

A SUPPLEMENTARY

A.1 Visualization Efficacy Comparison With More Baselines

A.1.1 Compare with PLA. One simple difference between our work and PLA [b] is that we use joint segments of sampled points for visualization while [b] uses disjoint segments for lossy compression. When disjoint segments are used for visualization, visual artifacts are produced from the connected lines of disjoint ends.

In essence, we target a different goal from PLA. PLA [b] uses RMSE to measure data fidelity, i.e., reconstruction accuracy in the data space. We use SSIM [19, 25, 35] to measure visual fidelity, i.e., pixel-level or ideally perceptual-level, similarity in the image space. We provide in Section A.3 a detailed formula for the widely used SSIM metric, which accounts for variations in luminance, contrast, and structural components that are valued by the "human visual system". The fundamental difference between these two metrics is that RMSE operates in the t-v data space, while SSIM operates in the x-y image space. The two spaces have different resolutions and scales. In terms of resolution, certain discrepancies in the data domain may not be visible at pixel resolution, and on the other hand, tolerable errors in the data domain may be visually apparent. In terms of scales, varying scales for the time and value axes are used in line charts [A]. Usually, a small unit on the time axis represents a larger range or duration than the same unit on the value axis. Hence visible pixel errors along the value dimension may be underestimated by the data-space metrics. Our empirical experiments in Section A.1.3 confirm these differences.

A.1.2 Compare with Cartographic Simplification. Our goal is closer to that of cartographic simplification [f], because we all recognize humans as the ultimate consumers of approximation. Hence we prioritize understanding how humans perceive images. For example, Attneave [13][f] identified high curvature points as perceptually important. In contrast, dimensionality reduction techniques for similarity search often focus on lower bounding the Euclidean distance, potentially sacrificing visually significant points due to a smoothing effect [B]. However, there are still differences between our goal and that of [f]. While [f] is in the field of general cartographic simplification, our work focuses more on the specific domain of time series line charts, where different scales for the time and value axes need to be considered.

A.1.3 Empirical Comparison with Sim-Piece, reuwi, and visval. We add three more methods into comparison, which are Sim-Piece [e], reuwi [29], and visval [33]. Sim-Piece is one of the most recent works on Piecewise Linear Approximation (PLA). It exploits the similarities of PLA segments to achieve compact representations in the field of lossy compression. Reuwi and visval are two classical methods in the field of cartographic simplification. Reuwi uses a point-to-line perpendicular distance threshold in the data space to sequentially segment time series. Visval proposes a bottom-up method to generalize lines by repeatedly eliminating the point with the smallest triangle area.

Figure 15 shows the SSIM comparison results. The visualizations of the sample data under $m = 320$ is shown in Figure 16. The first row, colored in green, presents the line charts of the original time

series from four datasets. Each subsequent row represents a different sampling method, with the SSIM metric displayed above to evaluate its similarity to the original line chart, considered as the ground truth. Within each column (i.e., for each dataset), the top four methods with the highest SSIM values are highlighted using blue solid-line rectangles. Additionally, visual defects of different sampling algorithms are emphasized with red dashed-line rectangles.

For Sim-Piece, it generates visual artifacts that do not exist in the data but are produced by connecting lines with disjoint ends. Besides, some regions with fewer segments succeed in keeping data fidelity, but fails to maintain visual fidelity (i.e. looking sparse). For reuwi, it misses many vertical spikes because pixel errors in the value direction is underestimated by the data-space perpendicular distance metric. For visval, it does not have the disadvantage of the previous two methods that an area is sparsely represented by a segment, but it has the disadvantage of missing sharp spikes due to their small triangle areas. In contrast, our proposed ILTS consistently ranks among the top four in terms of visual fidelity.

A.1.4 Empirical Comparison with SWAB. We further compare with a classic PLA method called SWAB [C]. The visualization results of SWAB are reported in Figure 16. For the specific settings of SWAB, we use the linear interpolation (joint) to find the approximating line and the residual error (i.e., sum of squares error) to measure the fit for each segment. We experimented with other settings, such as the linear regression (disjoint) and furthest point error, which did not yield the best results and were thus omitted.

First, SWAB shows clearly better visualization quality in terms of SSIM than Sim-Piece, in all the four datasets in Figure 16. This result is in line with [D] on the comparison between SWAB and Sim-Piece.

