Largest Triangle Sampling for Visualizing Time Series in Database

LEI RUI, Tsinghua University, China
XIANGDONG HUANG, Tsinghua University, China
SHAOXU SONG*, Tsinghua University, China
CHEN WANG, Tsinghua University, China
JIANMIN WANG, Tsinghua University, China
ZHAO CAO, Huawei Technologies Co., Ltd, China

In time series visualization, sampling is used to reduce the number of points while retaining the visual features of the raw time series. Area-based Largest Triangle Sampling (LTS) excels at preserving perceptually critical points. However, the heuristic solution to LTS by sequentially sampling points with the locally largest triangle area (a.k.a. Largest-Triangle-Three-Buckets, LTTB) suffers from suboptimal solution and query inefficiency. We address the shortcomings by contributing a novel Iterative Largest Triangle Sampling (ILTS) algorithm with convex hull acceleration. It refines the sampling results iteratively, capturing a broader perspective by integrating more points in each iteration. Remarkably, we prove that the largest triangle can always be found in the precomputed convex hulls, making the iterative sampling still efficient. Experiments demonstrate increased visual quality over state-of-the-art baselines and significant speedups over the brute force approach.

CCS Concepts: • **Information systems → Database query processing**.

Additional Key Words and Phrases: time series visualization, database query processing

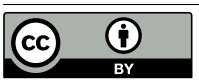
ACM Reference Format:
Lei Rui, Xiangdong Huang, Shaoxu Song, Chen Wang, Jianmin Wang, and Zhao Cao. 2025. Largest Triangle Sampling for Visualizing Time Series in Database. *Proc. ACM Manag. Data* 3, 1 (SIGMOD), Article 49 (February 2025), 26 pages. https://doi.org/10.1145/3709699

1 Introduction

Time series visualization, often represented by line charts [7], is extensively used. A contemporary visualization system consists of a back-end database server and a front-end visualization client. However, a naïve technique where the client queries the raw time series from the database fails to meet the rapid response time requirements, due to the large amounts of data transferred. Therefore, it is necessary to reduce the number of data points in the query result set, which can be accomplished by replacing raw data queries with sampling queries [27]. An ideal sampling query should meet two requirements: *visual quality* and *query efficiency*.

*Shaoxu Song (https://sxsong.github.io/) is the corresponding author.

Authors' Contact Information: Lei Rui, Tsinghua University, Beijing, China, rl18@mails.tsinghua.edu.cn; Xiangdong Huang, Tsinghua University, Beijing, China, huangxdong@tsinghua.edu.cn; Shaoxu Song, Tsinghua University, Beijing, China, sxsong@tsinghua.edu.cn; Chen Wang, Tsinghua University, Beijing, China, wang_chen@tsinghua.edu.cn; Jianmin Wang, Tsinghua University, Beijing, China, jimwang@tsinghua.edu.cn; Zhao Cao, Huawei Technologies Co., Ltd, Beijing, China, caozhao1@huawei.com.



