MinMaxLTTB: Leveraging MinMax-Preselection to Scale LTTB

Jeroen Van Der Donckt [*†] Jonas Van Der Donckt [*] Michael Rademaker Sofie Van Hoecke

IDLab, Ghent University - imec, Belgium

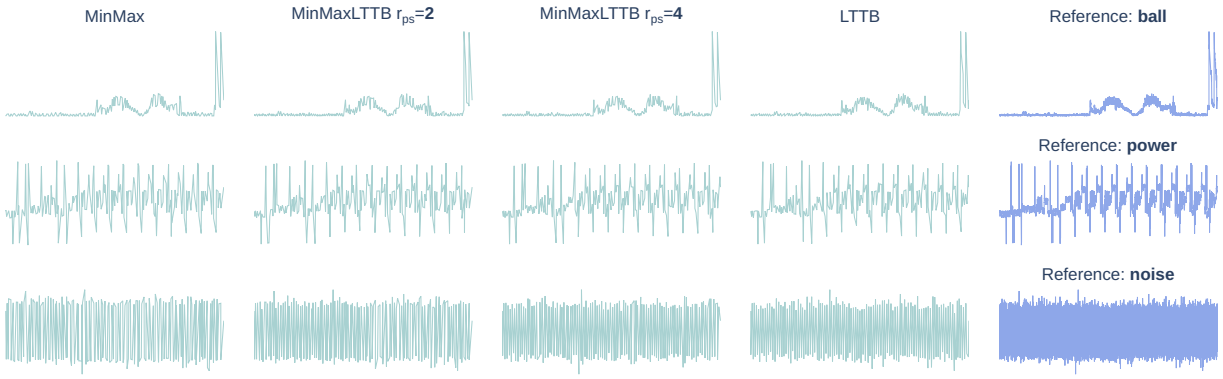

Figure 1: Visual analysis of MinMax (column 1), `MinMaxLTTB` with preselection ratio $r_{ps} \in \{2,4\}$ (column 2-3), and LTTB (column 4). Each row corresponds to a unique template, with the rightmost column presenting the reference images, constructed from 200k data points. The four adjacent aggregations utilize an equal number of output samples ($n_{out} = 200$). Columns 2 and 3 illustrate the effect of the preselection ratio r_{ps} for MinMaxLTTB. Visualization properties used include: Plotly as toolkit, line-width of 2 pixels, and anti-aliasing enabled. A GIF further demonstrates the above visualization for varying values for n_{out}.

Abstract

Visualization plays an important role in the analysis and exploration of time series data. To facilitate efficient visualization of large datasets, downsampling has emerged as a well-established approach. This work concentrates on LTTB (Largest-Triangle-Three-Buckets), a widely adopted downsampling algorithm for time series data point selection. Specifically, we introduce MinMaxLTTB, a two-step algorithm that significantly improves the scalability of LTTB. MinMaxLTTB consists of the following two steps: (i) the MinMax algorithm preselects a certain ratio of minimum and maximum data points, followed by (ii) applying the LTTB algorithm on only these preselected data points, effectively reducing LTTB's time complexity. The MinMax algorithm is computationally efficient and can be parallelized, enabling efficient data point preselection. Additionally, MinMax demonstrates competitive performance in terms of visual representation, making it also an effective data reduction method. Experimental results demonstrate that MinMaxLTTB outperforms LTTB by more than an order of magnitude in terms of computation time. Furthermore, preselecting a small multiple of the desired output size already yields similar visual representativeness compared to LTTB. In summary, MinMaxLTTB leverages the computational efficiency of MinMax to scale LTTB, without compromising on LTTB its favorable visualization properties. The code and experiments associated with this paper can be found at `https://github.com/predict-idlab/MinMaxLTTB`.

