Eurographics Conference on Visualization (EuroVis) 2016
K.-L. Ma, G. Santucci, and J. van Wijk
(Guest Editors)

Volume 35 (2016), Number 3

There is More to Streamgraphs than Movies: Better Aesthetics via Ordering and Lassoing

Marco Di Bartolomeo[†1] and Yifan Hu[‡2]

[1]Roma Tre University, Rome, Italy
[2]Yahoo Research, New York, NY, USA

Abstract

Streamgraphs were popularized in 2008 when The New York Times used them to visualize box office revenues for 7500 movies over 21 years. The aesthetics of a streamgraph is affected by three components: the ordering of the layers, the shape of the lowest curve of the drawing, known as the baseline, and the labels for the layers. As of today, the ordering and baseline computation algorithms proposed in the paper of Byron and Wattenberg are still considered the state of the art. However, their ordering algorithm exploits statistical properties of the movie revenue data that may not hold in other data. In addition, the baseline optimization is based on a definition of visual energy that in some cases results in considerable amount of visual distortion. We offer an ordering algorithm that works well regardless of the properties of the input data, and propose a 1-norm based definition of visual energy and the associated solution method that overcomes the limitation of the original baseline optimization procedure. Furthermore, we propose an efficient layer labeling algorithm that scales linearly to the data size in place of the brute-force algorithm adopted by Byron and Wattenberg. We demonstrate the advantage of our algorithms over existing techniques on a number of real world data sets.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

A time series is a sequence of numeric values each labeled with a temporal reference, and ordered by time. A standard way to visualize time series is to plot them on a cartesian chart that has time on the x-axis and the numeric values on the y-axis. This chart clearly shows how the series evolve over time. Plotting multiple time series in the same chart allows easy comparison between them. However it is not effective at showing the evolution of their sum. Stacked graphs, or stacked charts, are a representation that mitigates this limit. In a stacked graph, time series are shown as colored stripes (or *layers*) that flow in the direction of the x-axis, and whose thickness represents at each time instant the numeric value. Layers are stacked one on top of another without gaps. The result is a diagram in which it is easy to compare single layers with the total, at the expense of harder comparisons between pairs of layers. Given a stacked graph, the *baseline* is its bottommost curve. The simplest way to make a stacked graph is to stack layers on a straight line that corresponds to the x-axis. However different baselines may be used in order to reduce the amount of fluctuation in the layers.

First introduced by ThemeRiver [HHWN02], stacked graphs with curved baselines were popularized in 2008 by an article on The New York Times, which used them to visualize box office revenues for 7500 movies over 21 years. The visualization became immediately popular and controversial, gathering comments ranging from "fantastic" to "unsavory". Later on, a paper by Byron and Wattenberg [BW08] described the technique used in The New York Times, calling it *streamgraphs*. The article outlined how the aesthetics of a streamgraph is controlled by three components: the ordering of layers, the shape of the baseline, and the labeling of the layers. In addition, the authors explained how stacked graphs with a flat baseline, ThemeRiver-like graphs, and their streamgraphs all fitted into a general mathematical framework, and that each of these had different aesthetic properties. For this reason, in this paper we will refer to any kind of stacked graph as a streamgraph.

At the time of writing this article, the paper of Byron and Wattenberg [BW08] is still the most authoritative work on streamgraphs, and other works are mostly implementations or minor variations of the original concepts. The algorithmic pipeline of Byron and Wattenberg [BW08], and that of this paper, consists of three distinct steps: layers are first ordered; a baseline is then computed to minimize the fluctuation of layers; finally, labels are computed and superimposed on the streamgraph. However, in each of these steps,

† e-mail:dibartolomeo@ing.uniroma3.it
‡ e-mail:yifanhu@yahoo.com

the solutions in [BW08] have limitations that have not been documented in the literature. First, the layer ordering algorithm exploited statistical properties of the data of interest, box office revenues, that may not hold in other data. Second, the algorithm to compute a baseline is based on a 2-norm measurement of the fluctuation of layers, called *wiggle*, which works well for relatively smooth time series, but suffers from unpleasant distortions when a layer has sudden changes in thickness (or *jumps*, see Fig. 1). Third, in [BW08] a brute-force layer labeling algorithm was used that does not scale well with the size of the data.

We resolve each of these limitations by proposing better ordering, baseline calculation, and layer labeling algorithms, and verify the effectiveness of our solutions by performing experiments that compare our algorithms with the current state of the art. Our contributions are:

- an alternative definition of wiggle based on 1-norm that gives visually calmer layer arrangements even for time series with sudden jumps, and an algorithm that minimizes the wiggle (Section 3);
- a new ordering algorithm that is effective on general data (Section 4);
- an efficient layer labeling algorithm that scales linearly to the size of the time series data (Section 5).

Together, these represent a new algorithmic pipeline that significantly advances the state of the art in the creation of streamgraphs. The resulting JavaScript code will be made available as an open source extension of the popular D3 library [Bos]. Additional material on this submission, with high resolution figures produced by our algorithms, can be found online at [DH].

We note that defining wiggle by 1-norm avoids uniform small jumps across the layers in favour of a few larger jumps. This sparsifying effect is similar to that of using 1-norm for regularization in regression, known as lasso [Tib94], hence the name of this paper.

2. Related Work

The first work to introduce streamgraphs was ThemeRiver [HHWN02], a tool to visualize the time evolution of topics extracted from large collections of text documents. The thickness of a layer represented the popularity of a topic among the selected documents. The baseline was computed so that the drawing was symmetric with respect to the x-axis. No specific layer ordering algorithm was described, however the authors discussed about the possibility of letting the user decide the ordering, and of putting related layers close.

The paper from Byron and Wattenberg [BW08] formally introduced streamgraphs. It described the mathematical technique and design considerations behind the visualization used in a 2008 New York Times article showing box office revenues for 7500 movies over 21 years. The authors introduced the concept of *wiggle* as a measure of distortion of layers. Minimizing this measure aimed at both making layers more readable, and at avoiding the artificial look of ThemeRiver drawings in favor of a more natural, river-alike flowing. For ordering layers, the authors outlined how layers with high wiggle values should not be at the center of the drawing to avoid distortions to the other layers around them. Layers produced from box office revenues tend to have a sudden increase in the first weeks out, then they rapidly decrease as the interest in the movies declines. For this reason, layers were ordered by their "onset" time, with older movies at the center of the drawing and newer movies at the edges. It was left as future work to study how layers could be ordered by using the wiggle as a measure of quality. The paper also discussed layer coloring and labeling.

Many other works used streamgraphs, but most of them are applications of the concepts of [HHWN02] and [BW08], which were extended to work on specific data.

