Aesthetics and Ordering in Stacked Area Charts
Supplementary Material: Benchmarking Results

Steffen Strunge Mathiesen and Hans-Jörg Schulz
Aarhus University, Denmark

1 Setup

We implemented our ordering algorithm (`UpwardsOpt`) as well as the state-of-the-art algorithm (`BestFirst+TwoOpt`) as a Python 3.6 backend to a Tableau v.2019 chart. Charts were exported in a 16×10 aspect ratio. Our implementations are available as open source at https://github.com/steffen555/UpwardsOpt.

All benchmarks were run on a 2017 27 inch iMac 5K with a 3.4 GHz Intel Core i5 processor and 40 GB RAM. The datasets used for our experiments 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:

- *unempl* ($n = 28, m = 443$): Monthly unemployment numbers in the EU countries between Jan. 1983 and Nov. 2019. Source: https://ec.europa.eu/eurostat/web/lfs/data/database
- *sandy* ($n = 183, m = 33$): Daily number of 311 calls in NYC by subject between Oct. 14 and Nov. 15 2012. Source: https://data.cityofnewyork.us/Social-Services/311-Service-Requests-from-2010-to-Present/erm2-nwe9
- *covid* ($n = 206, m = 113$): Daily number of new Covid-19 cases per country between Dec. 31 2019 and Apr. 21 2020. Source: https://ourworldindata.org/coronavirus-source-data
- *hotel* ($n = 334, m = 115$): Weekly hotel bookings by travel agent between Jul. 2015 and Sep. 2017. Source: https://www.kaggle.com/lucacosseddu/hotelbookings-cleaned
- *messages* ($n = 604, m = 135$): Monthly message count per Facebook contact between May 2011 and Sep. 2019. Source: https://github.com/steffen555/UpwardsOpt/blob/main/datasets/messages.csv
- *liquor* ($n = 695, m = 240$): Weekly liquor sales revenue in Iowa by liquor brand between Jan. 2012 and Aug. 2016. Source: https://www.kaggle.com/residentmario/iowa-liquor-sales
- *movies* ($n = 881, n = 51$): Weekly US box office revenues by movie between Jan. and Dec. 2019. Source: https://www.boxofficemojo.com/weekly/by-year/2019/
- *names* ($n = 1000, m = 135$): Yearly number of new-borns for each of the top-1000 US names between 1880 and 2014. Source: https://www.kaggle.com/kaggle/us-baby-names

2 Procedure

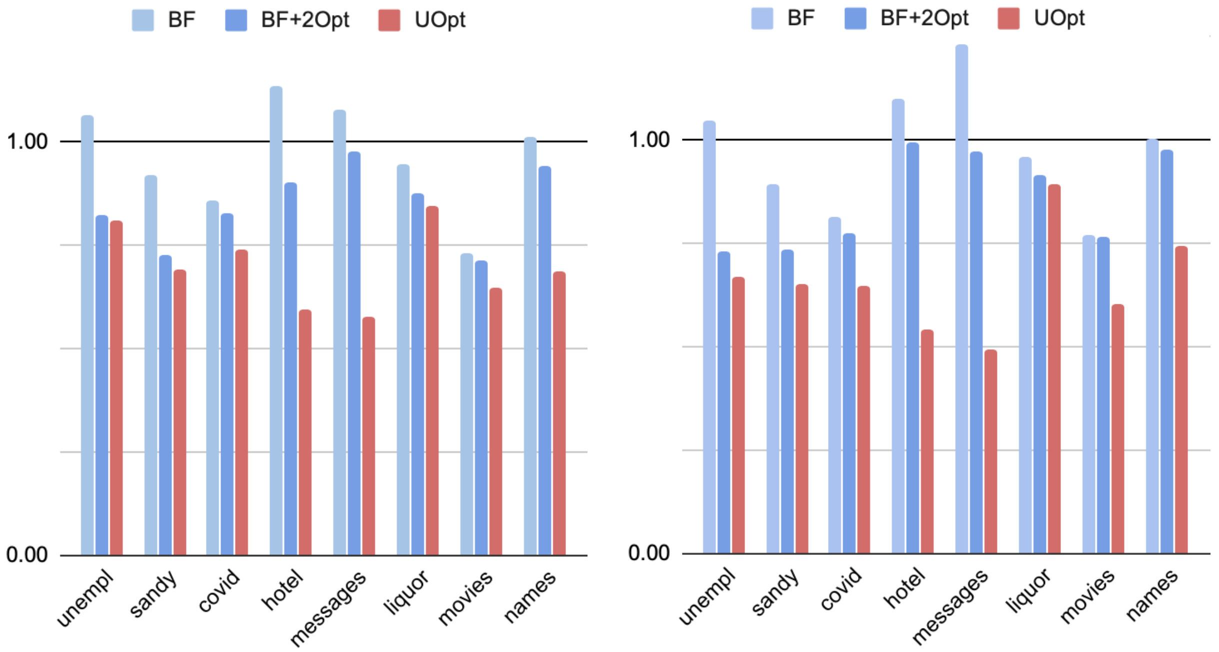
For a fair comparison between the different algorithms, we restricted the objective function to two cases: optimising only for flatness (minimising wiggle) and optimising only for straightness (minimising bumps). The significance exponent was set to $s = 1$, we used only outer lines – i.e., $\alpha = 0.5, \beta = 0.0, \gamma = 0.5$ – and a 1% threshold of minimum improvement. As a neutral reference point for our benchmarking, we generated $100,000$ randomly ordered stacks for each dataset and averaged their $cost_{chart}$ values. We then computed the optimised orderings using `BestFirst`, the combination of `BestFirst` and `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`.

3 Results

The results are documented in Table 1. In terms of quality and speed, we can observe that all visible trends persist for both, flatness and straightness. We can also observe that our algorithm `UpwardsOpt` produces better, but slower outputs than `BestFirst+TwoOpt` throughout all datasets. The use of the greedy `BestFirst`

dataset	n	m	relative costs, flatness			relative costs, straightness			times (secs), flatness		times (secs), straightness	
			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

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 and no bumps, respectively.



(a) Flatness

(b) Straightness

Figure 1: Relative layout costs from Table 1. Lower is better.

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*). All quality scores are also shown in Figure 1.

Quality-wise, `UpwardsOpt` performs only slightly better than `BestFirst+TwoOpt` for datasets with only few time series (e.g., for *unempl* or *sandy*), as well as for rather similar time series that do not exhibit much individual traits (e.g., for *liquor*). In both cases, the search space is simply not as large that both algorithms can find much different solutions – either because there are only a few time series to reorder in the first place, or because there are only few possible reorderings that would have an effect on the outcome. For certain datasets (e.g., for *messages*), `UpwardsOpt` improves significantly over `BestFirst+TwoOpt`. This is due to the characteristics of datasets like *messages*, which have a mix of longer and shorter layers. `BestFirst` will pick the shorter layers first, because they barely increase the overall cost. But in the end, only longer layers are left and will be placed on top of the shorter ones, creating a far from optimal starting point for `TwoOpt` In none of the cases, `UpwardsOpt` could do better than halving the average random $costs_{chart}$, but even bringing it down to 50% is still a significant improvement as $costs_{chart} = 0$ is usually not attainable.

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 does not disturb the entire chart, but only a small part of it, so that fewer iterations of `UpwardsOpt` are needed to reach the given 1% threshold of minimum improvement.