Index Terms: Time series—Line charts—Downsampling algorithms—MinMax– LTTB—Computational efficiency—Perception—Preselection– Evaluation

[*]contributed equally

[†]e-mail: (firstname)(dot)(lastname)(at)ugent(dot)be

1 Introduction

Visualization is widely recognized as a powerful tool for analyzing and exploring time series data, with line charts proving particularly effective for most tasks [2]. As the volume of time series data continues to expand, there is an increasing need for efficient visualization methods capable of handling large datasets [4, 10, 19]. To address this challenge, downsampling has emerged as a well-established technique that involves either aggregating or selecting a representative subset of the time series [1, 2, 16]. By reducing the number of data points while preserving the overall shape of the time series, downsampling minimizes network latency and improves rendering time, making it a vital component in numerous widely adopted time series databases [6, 20].

This work specifically focuses on the LTTB (Largest-Triangle-Three-Buckets) algorithm [23], which is a value preserving aggregation method as it downsamples by selecting data points from the original time series [13, 14]. LTTB has gained widespread adoption in industry, with companies like Uber incorporating it as a downsampling function in their M3 metrics platform [20] and TimeScaleDB offering LTTB as a server-side hyperfunction [6].

Despite its broad use, LTTB exhibits certain computational limitations that restrict its applicability to massive datasets containing billions of data points. Specifically, LTTB involves expensive operations to compute triangular surfaces for each data point and requires a sequential pass over the data, making it unsuitable for parallelization. To overcome these computational challenges, we introduce *MinMaxLTTB*, a two-step approach that (i) employs MinMax-preselection as an initial data reduction step, and (ii) applies LTTB on the preselected data points. In particular, this approach leverages the computational efficiency of the MinMax algorithm to make LTTB more scalable.

Utilizing the visual evaluation framework that we proposed in previous work [26], we evaluate the visual representativeness of the proposed MinMaxLTTB algorithm for various MinMax-preselection ratios. These findings provide empirical evidence for choosing an appropriate MinMax-preselection ratio. Furthermore, we assess the performance improvement of MinMaxLTTB over LTTB. Notably,

the presented technique is the default downsampling approach in our open-source time series visualization tool plotly-resampler [25], which has at the time of writing almost 2 million installations.

2 Related Work

This section provides an overview of related work in the field, focusing on the scalability of value preserving data aggregation algorithms for time series line chart visualization.

2.1 Time Series Visualization

Visualization is crucial for exploring time series data, with the human eye frequently being advocated as the ultimate data mining instrument [17]. For the majority of time series analysis tasks, simple visualizations, such as line charts, have proven most effective [2]. Interactive visualization techniques, as emphasized by Shneiderman's visual information-seeking mantra [22], are essential for exploring large time series data by providing an overview, enabling zooming and filtering, and allowing access to details-on-demand [29].

Most interactive approaches render visualizations client-side, often in web-based environments [5, 19]. However, this approach presents two significant challenges when handling large data volumes: (i) considerable network latency due to the transmission of large data volumes, and (ii) poor client-side rendering performance [1]. Both aspects limit responsiveness and interactivity, which, as highlighted above, is crucial for effectively exploring time series data. To overcome these limitations, data aggregation has proven to be an effective approach [4, 15, 16].

2.2 Data Aggregation for Time Series Visualization

Data aggregation for time series visualization can be categorized into density-wise data aggregation and downsampling. Density-wise data aggregation employs a shared color-coding to generate an image of the data on the server-side, which is transmitted to and displayed on the client front-end [12]. Downsampling reduces the number of data points that are transmitted to the visualization front-end while aiming to preserve specific characteristics or the overall shape of the data. Consequently, this approach results in reduced latency, enhanced client-side rendering time, and increased responsiveness, as the application is not burdened with rendering all data points [1].

Downsampling can be further differentiated into characteristic and value preserving data aggregation. Characteristic data aggregation aims to emphasize specific properties or trends in the data by employing aggregation operations such as mean, median, or smoothing [11, 21]. Conversely, value preserving data aggregation, also referred to as data point selection, selects data points from the original time series with the objective of preserving its overall shape.