Several works employ streamgraphs for visualizing topics extracted from text documents. TIARA [LZP*12] is a system that visualizes the temporal trend of topics, augmenting streamgraphs by adding keyword clouds inside layers. The ordering of layers is chosen on the basis of several contrasting criteria, which include the on-set approach of [BW08], and a new measure of the volatility of a layer. Although the paper does not evaluate the effectiveness of the ordering algorithm, their solution is one of the few attempts to order layers with an approach different from [BW08]. TIARA also studied the problem of how to fit a keyword cloud at a specific time point in a layer. In this paper we solve a different problem of finding the best time point in a layer to fit the largest possible label of a certain aspect ratio. HierarchicalTopics [DYW*13] deals with hierarchies in topics extracted from text documents. Coordinated views show both the topic hierarchy by means of a tree, and the time evolution of topics through a streamgraph with the algorithm of [HHWN02]. While a streamgraph is used to visualize a single level of the hierarchy, the user can put several of them next to another to compare different levels. In [XWW*13], streamgraphs are used to display the competition for public attention between various topics promoted by multiple types of opinion leaders in agenda-setting on social media. Another work dealing with the visualization of topics is the system described in [DGWC10]. The system supports the visualization of dynamic data, which the user can navigate by panning the current time interval. Layers are totally ordered by a global measure of "newness", that is the time of first appearance of a topic, and never change position. The on-set approach of [BW08] was ineffective in a dynamic setting.

Other works visualize the complex relationships between time series through modified versions of streamgraphs, to overcome the natural limits of this metaphor. LoyalTracker [SWL*14a] shows the time evolution of the "loyalty" of users of search engines through a streamgraph-like metaphor. TextFlow [CLT*11] focuses on the relations between topics extracted from text documents, and how they merge and split over time. The visualization has some similarity with streamgraphs, but topics can split and merge over time, and gaps between topics are allowed for outlining branching behaviors. RoseRiver [CLWW14] extends this approach to dynamic hierarchies of topics. EvoRiver [SWL*14b] visualizes how competitive or cooperative topics are in attracting attention on social media.

Other areas of visualization studied problems similar to that of streamgraphs, namely, optimal ordering and sloping of lines to minimize the wiggle. In Storyline visualization [TM12, LWW*13], this is achieved by genetic algorithms [TM12], or by quadratic programming [LWW*13]. In directed graph visualiza-

tion [GKNpV93], polyline edges across multiple layers are kept as parallel to the vertical direction as possible by linear programming.

3. Finding a Baseline via Wiggle Optimization

In this section we define the concept of wiggle for a streamgraph, and show how to compute a baseline for a streamgraph such that the wiggle is minimized. After discussing the limits of the existing techniques, we describe our solution, which encompasses a new definition of wiggle and an optimization method.

3.1. Basic Concepts

We assume we are working with discrete data, that is, a time series is a sequence of m numbers. Given an ordered list of time series, we assume that the ordering of the layers in a streamgraph follows that of the list. We denote f_i as the i-th time series, where $i = 1,2,\ldots,n$. Given a streamgraph of series f_i, we denote g_i as the sequence of y-coordinates corresponding to the points of series f_i. As a particular case, g_0 denotes the baseline of the streamgraph. Also, note that $g_i = g_0 + \sum_{j=1}^{i} f_j$ for $i = 1,2,\ldots,n$.

Wiggle is a metric introduced in [BW08] to measure the aesthetics of a streamgraph. Intuitively, the wiggle of a streamgraph can be thought of as an indication of how much it fluctuates. For example, a streamgraph with only flat layers has wiggle equal to 0. Wiggle is a measure of the visual complexity of a streamgraph, since distorted layers are harder to understand, e.g., an increasing trend in the thickness of a layer could be hidden by a visual distortion that visualizes the layer as a steep descent. The "weighted wiggle", which gives more importance to layers with higher thickness and is thus consider better, is defined in [BW08] as:

$$\mathrm{ww}_2(g_0) = \sum_{i=1}^{n} f_i \left(\frac{g_i' + g_{i-1}'}{2} \right)^2 \tag{1}$$

In (1), g' is the derivative of g. By definition, wiggle is a function which returns a number (*wiggle value*) for each x-coordinate of a streamgraph. However, with a slight abuse of notation, we also define the *wiggle value of a streamgraph* as the sum of the wiggle values of the streamgraph over all x-coordinates. Further, the wiggle value of a layer is the wiggle value of a streamgraph composed only of that layer and having a flat baseline. Finally, the wiggle value of a list of ordered layers is the wiggle value of the streamgraph that has that layer ordering and a baseline that minimizes the wiggle.

With a fixed layer ordering, the wiggle of a streamgraph depends only on the baseline. A method to find a baseline g_0 that minimizes the wiggle is described in [BW08], and consists of taking the derivatives of (1) with regard to g_0', and set them to zero. The optimization problem can be solved with a per-value approach, restricting (1) to a single x-coordinate of the streamgraph. Derivatives in the equations can be computed with backward finite differences.

3.2. Limits of Existing Techniques

Equation 1 was used to produce aesthetically pleasant streamgraphs of box office revenues in The New York Times article. It is effective in visualizing relatively smooth time series, but suffers from

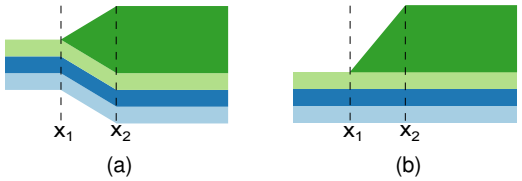


Figure 1: *Different baselines: (a) with weighted 2-norm minimization, notice the wiggle on all layers; (b) with 1-norm minimization: only the thick green layer has a wiggle.*

unpleasant distortions when a layer has sudden changes in thickness, or *jumps*. In Fig. 1 two possible drawings of four ordered layers are depicted, where one layer is shorter than the others. It is easy to see that the streamgraph in Fig.1a is affected by distortions at x_1 and x_2, that is, in correspondence of the first non-zero value of the short layer at x_2. Much of these are avoidable, as shown in Fig. 1b. Equation 1 gives a lower wiggle value for Fig. 1a than that for Fig. 1b. For example, assume that the short layer has thickness equal to 4 and the others have thickness equal to 1. Also, assume $x_2 - x_1 = 1$. In Fig. 1a, from bottom to top, values g_i', with $i = 0\ldots4$, at x_2 are -2, -2, -2, -2, and 2, respectively, therefore $\mathrm{ww}_2 = 12$. In Fig. 1b, with a similar reasoning, we obtain $\mathrm{ww}_2 = 16$. This means that, when minimizing (1), a baseline like Fig. 1a is preferred over that of Fig. 1b. In real data, the presence of many layers can amortize the distortion due to a jumping layer, though it is still noticeable and unpleasant, see Fig.2a.