ACM 2836-6573/2025/2-ART49
https://doi.org/10.1145/3709699

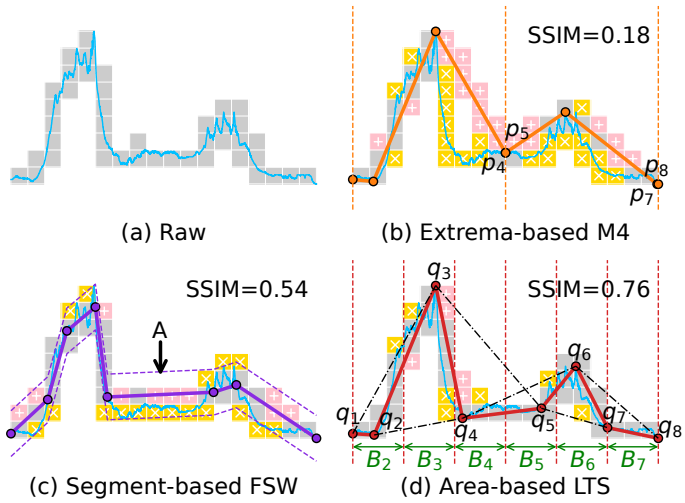


Fig. 1. Motivation of area-based sampling. Gray squares denote the pixels plotted by the raw data without sampling, i.e., (a) ground truth. Comparing (b)-(d) with (a), pink pixels (with "+" markers) indicate those plotted by the sampled data but not by the raw time series, while yellow pixels (with "x" markers) are marked in ground truth but not by the sampled 8 points, both the fewer the better.

1.1 Motivation

The Structural SIMilarity (SSIM) index (SSIM) is a widely used metric for assessing visual quality [19, 27, 31, 37, 38], with higher scores for better quality. Figure 1 shows some typical methods that plot a time series in 18 pixel columns, with/without sampling.

Ideally, M4 [27] selects $m = 4w$ points from w buckets that align with the pixel column time intervals, ensuring error-free visualization when rendered on a canvas with the target width w. However, user operations (e.g., resizing) and multiple physical screens (e.g., those on watches, phones and computers) may lead to various canvas width w, making the number of sampled data points m not exactly match the width w. When such a mismatch occurs, even slightly, the visual quality of M4 can deteriorate significantly. This degradation happens because the sampled points are tailored for line rasterization and become less effective, when the bucketing does not align with the pixel columns.

By selecting points that maximize the triangle area, perceptually critical points [9, 29] can be identified. For example, in Figure 1(d), the inflection point q_5 forms the largest triangle with its neighbors q_4 and q_6, and is selected by the area-based Largest Triangle Sampling (LTS) method (see Section 2 for the formal definition). Such inflection points effectively capture the change in trends before and after the inflection, leading to high structural similarity on pixels in the SSIM metric.

Unlike extrema-based methods which select horizontal or vertical extrema per bucket, area-based sampling can identify critical points that are not extrema. For example, the critical turning point q_5 sampled by LTS cannot be captured by the extrema-based method in Figure 1(b).

For segment-based methods, the goal of small distances in the data space (i.e., small y-differences) does not necessarily correspond to small distances in the pixel space. For example, Feasible Space Window (FSW) [30] in Figure 1(c) is an error-bounded piecewise linear approximation (ℓ_∞-PLA) method that greedily maximizes the segment length while bounding the y-difference of each point. The points are close to segment A in terms of y-differences, whereas the corresponding plotted pixels are distinct, i.e., low SSIM.

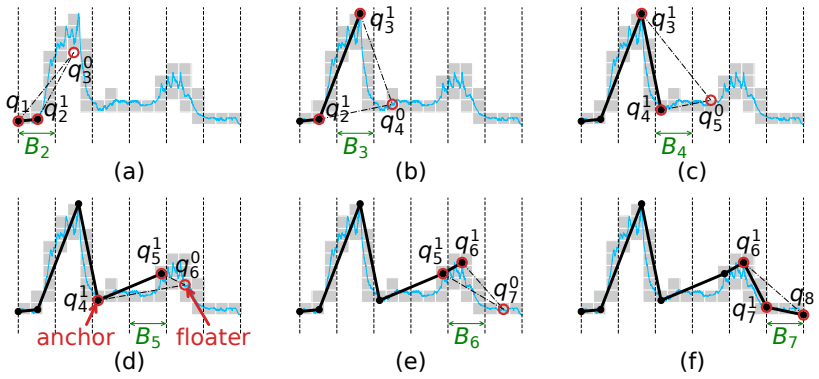


Fig. 2. Largest-Triangle-Three-Buckets (LTTB)

1.2 Challenge

Computing the LTS is obviously costly and not affordable, given various combinations of sampled points to form triangles. A linear-time approximation to the LTS problem, known as Largest-Triangle-Three-Buckets (LTTB) [34], sequentially samples points from left to right buckets. For each bucket, it selects the point that maximizes the triangle area formed with left anchor l and right floater r. Finally, all the anchor points constitute the sampling result.