Indeed, SWAB has the highest SSIM in datasets (b) Lightning7 and (d) Wine, i.e., the second and fourth columns in Figure 16, where our ILTS is comparable. As indicated by the red boxes in Figure 16(d), ILTS does not always present the extremely large or small values, whereas SWAB may miss some details in the middle.

However, for the datasets (a) HouseTwenty and (c) Mallat, i.e., the first and third columns in Figure 16, SWAB is not as accurate as the proposed ILTS. As shown, SWAB again may miss some details of fluctuations in the middle.

The reason is that PLA methods focus on clustering (or segmenting) points such that a local line segment can approximate them within a maximum error threshold. Note that the maximum error threshold is imposed globally. Therefore, SWAB presents better the extremely large or small values than the proposed ILTS. However, SWAB may ignore the details in the middle, which have relatively small errors, e.g., in Figure 16(d) of dataset Wine. Such small errors may be approximated in one segment, thus missing the detailed fluctuations.

[b] Themis Palpanas, Time Series Summarization Using User-Defined Amnesic Functions. IEEE Trans. Knowl. Data Eng. 20(7): 992-1006 (2008)

[e] Kitsios, Panagiotis Liakos, Sim-Piece: Highly Accurate Piecewise Linear Approximation through Similar Segment Merging. Proc. VLDB Endow. 16(8): 1910-1922 (2023)

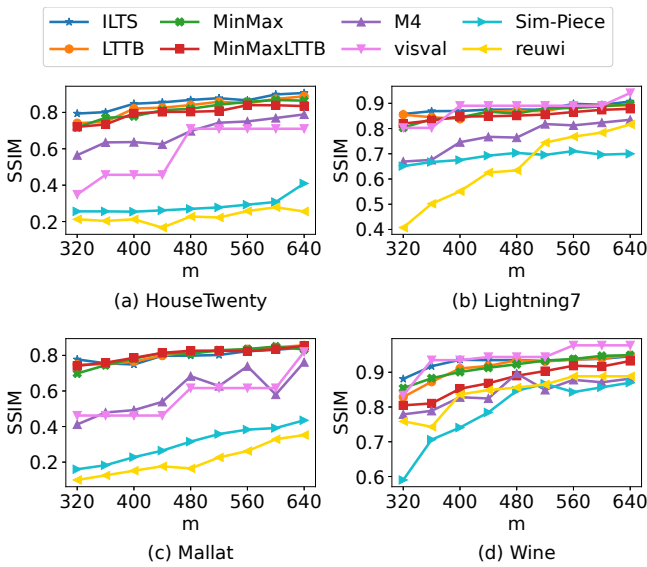


Figure 15: SSIM comparison experiments (Sim-Piece, reuwi, and visual baselines added)

[f] The awakening of Attnave's sleeping cat: identification of everyday objects on the basis of straight-line versions of outlines Joeri De Winter 1, Johan Wagemans

[A] Anshul Vikram Pandey, Katharina Rall, Margaret L. Satterthwaite, Oded Nov, and Enrico Bertini. 2015. How Deceptive are Deceptive Visualizations? An Empirical Analysis of Common Distortion Techniques. In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15). Association for Computing Machinery, New York, NY, USA, 1469–1478. <https://doi.org/10.1145/2702123.2702608>

[B] Tak-chung Fu, Fu-lai Chung, Robert Luk, and Chak-man Ng. 2008. Representing financial time series based on data point importance. Eng. Appl. Artif. Intell. 21, 2 (March, 2008), 277–300. <https://doi.org/10.1016/j.engappai.2007.04.009>

[C] Chu, S., Hart, D. & Pazzani, M. (2001). An Online Algorithm for Segmenting Time Series. In Proceedings of IEEE International Conference on Data Mining. pp 289

[D] https://www.linkedin.com/feed/update/urn:li:activity:7100360234749095937?updateEntityUrn=urn%3A%3Afs_feedUpdate%3A%28V2%2Curn%3A%3Aactivity%3A7100360234749095937%29

A.2 System Architecture and Response Time Decomposition

This section illustrates our motivation to expedite in-database sampling queries.

A.2.1 Data Visualization System. A typical data visualization system is shown in Figure 17. Large-scale time series data are stored in the remote database. Note that time series data are continually generated by (often millions of) sensors and sent to the database server. The recent data of a time series (from one sensor) are first buffered in memory. When the memory buffer of this time series is full, it is flushed/stored to the disk, as a disk page. Then, the next

segment of this time series is buffered, leading to the next disk page, and so on and so forth.