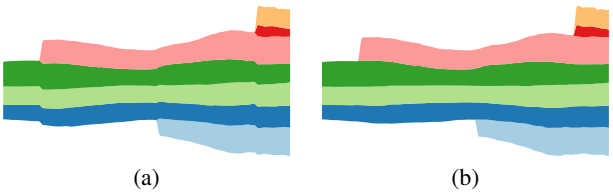


Figure 2: *Different baselines: (a) with 2-norm, distortions are present; (b) with 1-norm, all layers are smooth.*

There are two reasons for the unappealing distortions made by a baseline computed by minimizing (1). First, consecutive wiggle pairs (g_i', g_{i-1}') are cancelled if the two terms have equal absolute value and opposite sign. Such pairs are ignored in the minimization process, regardless of the amount of distortion they cause to other layers. In Fig. 1a, this happens with g_3' (light green) and g_4' (dark green). Also, the wiggle is defined based on 2-norm, which tends to favour many small "wiggles" to a large one, as seen in Fig. 1a and Fig. 2a.

3.3. Optimal Weighted Wiggle Under 1-norm

We define the 1-norm based weighted wiggle as follows:

$$\begin{aligned} \mathrm{ww}_1^a(g_0) &= \sum_{i=1}^{n} f_i \left(\frac{\left|g_i'\right| + \left|g_{i-1}'\right|}{2} \right) = \sum_{i=0}^{n} w_i \left|g_i'\right| \\ &= \sum_{i=0}^{n} w_i \left| g_0' + \sum_{j=1}^{i} f_i' \right| = \sum_{i=0}^{n} w_i \left| g_0' - p_i \right|. \end{aligned} \tag{2}$$

```
function OPTIMALBASELINEDERIVATIVE(w, p)
    d ← −∑_{i=0}^{n} w_i
    for i = 0, 1, ..., n do
        d ← d + 2w_i
        If d ≥ 0 return p_i
    end for
end function
```

Figure 3: *The optimal baseline algorithm*

Mathematically, the quadratic function involved in the 2-norm based definition of wiggle is smooth and easy to optimize, which may be part of the reason why Byron and Wattenberg chose to define wiggle that way. However in the following we show that the 1-norm based definition, though non-smooth, can also be solved easily, but does not suffer from the distortions due to sudden jumps.

Similarly to (1), we work on a per-value basis, that is, at a fixed x-coordinate of the streamgraph. We denote $w_i = \frac{1}{2}(f_i + f_{i+1})$ (we assume $f_0 = f_{n+1} = 0$), and $p_i = -\sum_{j=1}^{i} f_i^{'}$ (we assume $p_0 = 0$). The wiggle is based on 1-norm, which avoids uniform small jumps across the layers, a problem discussed in Section 3.2 that affects the 2-norm wiggle, in favour of a few larger jumps. This sparsifying effect is similar to the use of 1-norm for regularization in regression, known as *lasso* [Tib94]. Also, the norm is computed for each g_i' term, which avoids the cancelling of terms with equal absolute value and opposite sign.

To minimize (2), we first observe some properties of its derivative with respect to $g_0^{'}$, which is $d(g_0^{'}) = \sum_{i=0}^{n} w_i \ \mathrm{sgn}\left(g_0^{'} - p_i\right)$. With a slight abuse of notation, we reorder the p's from small to large and still denote them as p, in such a way that $p_0 \leq p_1 \ldots \leq p_n$. When looking at the $n+2$ intervals defined by this sequence from left to right, the derivative of the wiggle, when $g_0^{'} \in (-\infty, p_0)$, is $d_{-1} = -\sum_{i=0}^{n} w_i$. The derivative when $g_0^{'} \in (p_0, p_1)$ is $d_0 = d_{-1} + 2w_0$, the derivative when $g_0^{'} \in (p_1, p_2)$ is $d_1 = d_0 + 2w_1$, etc. That is, the derivative of the wiggle is negative for $g_0^{'} < p_0$, it increases as $g_0^{'}$ increases, and it is positive for $g_0^{'} > p_n$. Therefore, the minimum wiggle is achieved at the point where the derivative changes from negative to non-negative. Fig. 3 shows an algorithm for finding the optimal $g_0^{'}$. Effectively, it finds a weighted median of p_i's with weights w_i's. The returned value can be numerically integrated to obtain g_0, that is, a baseline value that minimizes (2).

Fig. 2 compares the effect of computing a baseline with 2-norm minimization, and with this technique. The baseline in Fig. 2a is computed by minimizing (1), and it is affected by distortion near the boundaries of short layers. The baseline in Fig. 2b is computed by minimizing (2), resulting in a smoother drawing.

4. Layer Ordering

This section describes an algorithm for ordering the layers of a streamgraph that is effective for general time series. It involves generating an initial good ordering, and a subsequent iterative refinement of the ordering.

4.1. Limits of Existing Techniques

In [BW08] an algorithm for ordering layers is proposed. Layers produced by box office revenues tend to have a sudden increase in the first weeks out, then they rapidly decrease as the interest in the movies declines. For this reason, layers were sorted by their "onset" time, defined as the x-coordinate of the first non-zero value. Ordering was done with an "inside-out" approach, which stacks the sorted layers by alternatively putting them above or below the two edges of the drawing. As a result older movies appear at the center of the drawing and newer movies at the outskirt, and the overall effect is visually appealing. Different implementations are possible, for example the D3 library [Bos] implements the algorithm by ordering layers by the x-coordinate of their maximum value. These approaches work well for movie data but are not appropriate for general data. For example, Fig. 5a is an ordering produced with the on-set method of [BW08]. Although layer 10 has an early onset time, it cannot be placed at the center of the drawing without distorting other layers. In [BW08] two alternative, more general approaches for ordering layers are briefly introduced. The first one implies measuring the amount of change in a layer by a "volatility" metric, and using it for ordering layers with the inside-out method in place of the on-set time. In such an ordering, layers with high volatility are placed as far away from the center of the drawing as possible, so not to perturb the other layers around them. Putting calm layers at the center also gives a good support for stacking other layers. The second method consists of ordering layers so that their wiggles mutually cancel out to the greatest extent possible. However, none of these ideas were implemented in [BW08].

4.2. Initial Ordering

`BestFirst` is the algorithm that we designed for producing an initial layer ordering. It is based on the intuition introduced in [BW08] that layers can mutually cancel their wiggle, and in fact it is a greedy implementation of that idea. The algorithm starts from an empty ordering and iteratively adds layers, choosing the one with the best wiggle at each iteration. More specifically, the algorithm starts with a straight line, whose two sides are labeled as *current top baseline* and *current bottom baseline*, and defined as two sequences of 0's. Then, all layers still to insert are tested. Each of them is stacked on both the current top and bottom baseline, and we compute the wiggle produced by each choice. Finally, the layer-baseline pair with the lowest wiggle is selected for insertion. The chosen current baseline is then updated by adding the thickness of the inserted layer. An iteration is done for each layer still to be inserted.