As shown in Figure 2, for the current bucket being sampled, the term "left anchor" refers to the sampled point in the left bucket, which has already been determined. The term "right floater" is used because the sampled point in the right bucket has not yet been determined and is temporarily substituted by the average of the data points in the right bucket.

Yet LTTB remains shortsighted despite using adjacent buckets. First, the previous sampling errors in the left will affect the following ones. Moreover, the incoming points from further behind the right adjacent bucket will not affect the currently sampled point. Consequently, as shown in Figure 2(d), LTTB returns q_5^1 of bucket B_5 and misses the critical turning point q_5 as in Figure 1(d). Using the average point q_6^0 in bucket B_6 as the right floater, LTTB does not realize that there are higher peaks behind bucket B_5.

On the other hand, the query efficiency issue of LTTB is underestimated, especially in IoT scenarios. A time series is often segmented and stored in dispersed disk pages (see Section 5.2). Consequently, even a simple linear scan can be costly due to random I/O. Precomputation is often used for query acceleration, e.g., by pre-aggregating sums and counts to enable faster average calculations. Unfortunately, the triangle area formed by each point and its anchor-floater cannot be calculated in advance, because the floater varies with distinct bucketing specified by online queries.

1.3 Intuition

To address both challenges in visual quality and query efficiency, we propose a novel Iterative Largest Triangle Sampling (ILTS) algorithm with convex hull acceleration. Instead of relying one time on the fixed anchor and floater like LTTB does, ILTS iteratively uses the latest sampled points to refine the sampling results.

The rationale behind this iterative refinement is the propagation of the influence of points over iterations, both forward and backward, for a holistic view in visualization. As verified in Figure 3(c), ILTS uses the sampled point q_6^1 in B_6 during the first iteration as the right floater r and succeeds in

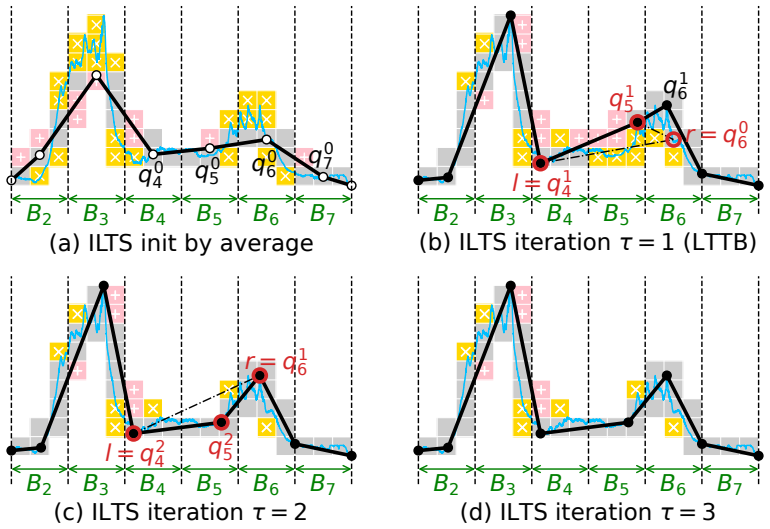


Fig. 3. Iterative Largest Triangle Sampling (ILTS)

selecting the critical point q_5^2 in B_5. Note that q_5^2 is affected by q_6^1 which is affected by q_7^0. In other words, the influence of B_7 propagates to B_5 through two iterations.

To enable precomputation without knowing buckets of sampling queries in advance, we calculate the convex hull for each disk page, and prove that the largest triangle in any bucket can always be found in the convex hulls. Note that the convex hull includes the outermost points of the set it encloses, and thus has the largest area among all polygons formed from a subset of the points.