When a user requests an ad-hoc visualization of a time series, the visualization client sends an online query request to the database. The query executor of the database deals with metadata and data and sends the result data back to the client. Finally, the client renders the data on the screen.

A.2.2 Response Time Decomposition. The big data poses challenges to every aspect of the visualization system, including database querying, communicating, and rendering [21]. We experiment on the response time decomposition and the results are shown in Figure 18.

Rendering. The client rendering time refers to the duration taken by Python to execute `plt.plot(t,v)` function. It turns out that Python requires around 20 seconds to plot 300 million data points. Moreover, in our practical experience, Grafana dashboard, as another widely used visualization tool, encounters out-of-memory errors when attempting to visualize time series data exceeding 100 million points.

Querying. While in-memory similarity search queries [a] have demonstrated remarkable speed, we focus on queries within disk-based databases as illustrated in Figure 17. Firstly, the underlying physical storage of data challenges fast querying. Note that high concurrency data writes are common in IoT scenarios, where thousands of devices with millions of sensors continuously ingest data into the database. Buffered time series data are periodically flushed to disk once the memory threshold is reached. For instance, as shown in Figure 17, the time series "A" marked in green is stored as pages across dispersed disk locations within a file. While compaction strategies are used to periodically merge small pages into larger ones, the dispersed I/O remains unavoidable and substantially impact the query performance. Secondly, the upper-layer encapsulation step of the query result set is also time-consuming for large data volumes. Consequently, executing raw data queries in such disk-based databases is time consuming. In our experiments, the execution time of the raw data query for 300 million points reached 200 seconds.

Communicating. Communication is also challenged by big data. In our experiments, transferring 300 million data points from the remote database to the visualization client takes seven minutes. Moreover, when multiple clients are connected to the database for data transmission simultaneously, the number of clients the server can serve concurrently is limited due to bandwidth constraints [25].

A.2.3 Our Motivation for Query Efficiency. Recently Plotly-resampler [32] proposes applying sampling in the frontend plotting software to address the slow rendering issues of big data. However, considering the large communication delays for big data transferred from the remote database, we believe that in-database sampling is essential. As shown in Figure 18, sampling queries such as LTTB can be employed instead of raw data queries to reduce data transfer and rendering time. Consequently, database query execution time becomes the bottleneck. Hence our work contributes to accelerating such in-database online sampling queries to further decrease the response time of visualization applications [28]. Specifically, we