2.2.1 Value Preserving Data Aggregation

Numerous value preserving data aggregation algorithms have been proposed in literature [3, 8]. Among these, EveryNth, MinMax, M4 [13], and LTTB [23] are arguably the most prevalent algorithms [6, 20, 26].

Table 1 presents an overview of the computational properties of these four algorithms. The EveryNth algorithm selects every n^{th} data point in order to construct an output of n_{out} data points, which is an inexpensive and common query operation [14]. MinMax downsampling involves selecting the vertical extrema for each bin (i.e., bucket B_i), using the (arg)min and (arg)max operations, which are also highly optimized queries [24]. M4 can be viewed as a combination of EveryNth and MinMax, essentially selecting the vertical (min and max) and horizontal (first and last) extrema for each bucket [13]. Given the highly optimized server-side query operations that these three algorithms capitalize on and the relatively low cost of the operations involved (selecting or comparing data), improving the computational properties of these three algorithms is practically infeasible. It is important to note that in-memory approaches also benefit from the low computational cost and the parallelizability of these algorithms.

Table 1: Overview of time series data point selection algorithms, where N denotes the time series length, and n_{out} represents the aggregation output size. We included the MinMaxLTTB algorithm for comparison.

	(i) Time	(ii) Memory	(iii) Parallelizable	(iv) Output Memory
EveryNth	$O(n_{out})$	$O(1)$	✓	$O(n_{out})$
MinMax	$O(N)$	$O(1)$	✓	$O(n_{out})$
M4 [13]	$O(N)$	$O(1)$	✓	$O(n_{out})$
LTTB [23]	$O(N)$	$O(1)$	×	$O(n_{out})$
MinMaxLTTB	$O(N)$	$O(n_{out})$	✓	$O(n_{out})$

LTTB is based on the concept of effective triangular areas, which is often employed in line simplification algorithms [23]. Specifically, LTTB selects in each bucket B_i the data point that forms the largest triangular surface with the previously selected data point and the next bucket's (B_{i+1}) average value. In contrast to the above three algorithms, LTTB involves more expensive operations, including (i) computing the average for each bin, and (ii) calculating and comparing the triangular surface for each data point within a bin. Furthermore, this algorithm requires a sequential pass over the data, making parallelization impossible. Implementing LTTB in database solutions also requires a user-defined function, benefiting less from server-side query optimizations [9]. As such, LTTB is considerably more expensive in both out-of-core (i.e., query) and in-memory contexts.

Given these observations on LTTB's computational properties, the focus of this work is to improve the scalability of LTTB, striving towards the same computational properties as M4 and MinMax.

3 MinMaxLTTB

We propose MinMaxLTTB, an enhancement to LTTB that addresses its unfavorable computational properties. We refer the reader to `github.com/predict-idlab/tsdownsample` [27] for a concrete implementation of MinMaxLTTB[1]. This is achieved by building on insights from prior research, which indicated the importance of selecting (alternating) vertical extrema for representative data aggregation [13, 23, 26]. MinMaxLTTB is realized in a two-step process, where we (i) *preselect* vertical extrema by using MinMax, and then (ii) *apply* LTTB on the preselected data points. In addition to the MinMax-preselection, the first and last data point of the original time series are also passed to the LTTB algorithm.

In the first step, we preselect $r_{ps} \cdot n_{out}$ data points, where $r_{ps} \geq 2$ denotes the integer preselection ratio and n_{out} the number of output data points. Remark that $r_{ps}/2$ indicates the number of LTTB *sub-buckets* for which vertical extrema (i.e., min and max) will be selected, e.g., an r_{ps} of 8 can be interpreted as dividing each LTTB bucket into 4 sub-buckets [2].