1.4 Contribution

The contributions of this paper are summarized as follows:

- We formally define the Largest Triangle Sampling problem.
- We introduce ILTS, an Iterative heuristic algorithm for Largest Triangle Sampling. It progressively improves the sampling results by incorporating influence from more points in each iteration, and prunes redundant computations by identifying reuse conditions.
- We propose to leverage convex hulls for database query acceleration of ILTS. The pruning idea is to narrow the search space from raw data points to a smaller set of convex hull vertices and enable early termination of the search whenever possible.
- We present the implementation details in a time series database Apache IoTDB [36]. We employ a compact bitmap representation for storing convex hulls and leverage bounding boxes of convex hulls for I/O pruning. The source code for ILTS and other baselines is in the GitHub repository [2].
- We show the superiority of ILTS in visual quality and query efficiency through extensive comparative experiments with state-of-the-art baselines, including comparisons with two very recent works [31, 37] in the context of a time series database. Moreover, we demonstrate the energy efficiency advantage of ILTS in resource-constrained settings. We also perform an ablation study to understand the contribution of each component of ILTS. For reproducibility, the experimental code, data, and scripts are available in [3].

All the contributions of this paper are organized as follows. Section 2 introduces the LTS problem. We then present the ILTS algorithm in Section 3. Section 4 presents the convex hull pruning method.

To advance, Section 5 describes the system implementation. The experimental results are reported in Section 6. We review related work in Section 7. Finally, Section 8 concludes the paper.

2 Largest Triangle Sampling

The effective area of a point is the area of the triangle formed by that point and its two adjacent points [35]. The effective area of a time series is thus the sum of the effective areas of all non-terminal points within it. The goal of Largest Triangle Sampling (LTS) is to sample one point per bucket such that the resulting sampled time series has the largest effective area.

Definition 1 (Largest Triangle Sampling). *Given an input time series $T = \{p_1, \ldots, p_n\} = \{(t_1, v_1), \ldots, (t_n, v_n)\}$ consisting of n data points sorted in increasing order of time and a threshold m for the length of the sampled time series ($m \geq 3$), the Largest Triangle Sampling (LTS) divides the time range (t_1, t_n) into $m-2$ equal-width buckets and selects one point per bucket along with the global first and last points, such that the effective area of the sampled time series is maximized. Formally,*

$$LTS(T) = \arg\max_{T'} A(T'),$$

where

$$T' = \{q_i \in T \mid q_i.t \in B_i,\ i = 1, \ldots, m\},\ A(T') = \sum_{i=2}^{m-1} \Delta q_{i-1} q_i q_{i+1},$$

$$B_i = \begin{cases} \{t_1\} & \text{if } i = 1, \\ (t_1 + \frac{(t_n - t_1) * (i-2)}{m-2}, t_1 + \frac{(t_n - t_1) * (i-1)}{m-2}] & \text{if } i = 2, \ldots, m-2, \\ (t_1 + \frac{(t_n - t_1) * (m-3)}{m-2}, t_n) & \text{if } i = m-1, \\ \{t_n\} & \text{if } i = m. \end{cases}$$

Strictly speaking, the output of argmax is a collection of sampled time series with the same largest effective area. For ease of presentation, we use the equal sign to indicate returning any one of them. Note that for generality, we treat the global first point p_1 as q_1 selected from the singleton interval $B_1 = \{t_1\}$ and the global last point p_n as q_m selected from $B_m = \{t_n\}$. The time bucket B_i may be empty, i.e., $\nexists p \in T$, s.t. $p.t \in B_i$. In that case, the corresponding q_i in the sampled time series does not exist, and the length of the sampled time series is less than the threshold m. For simplicity, below we assume that all buckets are non-empty.