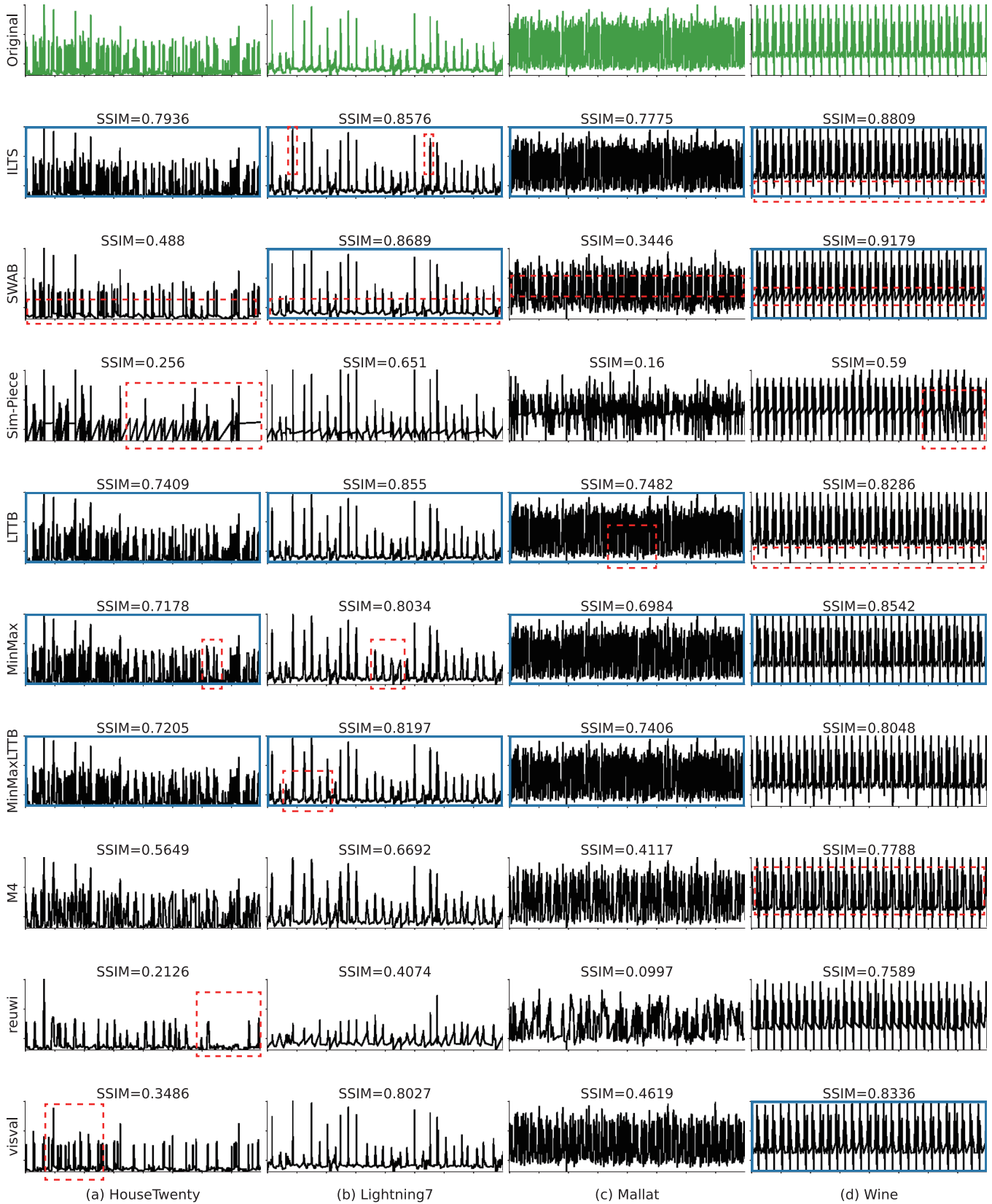


Figure 16: Visualizations of sample data. Blue solid-line rectangles highlight the top four methods with the highest SSIM values within each column. Red dashed-line rectangles emphasize the visual defects of different sampling algorithms.

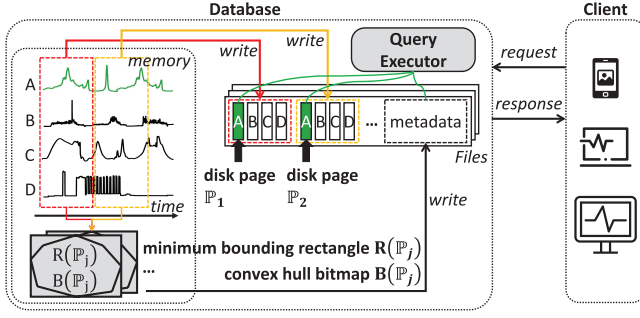


Figure 17: Architecture of a data visualization system with a disk-based remote database

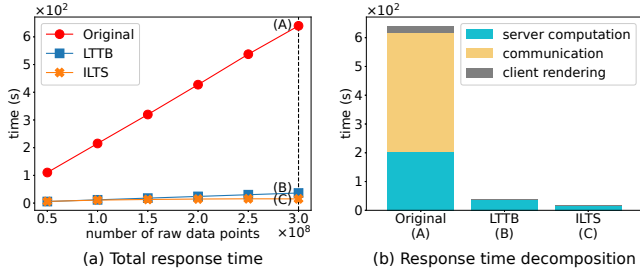


Figure 18: Cost of visualizing time series from a remote database

use precomputed convex hulls to accelerate online largest triangle sampling queries. Note that while a time series is stored as pages across various disk locations within a file, the metadata for these pages is stored contiguously at the end of the file, which is I/O friendly.

[a] <https://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html>

A.3 SSIM Formula

Let X and Y denote the pixel matrices of two images with the same dimensions $w \times h$. A simple error measure of the two images is the Mean Squared Error, i.e., $MSE(X, Y) = \frac{1}{wh} \sum_{i=1}^w \sum_{j=1}^h (X_{ij} - Y_{ij})^2$. The Structural SIMilarity (SSIM) index [19, 25, 35] is further proposed to account for variations in luminance, contrast, and structural components that are valued by human perception. Formally,

$$SSIM(X, Y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)},$$

where μ_x, μ_y are the average intensities for luminance comparison, σ_x^2, σ_y^2 are the variances for contrast comparison, σ_{xy} is the covariance of X and Y for structure comparison, and C_1, C_2 are small constants to stabilize division operations. SSIM yields a score within the range of -1 to 1, where values closer to 1 indicate higher visual similarity.