In the second step, the LTTB algorithm is applied on the $r_{ps} \cdot n_{out}$ data points, allowing LTTB to scale with n_{out} instead of the time series length N. Although the MinMax algorithm scales with N, it is relatively inexpensive and easily parallelizable, facilitating efficient data reduction. Furthermore, since extracting the (arg)min and (arg)max is a common query, this technique can be seamlessly integrated into database solutions where it benefits from server-side query optimizations such as caching [9].

[1] This implementation is used in our performance analysis, and thus also provides the parallelization capabilities.

[2] Note that MinMaxLTTB with $r_{ps} = 1$ corresponds to MinMax aggregation.

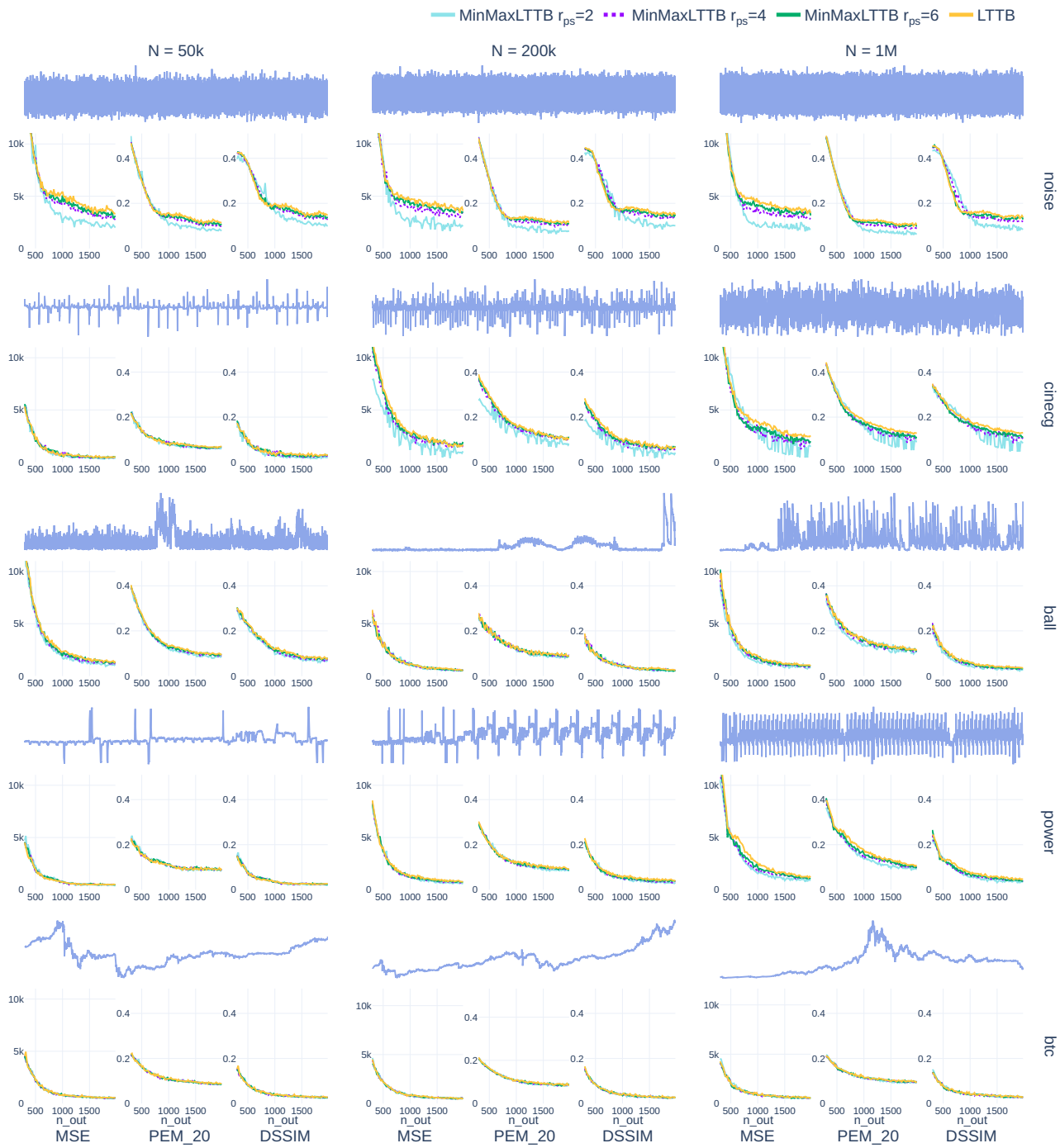

Figure 2: Assessing visual representativeness of `MinMaxLTTB` with $r_{ps} \in \{2,4,6\}$ for various time series templates. Each row displays a distinct time series dataset, with columns indicating the template size. All image templates, including the blue reference templates which are depicted above the metric subplots, were generated using Plotly's default settings (linear interpolation, line-width of 2 pixels). The metric subplots reveal trends in aggregation algorithm performance as n_{out} (x-axis) increases (range [200, 2000]). More information about these trends, metrics, conv-mask scaling, and templates can be found in [26]. PEM_20 refers to Pixel Error with a Margin of 20, where pixels differences above 20 are binarized and divided by the conv-mask size to obtain a ratio. DSSIM denotes conv-mask scaled structural dissimilarity, while MSE represents conv-mask scaled Mean Squared Error. A GIF and an HTML animation further demonstrate the visual representativeness of MinMaxLTTB for $r_{ps} \in [1-8]$.