`BestFirst` is effective at keeping "calm" layers inside the drawing and putting the others outside. However, on some instances it can produce aesthetically unpleasant orderings, such as in Fig. 5b, where it is easy to see that layer 24 and layer 6 should be switched (see also Fig. 4), and that layer 30 should be on top of them. The reason for the wrong decisions is that `BestFirst` computes the wiggle of a layer as an integral along the layer. Parts of a layer that have thickness equal to zero do not contribute to the wiggle, and this could mitigate the sudden increment in value (despite the high derivative) that is present at the boundaries of the zero intervals. As a result, a short layer tends to have much smaller wiggle than

a long layer, and is more likely to appear closer to the center of the streamgraph, distorting any other layers on top of it. Note that the phenomenon can happen also in absence of zero intervals. For example, a layer could start with a small constant value, followed by a jump. This problem is tricky to handle with any approach that makes greedy ordering decisions purely based on a measure of the amount of change of individual layers (including our `BestFirst` and the method based on volatility briefly introduced in [BW08]). This limits its general use, given that data with jumps is common in the real world (e.g., demographic data could have been collected starting from different dates for different countries).

4.3. Iterative Refinement

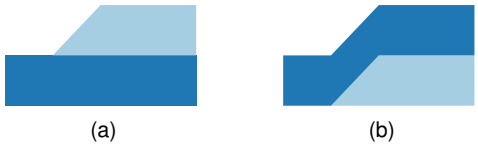


Figure 4: *Two orderings of two layers on a flat baseline. Case (b) produces a distortion, case (a) is preferred.*

Instead of trying to characterize the volatility of a layer purely based on the layer itself, for making better ordering decisions, we claim that a better metric of volatility can be indirectly measured by the effect a layer has on other layers. Referring to Fig. 5b, it becomes clear that the position of layer 24 is wrong only after layer 6 has been stacked on top of it, since the latter suffers from a distortion that would have been avoided by stacking it in another suitable position. Also, Fig. 4a is a better ordering than Fig. 4b, since in the latter the bottom layer does not properly "support" the top one, and distorts it. Following this intuition, we propose a refinement algorithm, `TwoOpt`, to further improve the initial ordering of `BestFirst`. Starting from an initial ordering, the algorithm iteratively compares two neighboring layers and decides which of the two possible orderings gives the lowest wiggle. The result of the iterative scans is that, eventually, layers with a high value of wiggle, or those with a tendency to induce distortions to the layers on top of them, are pushed towards the edges of the drawing.

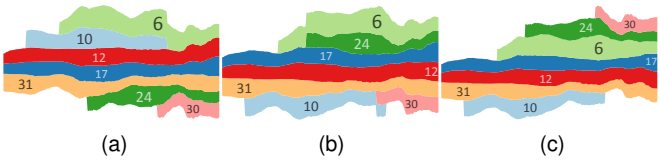


Figure 5: *Layer orderings produced by different algorithms: (a) on-set; (b) BestFirst; (c) TwoOpt. The first two have layer distortions. Labels identify layers and were positioned with the algorithm described in Section 5.*

Fig. 6 gives the pseudo code for `TwoOpt`. It starts from the initial ordering from `BestFirst`. Then, from the center of the ordering, it performs several inside-out scans towards the top and the bottom. Each scan uses the wiggle heuristic to compare every pair

```
1:  function TwoOpt(init, m, nr, ns, wig)
2:      ▷ init: initial ordering; m: the center of the drawing; nr: the number
    of repeats; ns: the number of scans; wig: the wiggle heuristic.
3:      ordering ← init, bestWiggle ← ∞, bestOrdering ← nil
4:      ▷ Do random executions and take the best result
5:      for r ← 1..nr do
6:          ▷ Do not discard the initial ordering
7:          If r > 1 shuffle the ordering
8:          ▷ Execute several inside-out scans
9:          for s ← 1..ns do
10:             for j ← m..lenght(ordering)−1 do
11:                 ▷ Compare the wiggle of j, j+1 and j+1, j
12:                 res ← cmp(j, j+1, ordering, wig)
13:                 if res > 0 then
14:                     swap(j, j+1, ordering)
15:                 end if
16:             end for
17:             Scan in the other direction, with j ← m − 1..2
18:         end for
19:         ▷ Evaluate the produced ordering
20:         graph ← midlineGraph(ordering, m − 1, m)
21:         currWiggle ← wig(graph)
22:         if currWiggle < bestWiggle then
23:             bestWiggle ← currWiggle
24:             bestOrdering ← ordering
25:         end if
26:     end for
27:     return bestOrdering
28: end function
```

Figure 6: *The algorithm for ordering layers*

of adjacent layers, and decides whether swapping them decreases their wiggle. More precisely, the two orderings of a pair of layers are stacked on a flat baseline, then their respective wiggles are computed and compared. Note that the reason we compare the orderings of two layers on a flat baseline, instead of evaluate using the total wiggle on the best baseline each time, is that the former is computationally cheaper, and that a good ordering tends to give relatively flat middle layers, hence using a flat baseline is a reasonable approximation. The scans are repeated a given number of times and each repetition produces a new ordering, which is evaluated through the wiggle heuristic. To do this, the ordered layers are stacked with a baseline such that the center of the drawing, i.e., the start point of the inside-out scan, is a straight line. Then the wiggle of this streamgraph is computed. The algorithm is repeated several times, shuffling the layers at each repetition. Finally, the ordering with the lowest wiggle is returned as result. Fig. 5c is a drawing with an ordering produced by `TwoOpt`, which does not have the distortions of the on-set and `BestFirst` versions (see respectively Figures 5a and 5b). The reason for the layer shuffling is that the initial ordering induces a bipartition of the layers, which is not changed by `TwoOpt`, since the two partitions are processed independently during the inside-out reordering. If two layers in a same partition distort each other in both their possible orderings, it is a problem that `TwoOpt` cannot solve. In some instances, moving one of the two layers to the other partition removes the problem. This is the situation shown in Fig. 5b, in which layer 10 and layer 30 are on the same side of the central reference line, and swapping them does not improve their wiggle. Fig. 5c is an alternative ordering in which the two conflicting layers are on the two sides of the

drawing and do not distort each other. `TwoOpt` shuffles the layers in an attempt to avoid bad bipartitions like Fig. 5b, and the chances of success increase with the number of tries.

5. Labeling of Layers

An important part of the design of a streamgraph is the placement of labels for the layers. Ideally a label is visually associated with the data it represents. In this section we propose a fast label placement algorithm that scales linearly to the data size, and logarithmically to the ratio between the largest and smallest desirable font size.

5.1. Limits of Existing Techniques

Byron and Wattenberg [BW08] placed labels within the layers themselves. The font size of the labels is adjusted to fit each layer. The labels are located to maximize the font size. They adopted a brute-force algorithm, but noted that "the online interactive piece does not use this proposed label placement strategy because of the poor real-time performance of the brute-force algorithm." Based on their brief description, we suspect that the computational complexity of their algorithm may be quadratic to the data size.

5.2. Algorithm Design

We assume that the top and bottom layers for which we need to place the label is defined by the sequences $\{t_i | i = 0,...,m-1\}$ and $\{b_i | i = 0,...,m-1\}$, respectively, with $t_i > b_i$.