In practice, SSIM is computed within a local window (e.g., a 11×11 square window) that slides across the entire image. The overall image quality is determined by averaging these SSIM values obtained from all windows, a method known as Mean SSIM (MSSIM) [35]. Considering the prevalent white background in time

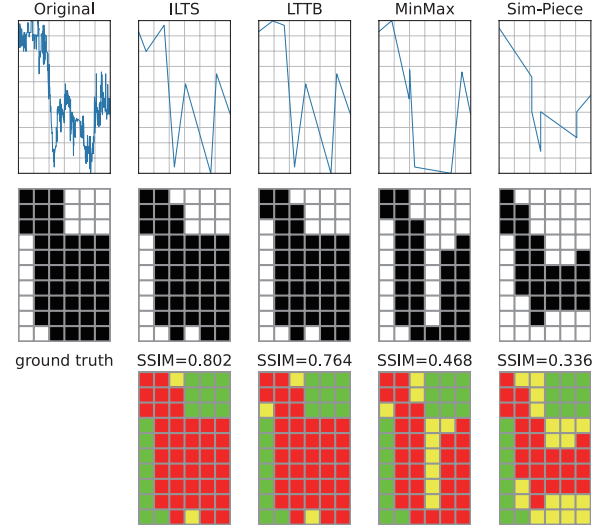


Figure 19: Example of SSIM on rasterized line charts

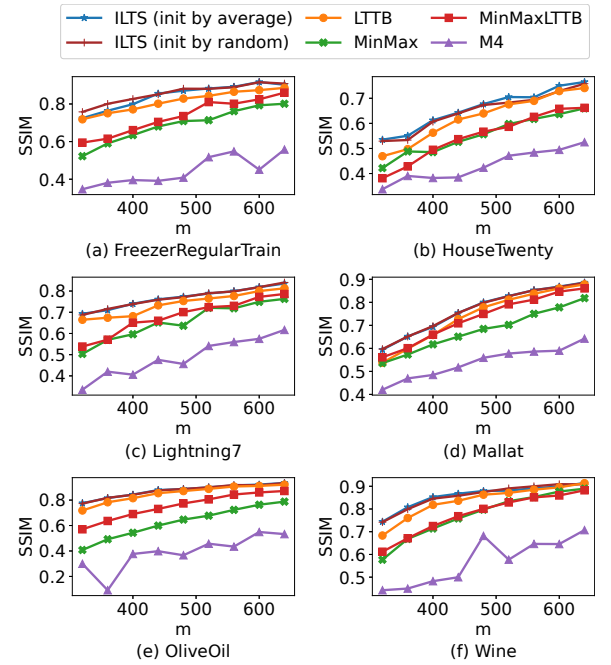


Figure 20: SSIM comparison experiments (adding ILTS initialized by random points)

series line charts, the OR-conv masking technique for SSIM [19] is introduced to effectively exclude the shared white background from comparison, enhancing the discriminability of the metric. In our experiments, we evaluate the visualization efficacy of sampling algorithms by calculating the masked SSIM between the original and the sampled line charts. The larger the SSIM value, the better the visual fidelity of the sampling algorithm.

Take Figure 19 as an example. Suppose that we sample eight points (i.e., $m = 8$) from an input time series of length $n = 600$ and

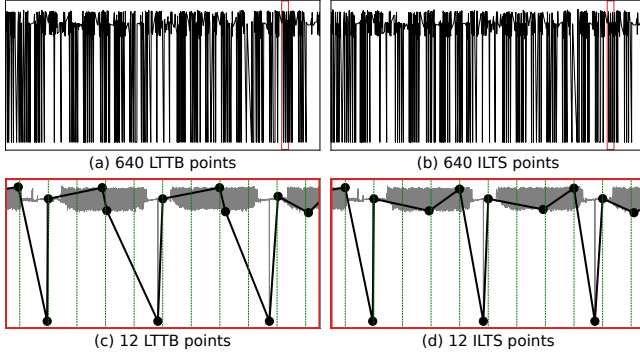


Figure 21: LTTB v.s. ILTS. The gray lines in (c) and (d) represent the original time series. ILTS visually captures the dense area more effectively through iterations.