3.1 Visual Representativeness and Preselection Ratio

We analyze the visual representativeness of MinMaxLTTB for various preselection ratios $r_{ps} \in \{2,4,6\}$ using the methodology proposed in [26]. We refer the reader to this prior work for details on the visual representativeness metrics and the selected time series templates.

A first key observation is that MinMaxLTTB's performance curves are on par with LTTB for low-roughness series such as the btc templates, power $\leq$200k, cinecg-50k, and ball-200k. In these cases, the MinMax-preselection ratio has little influence, possibly explained by the absence of prominent extrema in the templates. For higher roughness series such as the noise templates and cinecg

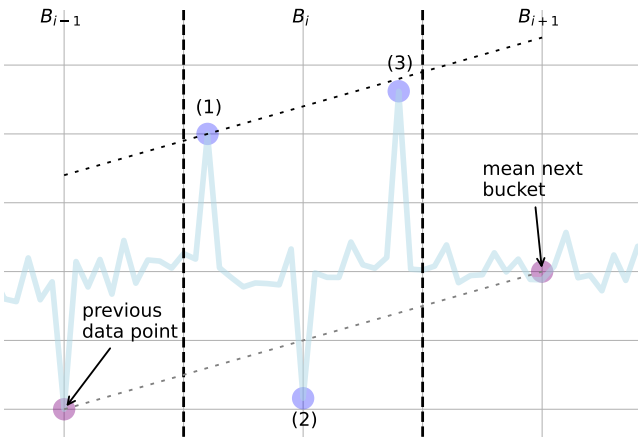

Figure 3: Illustration of LTTB's tendency to (i) select contrasting extrema values in neighboring buckets and (ii) favor extrema proximity to the left-bin edge. For all points in B_i, triangular areas are calculated using the prior selected point from B_{i-1} and the mean x and y values of B_{i+1}. Points (1) and (3), both opposing the selected extrema of B_{i-1}, compete for the largest triangular surface. Note that extremum (2) would yield a considerably smaller triangle than points (1) and (3). The dashed line passing through point (1) represents the *equisurface* line, with points above it generating larger triangles and those below resulting in smaller ones. This line will always be parallel to the one connecting the previous data point and the mean of the next bucket. Despite being the global extrema of B_i, (3) is not selected as it resides below the equisurface line of (1), illustrating LTTB's tendency to favor data points near the left-bin edge.