Suppose we have a label, with width w and aspect ratio σ (defined as the ratio between width and height of the label), to be placed in this layer such that it is centered at $x = i$. Then the lower boundary of this label must be greater than b_j for all $j = i - w/2,\ldots,i+w/2$. Thus the lowest the label could reach along the y-direction is the top most point of the bottom sequence within the x-window size of w centered at i, namely $B_i = \max_{i-w/2\le j\le i+w/2} b_j$. Similarly the highest it can reach along the y-direction is the bottom most point of the top sequence within the x-window of size w centered at i, or $T_i = \min_{i-w/2\le j\le i+w/2} t_j$. Fig. 7 shows a layer with $m = 10$ points and $w = 4$. On the left side, the shaded area shows the case with $i = 2$. Here we have $B_2 = 1$ and $T_2 = 2$. On the right side, when $i = 7$, we have $B_7 = 1$ and $T_7 = 3$.

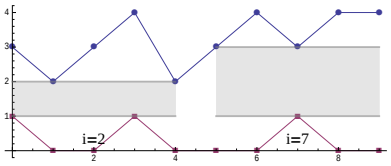


Figure 7: *Lower and upper bounds for a label with width $w = 4$. Left: $i = 2$. Lower/upper bounds $B_2 = 1$ and $T_2 = 2$. Right: $i = 7$. Lower/upper bounds $B_7 = 1$ and $T_7 = 3$*

Therefore to place a label with the largest possible font size, we can start from a large label width w based on the largest font size to use and the number of characters, then vary i from $w/2$ to $n-1-w/2$, and check whether the achievable height of the label, $H_i = T_i - B_i$, satisfies the aspect ratio requirement $w/H_i \le \sigma$. If it is true, we can define the center of the label as $(i,\ 0.5*(B_i+T_i))$. If not, we shrink w by a constant factor (e.g., 10%), and repeat the process until a feasible solution is found. This process takes a number of steps logarithmic to the the ratio between the largest font size, and largest feasible font size, because after that many steps, w will be small enough to allow a feasible solution. To compute B_i (or T_i) for $w/2 \le i \le m-1-w/2$, a naive way would be to compute the maximum (or minimum) of a sliding x-window of width w over the sequences b (or t). However this would take time $O(wm)$. Instead we can use a faster sliding window min/max algorithm that compute all mins or maxes in time $O(m)$.

This fast labeling algorithm is given in Fig. 8. In the algorithm, the *slidingWindowMin* and *slidingWindowMax* functions implement the faster sliding window min/max algorithm (see, e.g., [sli]).

```
1:  function LABELING(t, b, minFontWidth, maxFontWidth,
    label, sigma)
2:      w ← round(maxFontWidth * length(label))
3:      w_min ← round(minFontWidth * length(label))
4:      while w > w_min do
5:          T = slidingWindowMin(t, w)
6:          B = slidingWindowMax(b, w)
7:          imax ← argmax_i{B_i − T_i | i = 0, 1, ..., m − w − 1}
8:          hmax ← B_imax − T_imax
9:          center = (½w + imax, ½(T_imax + B_imax))
10:         if w ≤ σ * hmax then
11:             render label at center with width w and height w/σ
12:             return
13:         end if
14:         w ← w − max(1, round(0.1 * w))
15:     end while
16: end function
```

Figure 8: *The algorithm for layer labeling*

6. Time Complexity of the Algorithms

We analyze the complexity of our algorithms in the following. Recall that we assume that each time series is a sequence of m numbers, and there are n time series. Therefore, nm is the data size.

Like the 2-norm baseline algorithm [BW08], the proposed weighted 1-norm baseline algorithm works on a per-x-value basis. It takes $O(n)$ operations to find the baseline derivative using Algorithm 3 for each x-value. Thus the overall complexity is $O(nm)$.

Our ordering algorithms execute, as sub-procedures, the operations of stacking a set of layers on a baseline, and of computing its wiggle value. Both these operations take $O(nm)$ time.

`BestFirst` executes one iteration for each time series to add to the streamgraph, so there are $O(n)$ iterations. In turn, each iteration tests every time series that has not be added to the streamgraph, requiring another $O(n)$ factor. A time series is tested by stacking it on each of the two current baselines, and evaluating its wiggle. These two operations, executed on a single time series, take $O(m)$ time. Therefore, the total time complexity of `BestFirst` is $O(n^2 m)$.

`TwoOpt` depends on two parameters: the number of repeats r and the number of scans s. Refer to Fig. 6. First, the initial layer ordering is shuffled, which can be done in $O(n)$ time with the Fisher-Yates method. Then s inside-out scans are executed. For each of

them, time series are pairwise compared and $O(n)$ comparisons are performed. Comparing two time series means stacking them on a flat baseline in each of their two possible orderings. Constructing these two streamgraphs and computing their wiggles takes $O(nm)$ time. After the scans, a baseline for the ordered series is computed, such that the center of the resulting streamgraph, i.e., the start point of the inside-out scan, is a straight line. This requires we sum at each time point all series that are below the reference straight line. So, computing the baseline takes $O(nm)$ time. Finally, stacking the ordered layers on this baseline and computing its wiggle takes $O(nm)$ time. Since the entire procedure is repeated r times, the time complexity of `TwoOpt` is $O(r(n+snm+nm)) = O(rsnm)$.

The layer labeling algorithm finds the largest possible height for a fixed width of the label in time linear to m per layer. The overall complexity is $O(nmlog(r))$, where r is the ratio between the largest font size to try out and the largest feasible font size.

7. Experiments

We performed a numerical evaluation to compare the effectiveness and the efficiency of our layer ordering algorithm with the state of the art. The quality of an ordering is measured using the 1-norm (2) and the 2-norm (1) wiggle values. The hypothesis tested in these experiments is that an ordering algorithm designed to reduce the wiggle produces drawings with a lower wiggle than other algorithms.

7.1. Datasets

The experiments were conducted on a number of datasets, which we collected from real applications. These are:

- Unemployment: unemployment statistics [datb] for 31 European countries, from 1983 to 2013. 371 time points.
- Movies: U.S. box office revenues [data] of 161 movies which were on screen in the first six months of 2015. 28 time points.
- Sandy: number of calls to the NYC 311 [date] in the days before and after hurricane Sandy hit New York City, i.e., from 10/14/12 to 11/17/12, for 158 topics. 35 time points.
- Stocks: stock prices of 662 companies listed on the NASDAQ [datf] from 2005 to 2015. 121 time points.
- Marketcap: market capitalization of 662 companies listed on the NASDAQ [datg] from 1995 to 2015. 241 time points.
- Google: relative variations in volume of Google search traffic in the U.S. [datc] across 27 sectors of the economy from 2014 to 2015. 365 time points.
- Linux: number of commits on GitHub on the Linux kernel [datd] by 786 contributors from 2013 to 2015. 24 time points.