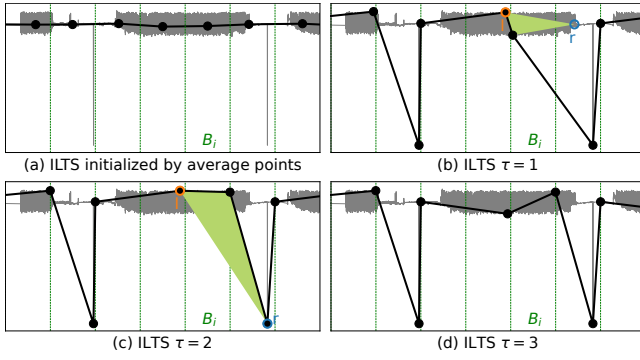


Figure 22: ILTS iteration example (continue Figure 21)

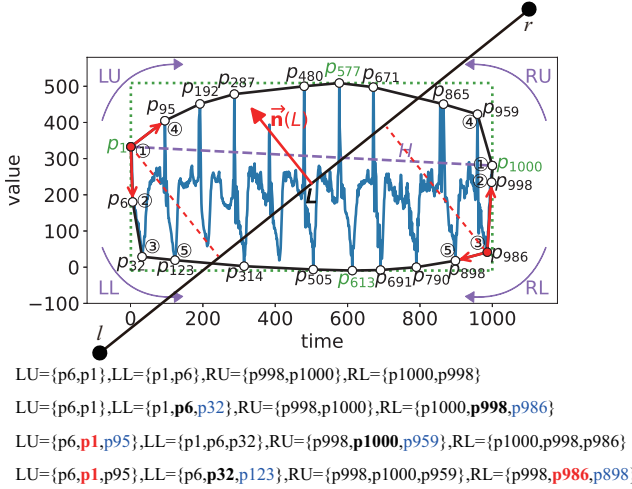


Figure 23: Improved example of the convex hull acceleration (for better Figure 4)

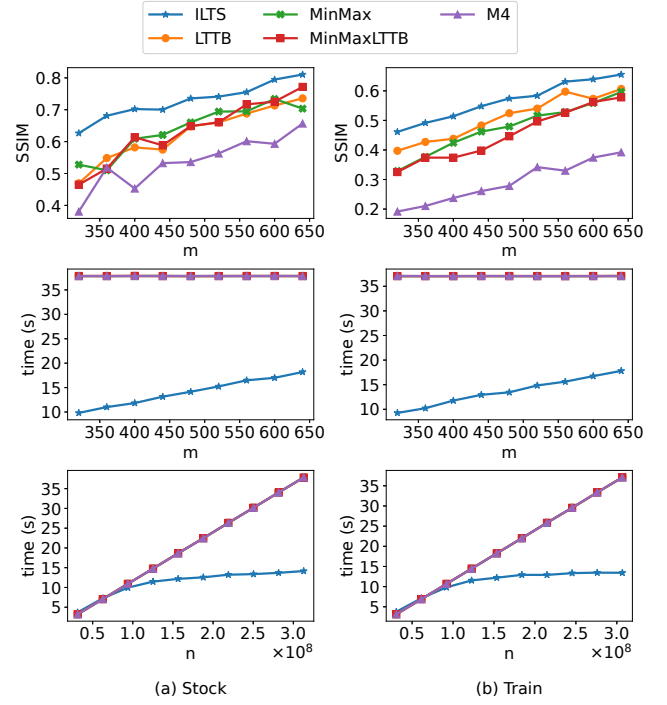


Figure 24: Experiments on two additional datasets

render the line chart on a 6×10 canvas. The first column represents the original time series, and each subsequent column corresponds to a different sampling algorithm. The first row displays the time series in high resolution, and the second row shows the rendered line charts. The bottom row uses different colors to encode the pixel differences between the original (labeled as "ground truth") and sampled line charts. Green pixels indicate those unmarked in both images, red pixels indicate those marked in both images, and yellow pixels denote those marked in the original but missed in the sampled image. The SSIM results show that our proposed ILTS achieves the best visual fidelity.