≥200k, MinMaxLTTB even seems to perform (slightly) better, especially for low $r_{ps} \in \{2,4\}$. This observation can be attributed to LTTB favoring the selection of data points near the left bin-edge. Figure 3 intuitively demonstrates that extrema near the left bin-edge allow creating larger triangular surfaces (with the selected extremum from the previous bucket), potentially omitting larger extrema that occur in the center or right part of the bin. Since MinMax-preselection with a low r_{ps} preselects fewer options, while only considering the vertical position, LTTB's tendency to select near left-bin edge extrema values is partly mitigated. For example, in Figure 3, using $r_{ps} = 2$, points (2) and (3) would get preselected for bucket B_i, resulting in MinMaxLTTB selecting the bin-maximum (3) instead of (1). This results in improved visual representativeness (compared to the reference template) as more prominent data points are retained.

Consequently, increasing the preselection ratio makes the MinMaxLTTB performance curves less noisy and shift more towards the LTTB curves, as MinMaxLTTB will have more near left bin-edge preselected extrema. In summary, MinMaxLTTB does not degrade in visual representativeness with respect to visual LTTB, and a low $r_{ps} \in \{4,6\}$ already results in high visual similarity to LTTB.

3.2 Performance

MinMaxLTTB exhibits the same linear time complexity as LTTB (see Table 1). The memory complexity of MinMaxLTTB is $O(n_{out})$ instead of $O(1)$ because, after the first step, $r_{ps} \cdot n_{out}$ indices are preselected (and thus stored in memory). In the previous section, it was demonstrated that a low r_{ps} is sufficient to achieve comparable visual representativeness. Therefore, the additional memory overhead associated with MinMaxLTTB is almost negligible compared to LTTB, since the memory complexity of constructing the output is $O(n_{out})$ [3].

[3]Note that this cost is unavoidable for all algorithms, and therefore included in the output memory column of Table 1.

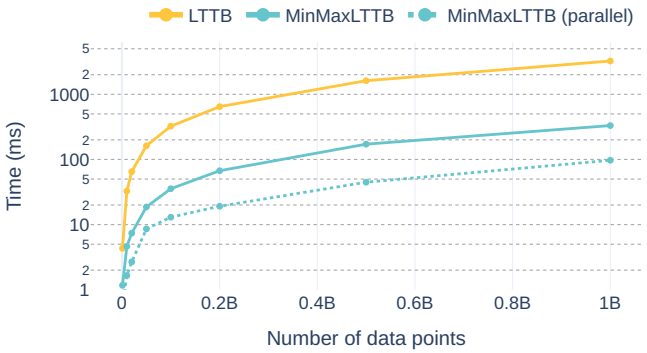

Figure 4: In-memory performance analysis of LTTB and MinMaxLTTB. For LTTB the C implementation from `plotly-resampler` v0.8.3.2 is used, for MinMaxLTTB the implementation can be found here.

Although the runtime of both algorithms scales linearly with the time series size (N), the slope of this linear scaling is significantly lower (i.e., better) for MinMaxLTTB. This can intuitively be explained by looking at the operations that are performed (for each data point) by both algorithms. In particular, both algorithms perform comparisons, with LTTB comparing for the largest triangular surface and MinMax comparing for extrema. However, in addition to a comparison, LTTB's operations include the calculation of the triangular surfaces (and the bin-wise average), thus requiring more operations than MinMax, resulting in a higher slope for LTTB. In addition to superior linear scaling, MinMaxLTTB also allows for parallelization, which further alleviates the linear scaling to multiple cores. Figure 4 demonstrates this improved scaling of MinMaxLTTB, enhancing the performance with an order of magnitude (10x). Furthermore, when applying parallelization on MinMaxLTTB, the performance increases 30x compared to LTTB. This improvement in performance was observed while utilizing 32 threads, which is the number of logical cores on the benchmarking CPU.