7.2. Experimental Design

The algorithms evaluated in the experiments were: Optimum, TwoOpt, TwoOptR, BestFirst, Onset, D3 and Random. TwoOpt and TwoOptR both implement the algorithm in Fig. 6, with the difference that TwoOpt applies BestFirst to produce an initial layer permutation, while TwoOptR starts from a random permutation. Optimum is a brute-force ordering algorithm that explores all layer permutations, and selects the one that minimizes the wiggle. Finally, algorithm Random orders the layers randomly. All algorithms were implemented in Javascript and the experiments were executed on a machine with this setting: (a) Mac OSX 10.9.5; (b) 2.3 GHZ Core i7; (c) 16 GB RAM; (d) Node.js 0.12.2.

Table 1: *Normalized wiggles on 8 randomly selected layers, averaged over 20 random selections. Each entry shows 1-norm/2-norm wiggles. Lower values (blue) are better than higher values (red).*

dataset	TwoOpt	TwoOptR	BestFirst	OnSet	D3	Random
unemp	0.16 / 0.05	0.17 / 0.01	0.22 / 0.23	0.26 / 0.25	1.00 / 1.00	0.56 / 0.42
movies	0.06 / 0.15	0.06 / 0.19	0.72 / 0.91	0.27 / 0.97	0.34 / 1.00	1.00 / 0.88
sandy	0.21 / 0.44	0.14 / 0.28	0.15 / 0.62	0.11 / 0.50	0.25 / 0.77	1.00 / 1.00
stocks	0.17 / 0.07	0.18 / 0.12	0.64 / 0.55	0.64 / 0.53	0.90 / 0.88	1.00 / 1.00
marketcap	0.03 / 0.02	0.02 / 0.02	0.17 / 0.07	0.31 / 0.40	0.70 / 0.55	1.00 / 1.00
google	0.13 / 0.17	0.14 / 0.20	0.59 / 0.44	0.82 / 0.84	1.00 / 1.00	0.71 / 0.74
linux	0.26 / 0.16	0.32 / 0.24	0.33 / 0.34	0.57 / 0.71	0.52 / 0.56	1.00 / 1.00

Table 2: *Normalized wiggles of algorithms on 50 randomly selected layers, averaged over 20 random selections. Each entry shows 1-norm/2-norm wiggles. Lower values (blue) are better than higher values (red).*

dataset	TwoOpt	TwoOptR	BestFirst	OnSet	D3	Random
unemp.	0.00 / 0.00	0.00 / 0.02	0.34 / 0.25	0.14 / 0.10	1.00 / 1.00	0.71 / 0.60
movies	0.00 / 0.00	0.39 / 0.07	0.10 / 0.08	0.10 / 0.42	0.27 / 0.46	1.00 / 1.00
sandy	0.00 / 0.00	0.32 / 0.22	0.04 / 0.06	0.59 / 0.60	0.30 / 0.33	1.00 / 1.00
stocks	0.00 / 0.00	0.00 / 0.00	0.76 / 0.46	0.38 / 0.50	1.00 / 0.96	0.91 / 1.00
marketcap	0.00 / 0.00	0.19 / 0.17	0.15 / 0.32	0.12 / 0.30	0.58 / 1.00	1.00 / 0.99
google	0.00 / 0.00	0.02 / 0.05	0.74 / 0.23	0.93 / 1.00	1.00 / 0.88	0.72 / 0.75
linux	0.00 / 0.00	0.43 / 0.33	0.05 / 0.02	0.83 / 0.81	0.66 / 0.57	1.00 / 1.00

To evaluate the effects of the number of layers on the result of the algorithms, we executed two families of experiments. In the first family, for each dataset we randomly selected 8 layers, and ordered them using each algorithm. Then, for each ordering we computed a streamgraph with a baseline that minimizes the wiggle (either 1-norm or 2-norm) and computed its wiggle value. This process is repeated 20 times and the average wiggle value is taken. The second family of experiments is similar to the first, but at every repetition we randomly selected 50 layers. The only exceptions are dataset Unemployment and Google, for which we selected 20 layers because the datasets contained fewer than 50 layers. Algorithm Optimum was not executed in the second family of experiments, because it would have taken too long on that number of layers.

7.3. Results

7.3.1. Wiggle Values

Tables 1 and 2 show the results of the first and the second family of experiments, respectively. Values are normalized by replacing each value x with $x' = \frac{x-min}{max-min}$, where min and max are the minimum and the maximum wiggle value in the same table row as x. In Table 1, min was always the wiggle value of algorithm Optimum. This algorithm is not shown, since its normalized wiggle was always 0.

Both tables outline that TwoOpt had the best performance, with the exception of dataset Sandy in Table 1. This confirms our hypothesis that an ordering algorithm specifically designed to reduce the wiggle produces drawings with a lower wiggle than other algorithms. TwoOptR had variable performance depending on the dataset, which suggests that starting from a BestFirst layer ordering gives better results than a random ordering.

While Table 1 shows that the wiggle of TwoOpt is close to

Table 3: *Running times (in milliseconds) of algorithms on 50 randomly selected layers, averaged over 20 random selections. Lower values (blue) are better than higher values (red).*

dataset	TwoOpt	TwoOptR	BestFirst	OnSet	D3	random	ww1a	ww2	labeling
unempl.	205.04	190.28	18.25	0.91	3.89	0.02	5.34	2.03	15.86
movies	38.05	32.24	8.57	0.26	1.03	0.02	0.92	0.49	23.05
sandy	64.93	58.85	9.91	0.28	1.23	0.02	1.11	0.67	28.54
stocks	378.25	365.74	30.78	0.69	2.83	0.02	3.64	1.34	48.10
marketcap	688.67	682.19	63.68	1.29	5.48	0.02	7.47	3.20	112.41
google	174.98	165.79	15.71	0.78	2.52	0.02	4.28	0.89	24.35
linux	46.81	36.93	8.04	0.23	0.96	0.02	0.86	0.44	30.99

the optimum, we noticed, by looking at the computed drawings, that selecting only 8 random layers from a big pool often put together one or two layers with huge thickness and several very thin layers, making the results less affected by the layer ordering and hence less significant for comparing the ordering algorithms. The results in Table 2 are more representative of a realistic scenario and we further discuss them. Refer also to the figures in the additional material, which represent the drawings produced by the various algorithms on one instance for each dataset in Table 2. BestFirst had very poor performance on datasets Stocks and Google. In these examples, some thick layers were put at the two edges of the drawing because they presented relatively high level of wiggle on specific parts due to their thickness. However, they were impacted by the cumulated wiggle of the many underlying thinner layers, which caused distortions along the entire layer extent. The greedy approach of BestFirst could not foresee this type of situations, which were adjusted by TwoOpt by moving the thick layers slightly towards the center of the ordering. Dataset Stocks also contained some jumps, for which BestFirst was not designed. OnSet had excellent performance on dataset Movies, which was expected. It also had good performance on datasets Unemployment and Marketcap, where many layers contained on-set points. Correctly handling these allowed OnSet to avoid orderings with high values of wiggle. On the other hand, in absence of on-set points, OnSet had poor performance (Sandy, Stocks) which at times are close to that of randomly ordering the layers (Google, Linux).

