305 (2013). https://doi.org/10.1109/TVCG.2013.140
10. Huggett, R.: Multiple Line Graphs (2), pp. 43–46. Palgrave Macmillan (1990). https://doi.org/10.1007/978-1-349-11245-6_10
11. Liu, S., Zhou, M.X., Pan, S., Song, Y., Qian, W., Cai, W., Lian, X.: TIARA: Interactive, topic-based visual text summarization and analysis. ACM TIST **3**(2), 25:1–28 (2012). https://doi.org/10.1145/2089094.2089101
12. Rodrigues, A.M.B., Barbosa, G.D.J., Lopes, H., Barbosa, S.D.J.: Comparing the effectiveness of visualizations of different data distributions. In: Proc. of SIBGRAPI'19. pp. 84–91. IEEE (2019). https://doi.org/10.1109/SIBGRAPI.2019.00020
13. Thudt, A., Walny, J., Perin, C., Rajabiyazdi, F., MacDonald, L., Vardeleon, D., Greenberg, S., Carpendale, S.: Assessing the readability of stacked graphs. In: Proc. of GI'16. pp. 167–174 (2016). https://doi.org/10.20380/GI2016.21
14. Toledo, A., Sookhanaphibarn, K., Thawonmas, R., Rinaldo, F.: Evolutionary computation for label layout on unused space of stacked graphs. International Scholarly Research Notices pp. 139603:1–10 (2012). https://doi.org/10.5402/2012/139603
15. Wang, Y., Wu, T., Chen, Z., Luo, Q., Qu, H.: STAC: Enhancing stacked graphs for time series analysis. In: Proc. of IEEE PacificVis'16. pp. 234–238. IEEE (2016). https://doi.org/10.1109/PACIFICVIS.2016.7465277
16. Wills, G.: Visualizing Time: Designing Graphical Representations for Statistical Data. Springer (2012). https://doi.org/10.1007/978-0-387-77907-2
17. Wu, T., Wu, Y., Shi, C., Qu, H., Cui, W.: PieceStack: Toward better understanding of stacked graphs. IEEE TVCG **22**(6), 1640–1651 (2016). https://doi.org/10.1109/TVCG.2016.2534518