4 Conclusion

This work proposes MinMaxLTTB, a downsampling technique that mitigates the unfavorable computational properties of LTTB through the use of MinMax-preselection. Our evaluation of MinMaxLTTB's visual representativeness reveals that even a small preselection ratio $r_{ps} \in \{4,6\}$ yields a high degree of similarity to LTTB. As future work, a case study should further affirm these findings. Performance analysis demonstrated that MinMaxLTTB's computation time decreases by over an order of magnitude compared to LTTB. This is particularly significant for scalable visualization of big data, as interactive latency greatly influences the rate at which users make observations during exploratory analysis [18]. We further hypothesize that the demonstrated effectiveness of MinMax-preselection could potentially be extended to other computationally demanding data point selection algorithms such as Visvalingam-Whyat [28], Douglas-Peucker [7], and Longest-Line-Bucket [23]. Finally, the fact that MinMaxLTTB is currently the default aggregation algorithm for already over half a year in a widely utilized visualization tool serves as strong evidence of its efficacy.

Acknowledgments

The authors wish to thank Louise Van Calenbergh for proofreading the manuscript. Jonas Van Der Donckt (1S56322N) is funded by a doctoral fellowship of the Research Foundation Flanders (FWO). Part of this work is done in the scope of the imec.AAA Context-aware health monitoring project.

REFERENCES

[1] R. Agrawal, A. Kadadi, X. Dai, and F. Andres. Challenges and opportunities with big data visualization. In *Proceedings of the 7th International Conference on Management of computational and collective intElligence in Digital EcoSystems*, pp. 169–173. ACM, Caraguatatuba Brazil, Oct. 2015. doi: 10.1145/2857218.2857256

[2] W. Aigner, S. Miksch, W. Müller, H. Schumann, and C. Tominski. Visualizing time-oriented data—A systematic view. *Computers & Graphics*, 31(3):401–409, June 2007. ZSCC: 0000450. doi: 10.1016/j.cag.2007.01.030

[3] P. Bae, K.-W. Lim, W.-S. Jung, and Y.-B. Ko. Practical implementation of m4 for web visualization service. *Journal of Communications and Networks*, 19(4):384–391, 2017. doi: 10.1109/JCN.2017.000062

[4] N. Bikakis. Big Data Visualization Tools. *arXiv:1801.08336 [cs]*, Feb. 2018. ZSCC: 0000075 arXiv: 1801.08336.

[5] E. G. Caldarola and A. M. Rinaldi. Big data visualization tools: a survey. *Research Gate*, 2017.

[6] J. Davi Paganini. Downsampling in the database: How data locality can improve data analysis. `www.timescale.com/blog`, Feb 2023.

[7] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization*, 10(2):112–122, 1973. doi: 10.3138/FM57-6770-U75U-7727

[8] A. Gil, M. Quartulli, I. G. Olaizola, and B. Sierra. Towards smart data selection from time series using statistical methods. *IEEE Access*, 9:44390–44401, 2021. doi: 10.1109/ACCESS.2021.3066686

[9] J. Gjengset, M. Schwarzkopf, J. Behrens, L. T. Araújo, M. Ek, E. Kohler, M. F. Kaashoek, and R. T. Morris. Noria: dynamic, partially-stateful data-flow for high-performance web applications. In *OSDI*, vol. 18, pp. 213–231, 2018.

[10] P. Godfrey, J. Gryz, P. Lasek, and N. Razavi. Interactive visualization of big data. In *Beyond Databases, Architectures and Structures. Advanced Technologies for Data Mining and Knowledge Discovery: 12th International Conference, BDAS 2016, Ustroń, Poland, May 31-June 3, 2016, Proceedings 11*, pp. 3–22. Springer, 2016.

[11] J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, and P. J. Haas. Interactive data analysis: The control project. *Computer*, 32(8):51–59, 1999. doi: 10.1109/2.781635

[12] Holoviz-community. Datashader, quickly and accurately render even the largest data. `https://github.com/holoviz/datashader`.