We illustrate the effect of ordering and baseline algorithms in Fig. 9. Clearly D3 (top) has a drifting medium line due to the way 2-norm handles the peaks of "SAP", which caused significant distortion to the other layers. Likewise, a drift of medium line is seen in Fig. 10. Fig. 11 shows the top 20 companies from the same dataset Marketcap, but from Jun 2001 to Jun 2011. In this figure, the effect of the ordering is more visible, because the lightgreen ("SAP") layer in Fig. 11a is placed by OnSet at the center of the drawing, distorting the other layers.

7.3.2. Running Times

Table 3 shows the running time of the various algorithms, averaged over all instances of Table 2. From the table, our ordering algorithms are slower than the state of the art, namely TwoOpt and BestFirst are slower than OnSet, D3, and Random, and TwoOpt was the slowest. Because of its iterative nature, it scanned all layers several times. Also, we configured it to do a number of layer scans equal to the number of layers, and a constant number of repeats. The number of time points in data greatly contributed to the running time of our algorithms. This is noticeable for datasets Unemployment, Stocks,

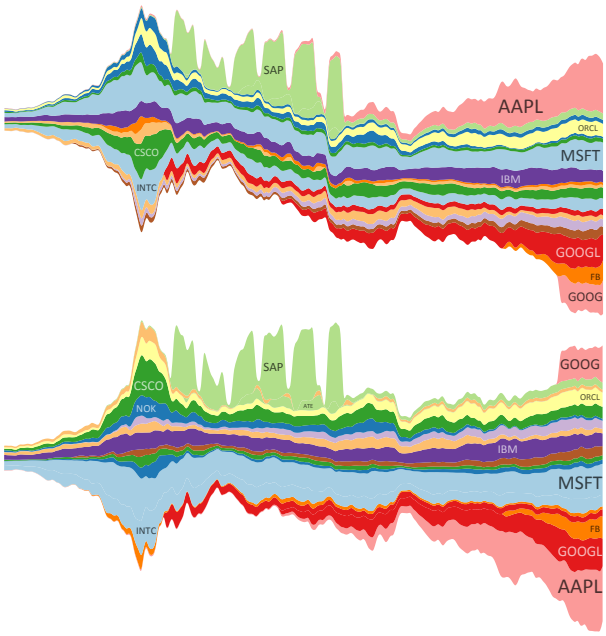


Figure 9: *Marketcap data for the top 50 companies from 1995-2015. Top: D3's implementation of Byron and Wattenberg [BW08]. Bottom: with* `TwoOpt` *ordering and a baseline algorithm that minimizes weighted 1-norm wiggle.*

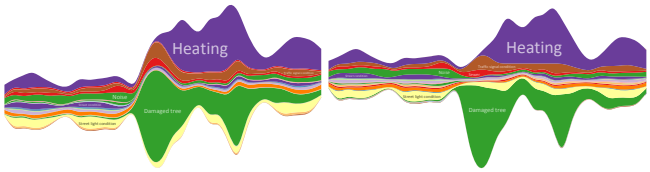
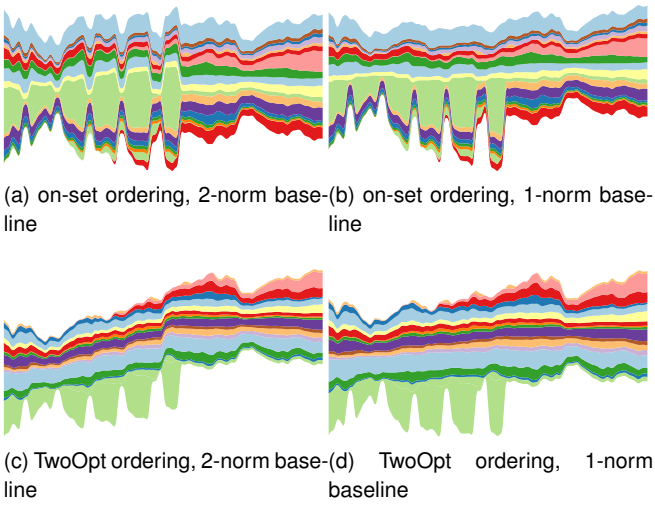


Figure 10: *Number of calls on 50 topics to the NYC 311 for the days before and after hurricane Sandy hit New York City. Left: D3's implementation of Byron and Wattenberg [BW08]. Right: with* `TwoOpt` *ordering and the 1-norm baseline algorithm.*

Marketcap, and Google, which contained many time points. When the number of layers and the number of time points are comparable, our algorithms are cubic in the size of the input (see Section 6 for a description of the time complexity). The good performance of OnSet and D3 were mainly due to the simplicity of their logic, which allowed for very efficient implementations. OnSet performed better than D3 because it stops scanning a layer at the first non-zero value, while D3 performs a full scan. We conclude that considering the wiggle for ordering the layer requires complex algorithms with non-negligible running times. However, it is worth pointing out that running time for each ordering algorithm is below 1 second, which is acceptable for real uses. The 1-norm and 2-norm baseline algorithms (ww1a and ww2) both take very little time. The labeling algorithm is also very fast. We did not compare it with the labeling algorithm of [BW08] since the latter was described as a brute-force algorithm with "poor real-time performance", too brief a description to allow a proper implementation.



(a) on-set ordering, 2-norm baseline (b) on-set ordering, 1-norm baseline

(c) TwoOpt ordering, 2-norm baseline (d) TwoOpt ordering, 1-norm baseline

Figure 11: *Top 20 companies from dataset Marketcap, from Jun 2001 to Jun 2011, with different layer orderings and baselines. (a) The light-green layer is at the center of the drawing, distorting the other layers. (b) Because of the 1-norm baseline, the distortion caused by a wrong ordering of the light-green layer is limited to the lower half of the drawing. (c) The light-green layer "pushes" up the other layers because of the 2-norm baseline, creating an upward-going drawing. (d) None of the previous issues are present.*

8. Discussion

In [BW08] several design issues of stacked charts are introduced and discussed. A central consideration is the trade-off between the readability of single layers and of the total. While a flat baseline allows for an easy readability of the total, single layers can be significantly distorted. The baseline of [HHWN02] produces drawings that are symmetric with regard to the x-axis, reducing the distortion of single layers without compromising the readability of the total. Also, it minimizes the wiggle of the silhouette, making the outline of the drawing less "spiky". However, [BW08] outlines how, without explicit care, distortions make a layer hard to understand and also propagate to other layers. Their 2-norm baseline addresses these two issues, increasing the readability of single layers at the expense of the global trend. Our 1-norm baseline is a new step in the same direction. It is designed for better handling layer jumps and, in general, can produce drawings where the readability of individual layers is further prioritized. We agree with the implicit assumption of [BW08] that such a property is desirable, but believe that applications should be evaluated case by case, based on data and user tasks, to determine the most appropriate trade-off between the readability of layers and of the global trend. We note that both 1-norm and 2-norm are reasonable definitions of wiggle to use when there are not sudden jumps. However, it is not clear to us which definition, or an alternative definition, correlates best with the human visual perception of wiggle. For example, in the current definitions, wiggles are additive over time. A time series $\{y, y+1, y+2\}$ is considered to have the same wiggle as $\{y, y+1, y\}$, but perceptually the former is smoother, while the latter represents a spike. Thus a wiggle definition based on both first- and second-order derivatives may be more expressive, but possibly harder to optimize.