A.4 Proof of Proposition 1

According to [31], the Largest-Triangle-Three-Buckets (LTTB) algorithm goes through buckets B_i from left to right (i.e., from $i = 2$ to $i = m - 1$) and works at three bucket at a time. It employs the selected point from the previous bucket and the average point of the next bucket as heuristic points. The algorithm then selects the point in the current bucket that forms the largest triangle area with these two heuristic points. It is important to note that LTTB does not directly output the average point of each bucket as the final sampled result. If the Iterative Largest Triangle Sampling (ILTS) algorithm is initiated using average points and iterated one step further, it will identify an identical set of points as LTTB, since both algorithms utilize the same heuristic points for selection from the same set of points. Therefore, we prove that LTTB is a special case of ILTS.

A.5 Time Complexity Analysis

Figure 17 gives an overview of the data write and read process. When the in-memory time series data are flushed to dispersed disk locations as pages, the metadata of these pages are retained in memory because metadata is much smaller than data. When a file is to be closed, the metadata are then written sequentially to the end of the file. During querying processing, the query executor first loads all metadata into memory and the minimum bounding rectangle (MBR) in the metadata is utilized to prune unnecessary pages (Section 4.3). For pages unpruned, I/Os are inevitable; however, we can leverage the convex hull bitmap metadata to fast search on the convex hull points, thereby reducing CPU costs (Section 4.2). The query cost can be modelled as follows.

$$T = T_M + \sum_{i=1}^c I_i * (T_D + k * T_P),$$

where

- T_M : the time cost of loading all relevant metadata into memory
- c : the total number of pages for the time series
- I_i : 0 if the i -th page is pruned by the MBR, 1 otherwise
- T_D : the time cost of loading one disk page into memory
- T_P : the time cost of traversing all points in a page
- k : a coefficient between $[0,1]$, reflecting the proportion of searched convex hull points to the total number of points in the page

A.6 Avoid Redundant Computation

Algorithm 3 enhances Algorithm 1 by adding the redundancy removal logic as introduced in Section 3.2. Specifically, Line 7 shows the condition under which a bucket can reuse the sampling point from the previous iteration. $leftEq = true$ and $R_{i+1} = true$ means that its left and right adjacent latest sampled points remain unchanged in their last two iterations, respectively. Line 16 indicates the condition for early termination of the iteration, which occurs when all sampled points remain unchanged in their last two iterations.

A.7 Visualization Efficacy Comparison Using Other Initialization Mechanism

Referring to line 2 of Algorithm 1, the proposed ILTS can initialize by either random or average points in the buckets. We experiment on ILTS initialized by either average or random points in Figure 20. As can be seen, ILTS works well under both average and random initialization.

A.8 Better Figure 1

To better illustrate the motivation and superiority of our proposal in terms of visualization quality, we will replace the previous Figure 1 by the new Figure 21, and reserve the original Figure 1 solely as an illustrative example to explain the definition and steps of sampling methods.

Figure 21 shows the line charts plotted by the existing LTTB in (a) and our ILTS in (b). To illustrate the inferiority, we zoom in the red boxes in (c) and (d), respectively. As shown in Figure 21(c), the existing method LTTB, line-dot in black, fails to cover the area of

Algorithm 3: Iterative largest triangle sampling (augmented with redundancy removal)

Input: an input time series $T = \{p_1, \dots, p_n\}$, a threshold m for the length of the sampled time series, the number of iterations τ

Output: a sampled time series $ILTS(T) = \{q_1^*, \dots, q_m^*\}$

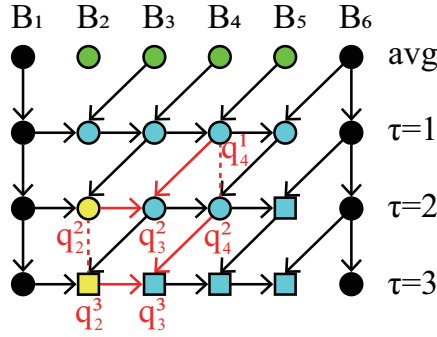
- 1 determine all time buckets B_i , $i = 1, \dots, m$, as in Definition 2
- 2 initialize q_i^* by either a random sample or the average of all points in the bucket B_i , $i = 1, \dots, m$
- 3 initialize the flag array R with the first $m - 1$ elements being *false* and the last element being *true*
- 4 **for** $k \in \{1, \dots, \tau\}$ **do**
- 5 initialize the flags $allEq = true$, $leftEq = true$
- 6 **for** $i \in \{2, \dots, m - 1\}$ **do**
- 7 **if** $leftEq$ **and** R_{i+1} **then**
- 8 set $R_i = true$
- 9 continue
- 10 set $l = q_{i-1}^*$, $r = q_{i+1}^*$, $q = q_i^*$
- 11 set $q_i^* = \arg \max_{p \in T, p.t \in B_i} \Delta lpr$ by the largest triangle routine as in Definition 3, with convex hull acceleration
- 12 **if** $q_i^* = q$ **then**
- 13 set $leftEq = true$, $R_i = true$
- 14 **else**
- 15 set $allEq = false$, $leftEq = false$, $R_i = false$
- 16 **if** $allEq$ **then**
- 17 break
- 18 **return** $\{q_1^*, \dots, q_m^*\}$