[13] U. Jugel, Z. Jerzak, G. Hackenbroich, and V. Markl. M4: a visualization-oriented time series data aggregation. *Proceedings of the VLDB Endowment*, 7(10):797–808, June 2014. doi: 10.14778/2732951.2732953

[14] U. Jugel, Z. Jerzak, G. Hackenbroich, and V. Markl. VDDA: automatic visualization-driven data aggregation in relational databases. *The VLDB Journal*, 25:53–77, Feb. 2016. doi: 10.1007/s00778-015-0396-z

[15] S. Kumar, M. P. Andersen, and D. E. Culler. Mr. Plotter: Unifying Data Reduction Techniques in Storage and Visualization Systems, June 2021. arXiv:2106.12505 [cs].

[16] B. C. Kwon, J. Verma, P. J. Haas, and C. Demiralp. Sampling for Scalable Visual Analytics. *IEEE Computer Graphics and Applications*, 37(1):100–108, Jan. 2017. doi: 10.1109/MCG.2017.6

[17] J. Lin, E. Keogh, and S. Lonardi. Visualizing and Discovering Non-Trivial Patterns in Large Time Series Databases. *Information Visualization*, 4(2):61–82, June 2005. ZSCC: 0000178. doi: 10.1057/palgrave.ivs.9500089

[18] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *IEEE transactions on visualization and computer graphics*, 20(12):2122–2131, 2014.

[19] R. Netek, J. Brus, and O. Tomecka. Performance testing on marker clustering and heatmap visualization techniques: A comparative study on javascript mapping libraries. *ISPRS International Journal of Geo-Information*, 8(8), 2019. doi: 10.3390/ijgi8080348

[20] B. Raskin and N. Aggarwal. The billion data point challenge: Building a query engine for high cardinality time series data. `uber.com/billion-data-point-challenge`, Dec 2018.

[21] K. Rong and P. Bailis. ASAP: prioritizing attention via time series smoothing. *Proceedings of the VLDB Endowment*, 10(11):1358–1369, Aug. 2017. doi: 10.14778/3137628.3137645

[22] B. Shneiderman. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *Proceedings 1996 IEEE symposium on visual languages*, pp. 336–343. IEEE, 1996. doi: 10.1016/B978-155860915-0/50046-9

[23] S. Steinarsson. Downsampling Time Series for Visual Representation. Master's thesis, University of Iceland, 2013. doi: 1946/15343

[24] K. Tangwongsan, M. Hirzel, S. Schneider, and K.-L. Wu. General incremental sliding-window aggregation. *Proceedings of the VLDB Endowment*, 8(7):702–713, 2015.

[25] J. Van Der Donckt, J. Van Der Donckt, E. Deprost, and S. Van Hoecke. Plotly-resampler: Effective visual analytics for large time series. In *2022 IEEE Visualization and Visual Analytics (VIS)*, pp. 21–25. IEEE, 2022. doi: 10.1109/VIS54862.2022.00013

[26] J. Van Der Donckt, J. Van Der Donckt, M. Rademaker, and S. Van Hoecke. Data point selection for line chart visualization: Methodological assessment and evidence-based guidelines. *arXiv preprint arXiv:2304.00900*, 2023.

[27] J. Van Der Donckt, J. Van Der Donckt, and S. Van Hoecke. tsdownsample: high-performance time series downsampling for scalable visualization. *arXiv preprint arXiv:2307.05389*, 2023.

[28] M. Visvalingam and J. D. Whyatt. Line generalisation by repeated elimination of points. *The cartographic journal*, 30(1):46–51, 1993. doi: 10.1179/000870493786962263

[29] J. Walker, R. Borgo, and M. W. Jones. TimeNotes: A Study on Effective Chart Visualization and Interaction Techniques for Time-Series Data. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):549–558, Jan. 2016. doi: 10.1109/TVCG.2015.2467751