BestFirst in combination with TwoOpt, in our experiments, proved to be an effective ordering algorithm for generic data. However, we believe that OnSet produced more pleasant drawings on the *movies* dataset, with layers that appear to "flow" towards the center and a general organic look of the drawing (see the additional material). This is not surprising, since OnSet was specifically designed for this type of data. We also noticed that BestFirst and TwoOpt tend to put thin layers at the center of the drawing – a visually prominent position. This can be aesthetically undesirable, because thick layers tend to be more important than thin ones. It could be beneficial to modify the ordering algorithms to take into account layer thickness. Finally, although the metrics in our experiments showed differences, we did not see significant differences visually between the drawings produced by the various ordering algorithms on datasets *linux* and *google* (see [DH]). This could be due to the low variability in the shape of the layers of those datasets, and implies that in some applications the higher running time of complex algorithms like TwoOpt may not always be advantageous.

In the presentation of our fast layer labeling algorithm, for simplicity we assumes that labels have integer width. This is reasonable for time series with hundreds or more points, for which integer width is fine-grained enough. For time series with few points, if a feasible position for a label is not found, our algorithm adaptively interpolates the time series by doubling the number of points, and applies the layer labeling procedure to the new time series, until a feasible position for the label is found. This effectively makes the label width a floating point number.

9. Conclusions

The aesthetics of a streamgraph is affected by the ordering of the layers, the shape of the baseline of the drawing, and the labeling for the layers. This paper advances the state of the art for streamgraphs by proposing an ordering algorithm that works well regardless of the properties of the input data, a 1-norm baseline procedure that overcomes the distortion associated with the existing baseline algorithm, particularly when there are sharp changes in the time series, and an efficient layer labeling algorithm that scales linearly to the data size. We demonstrate both qualitatively and quantitatively the advantage of our algorithms over existing techniques on a number of real world data sets.

As far as we know, streamgraphs have been used exclusively for time series with only positive values. The feasibility of using them to visualize time series with both positive and negative values remains an open problem.

10. Acknowledgments

The work of the first author is partially supported by the Italian Ministry of Education, University, and Research (MIUR) under PRIN 2012C4E3KT national research project "AMANDA: Algorithmics for MAssive and Networked DAta", and by the EU FP7 project "Preemptive: Preventive Methodologies and Tools to Protect Utilities", grant no. 607093.

We are particularly thankful to Yahoo! for hosting the first author.

References

[Bos] BOSTOCK M.: Data-driven documents. http://d3js.org/. 2, 4

[BW08] BYRON L., WATTENBERG M.: Stacked graphs – geometry & aesthetics. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (Nov. 2008), 1245–1252. 1, 2, 3, 4, 5, 6, 8, 9

[CLT*11] CUI W., LIU S., TAN L., SHI C., SONG Y., GAO Z., QU H., TONG X.: Textflow: Towards better understanding of evolving topics in text. *Visualization and Computer Graphics, IEEE Transactions on 17*, 12 (Dec 2011), 2412–2421. 2

[CLWW14] CUI W., LIU S., WU Z., WEI H.: How hierarchical topics evolve in large text corpora. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (Dec 2014), 2281–2290. 2

[data] Box office mojo. http://www.boxofficemojo.com. 7

[datb] Eurostat. http://ec.europa.eu/eurostat/en/web/lfs. 7

[datc] Google domestic trends. https://www.google.com/finance/domestic_trends. 7

[datd] Linux kernel. https://github.com/torvalds/linux. 7

[date] Nyc opendata. https://nycopendata.socrata.com/Social-Services/311-Service-Requests-from-2010-to-Present/erm2-nwe9. 7

[datf] Yahoo! finance. http://finance.yahoo.com. 7

[datg] Ycharts. https://www.ycharts.com. 7

[DGWC10] DORK M., GRUEN D., WILLIAMSON C., CARPENDALE S.: A visual backchannel for large-scale events. *Visualization and Computer Graphics, IEEE Transactions on 16*, 6 (Nov 2010), 1129–1138. 2

[DH] DI BARTOLOMEO M., HU Y.: Additional streamgraph drawings. http://streamgraphs.github.io/. 2, 9

[DYW*13] DOU W., YU L., WANG X., MA Z., RIBARSKY W.: Hierarchicaltopics: Visually exploring large text collections using topic hierarchies. *IEEE Transactions on Visualization and Computer Graphics 19*, 12 (Dec 2013), 2002–2011. 2

[GKNpV93] GANSNER E. R., KOUTSOFIOS E., NORTH S. C., PHONG VO K.: A technique for drawing directed graphs. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING 19*, 3 (1993), 214–230. 3

[HHWN02] HAVRE S., HETZLER E., WHITNEY P., NOWELL L.: Themeriver: Visualizing thematic changes in large document collections. *IEEE Transactions on Visualization and Computer Graphics 8*, 1 (Jan. 2002), 9–20. 1, 2, 9

[LWW*13] LIU S., WU Y., WEI E., LIU M., LIU Y.: Storyflow: Tracking the evolution of stories. *IEEE Trans. Vis. Comput. Graph. 19*, 12 (2013), 2436–2445. 2

[LZP*12] 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 Trans. Intell. Syst. Technol. 3*, 2 (Feb. 2012), 25:1–25:28. 2

[sli] Sliding window maximum. http://articles.leetcode.com/2011/01/sliding-window-maximum.html. 6

[SWL*14a] SHI C., WU Y., LIU S., ZHOU H., QU H.: Loyaltracker: Visualizing loyalty dynamics in search engines. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (Dec 2014), 1733–1742. 2

[SWL*14b] SUN G., WU Y., LIU S., PENG T. Q., ZHU J. J. H., LIANG R.: Evoriver: Visual analysis of topic coopetition on social media. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (Dec 2014), 1753–1762. 2

[Tib94] TIBSHIRANI R.: Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B 58* (1994), 267–288. 2, 4

[TM12] TANAHASHI Y., MA K.-L.: Design considerations for optimizing storyline visualizations. *IEEE Trans. Vis. Comput. Graph. 18*, 12 (2012), 2679–2688. 2

[XWW*13] XU P., WU Y., WEI E., PENG T. Q., LIU S., ZHU J. J. H., QU H.: Visual analysis of topic competition on social media. *IEEE Transactions on Visualization and Computer Graphics 19*, 12 (Dec 2013), 2012–2021. 2