Example 1. *As shown in Figure 1(d), given the raw time series T (colored in blue) and a threshold $m = 8$ for the length of the sampled time series, the six time buckets $B_2, \ldots, B_7$ that equally divide the total time range are annotated in green. T' is obtained by selecting one point per equal-width bucket plus the global first and last points. $LTS(T)$ is the one with the largest effective area among all possible T'. The effective area of $LTS(T) = \{q_1, \ldots, q_8\}$ is the sum of the areas of the six dash-dotted triangles, i.e., $A(LTS(T)) = \Delta q_1 q_2 q_3 + \Delta q_2 q_3 q_4 + \Delta q_3 q_4 q_5 + \Delta q_4 q_5 q_6 + \Delta q_5 q_6 q_7 + \Delta q_6 q_7 q_8$.*

3 Iterative Sampling

As mentioned in the introduction, LTTB lacks the view of incoming buckets beyond the bucket immediately to the right. In this section, we present an iterative heuristic algorithm called Iterative Largest Triangle Sampling (ILTS) to address this limitation. ILTS iteratively refines its sampling results to capture a more holistic view of the underlying data.

3.1 Iterative Computation

The LTS problem in Definition 1 can be approximated using the largest triangle routine defined below. It relies on two fixed points to sample the local optimal point in a bucket at a time.

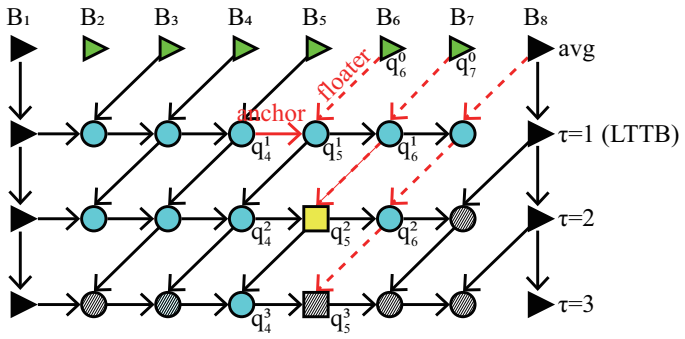


Fig. 4. The influence of future points spreads backward with each iteration. Horizontal and diagonal arrows indicate sources of left anchor and right floater points, respectively. Shape changes reflect alterations in sampled points.

Definition 2 (Largest Triangle Routine). *Given the set of data points $\mathbb{P}$ in a bucket, the left anchor point l, and the right floater point r, the largest triangle routine selects $p^* \in \mathbb{P}$ that forms the largest triangle area with l and r, i.e.,*

$$p^* = \arg\max_{p \in \mathbb{P}} \Delta lpr. \tag{1}$$

The remaining question is how to determine l and r. ILTS addresses this by sequentially and iteratively leveraging adjacent latest sampled points. Let q_i^τ denote the sampled point in the i-th bucket during the τ-th iteration, $\tau = 0$ for initialization. Then we have

$$q_i^\tau = \arg\max_{p \in T,\ p.t \in B_i} \Delta q_{i-1}^\tau p q_{i+1}^{\tau-1},\ \tau \geq 1,\ i = 2, \ldots, m-1.$$

It means when selecting the local optimal point in the current bucket B_i during the τ-th iteration, ILTS takes the sampled point in the left bucket during the current iteration as the left anchor l and the sampled point in the right bucket during the previous iteration as the right floater r.

Example 2. *Figure 3 shows the iterative process of ILTS. For bucket B_5, the sampled point is initialized by the average point q_5^0 as in Figure 3(a), and then iteratively updated by the largest triangle routine. Specifically, during the first iteration in Figure 3(b), q_5^1 is selected based on $l = q_4^1$ and $r = q_6^0$. And during the second iteration in Figure 3(c), the critical point q_5^2 with high curvature is preserved with $l = q_4^2$ and $r = q_6^1$. Finally in Figure 3(d), the third iteration does not further improve the result, i.e., achieving convergence. As additionally shown in Figure 4, the initially sampled points reflect only the influence of their own buckets. With each subsequent iteration, the number of buckets to the right that can directly or indirectly influence the sampling of the current bucket grows, as indicated by the red diagonal dashed arrows in Figure 4.*

3.2 Iterative Pruning

To speed up the computation of LTS, one observation is that during each iteration, we can prune sampling in a bucket, i.e., avoid the computation of the largest triangle routine, if its adjacent sampled points remain unchanged during their last two iterations.