ground truth, lines in gray. In contrast, our proposed ILTS in (d) more accurately covers the ground truth.

A.9 Better Figure 4

We improve the original Figure 4 in Figure 23. The fixed line L is shown to connect the two heuristic points l and r . The effect of normal vector is emphasized by adding auxiliary perpendicular lines. The intermediate results of triplets LU, LL, RU, RL during fast search on the convex hull are also labeled in the figure.

Without using the normal vector $\vec{n}(L)$, we need to traverse all the convex hull vertices to find the point farthest from the line L . Instead, the normal vector $\vec{n}(L)$ provides a specific direction for this search. That is, if the vectors from a convex hull vertex to its two geometrically adjacent vertices form both obtuse (or acute) angles with the normal vector, then that vertex may be the farthest point. For example, $\vec{p_1 p_95}$ and $\vec{p_1 p_6}$ form both obtuse angles with the normal vector $\vec{n}(L)$, indicating that p_1 is the highest vertex along $\vec{n}(L)$ and thus might be the farthest point from L . For any given line L , there are at most two kinds of extreme vertices along $\vec{n}(L)$, i.e., highest and lowest. Therefore, in Figure 4, we terminate the search on the convex hull vertices once the two candidate convex hull vertices are found (i.e., the highest p_1 and the lowest p_{986} along $\vec{n}(L)$), without traversing the remaining vertices.



- q_i^k : the sampled point from B_i in the k -th iteration
- horizontal arrows: sources of left heuristic points
- diagonal arrows: sources of right heuristic points
- colors: change of sampled points in each bucket
- squares: buckets that reuse previously sampled results

Figure 25: Example of avoiding redundant computation between iterations (for better Figure 3)

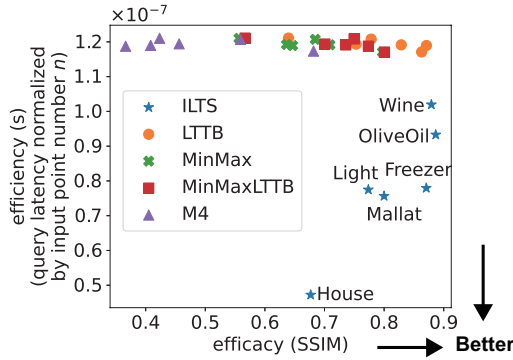


Figure 26: Efficiency v.s. efficacy, combining Figure 6 and Figure 8 under $m = 480$ (for better Figure 10)

A.10 Experiments on More Datasets

We add experiments on two large-scale real datasets in Figure 24. **Stock** is an open stock market data on Kaggle [<https://www.kaggle.com/datasets/debashis74017/stock-market-data-nifty-100-stocks-5-min-data>]. The largest input number used in our experiments is $n = 312, 960, 350$. **Train** is a train sensor dataset provided by real customers of Apache IoTDB. The largest input number used in our experiments is $n = 306, 726, 888$. The experimental results in Figure 24 show that ILTS achieves the best visualization efficacy, and its query latency increases linearly with the number of sampled points m , and sublinearly with the number of input points n . Note that the query efficiency advantage of ILTS becomes more apparent on a larger n . This is because as n increases, the percentage of disk pages pruned by the upper bounding technique (Section 4.3) also increases.

A.11 Other Improved Figures

We improve the original Figure 3 in Figure 25 by adding legends to make it more self-contained. We improve the original Figure 10 in Figure 26 by adding arrows indicating the better directions and adding text in the figure title to explain the data sources.