Formally, for a bucket B_i to be sampled during the τ-th iteration, if $q_{i-1}^\tau = q_{i-1}^{\tau-1}$ and $q_{i+1}^{\tau-1} = q_{i+1}^{\tau-2}$, we have $q_i^\tau = q_i^{\tau-1}$, i.e., reusing the previously sampled point.

Example 3. *Shape changes in Figure 4 represent the change of sampled points in each bucket over iterations. The hatched shading indicates the buckets that meet the reuse condition during some*

iteration. For example, from $q_4^3 = q_4^2$ and $q_6^2 = q_6^1$, we can prune sampling in bucket B_5 during the third iteration, but reuse $q_5^3 = q_5^2$ instead.

3.3 Iterative Algorithm

The pseudocode of ILTS is shown in Algorithm 1. After initialization (line 2), during each subsequent iteration (line 4), proceeding sequentially from left to right buckets (line 6), ILTS applies the largest triangle routine in a bucket (line 11) with anchor-floater being its adjacent latest sampled points (line 10). Line 7 shows the condition under which a bucket can reuse its sampled point from the previous iteration. That is, $leftEq = true$ and $F_{i+1} = true$ means that its left and right sampled points remain unchanged during their last two iterations, respectively. Line 16 indicates the condition for early termination of the entire iteration process, which occurs when all sampled points remain unchanged in their last two iterations. In such cases, further iterations are unnecessary because the sampling result has stabilized.

Algorithm 1: Iterative Largest Triangle Sampling

Input: an input time series $T = \{p_1, \ldots, p_n\}$, a threshold m for the length of the sampled time series, the max number of iterations k
Output: a sampled time series $ILTS(T) = \{q_1, \ldots, q_m\}$

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

For initialization, q_i^0 can either be random sampling or average. We highlight that LTTB is a special case of ILTS with average initialization and single iteration. As a result, the acceleration technology of ILTS can be applied to LTTB without obstacles.

Note that ILTS is an approximation to the LTS problem but can give the optimal solution in some special cases, such as triangle waves [1]. Furthermore, ILTS usually converges after a few iterations, with empirical results showing that it stabilizes after about four iterations [4].

Proposition 1 (Optimal and convergent special case). *When the input is a triangle wave, ILTS can provide the optimal solution to the LTS problem and can converge in one iteration.*

Proof. Please see full proof in [4]. □

4 Convex Hull Pruning

In this section, we devise a convex hull based method to accelerate ILTS. Specifically, when iterative pruning in the previous section fails, in order to speed up the computation of the largest triangle routine (i.e., line 11 of Algorithm 1), we perform a fast search on the precomputed convex hull instead of raw data points, thereby reducing the CPU cost of traversing points.

4.1 Narrow Down the Search Space

Given a set of data points $\mathbb{P}$, the convex hull of $\mathbb{P}$ is the smallest convex polygon containing $\mathbb{P}$. That is, it is the unique convex polygon whose vertices are a subset of $\mathbb{P}$ and within which all points in $\mathbb{P}$ fall [16]. Let $CH(\mathbb{P}) \subseteq \mathbb{P}$ denote the vertices of the convex hull of $\mathbb{P}$. In the following, we transform the routine of finding the point with the largest triangle area formed with the anchor-floater pair (Definition 2) into a search confined to the convex hull vertices for the farthest point from the anchor-floater line (Proposition 2).

Proposition 2 (farthest convex hull vertex). *The largest triangle routine in Definition 2 is equivalent to selecting $p^* \in CH(\mathbb{P})$ that has the largest perpendicular distance to the fixed line L connecting l and r, i.e.,*

$$p^* = \arg\max_{p \in \mathbb{P}} \Delta lpr = \arg\max_{p \in CH(\mathbb{P})} d(p, L), \tag{2}$$

where

$$d(p, L) = \frac{|A * p.t + B * p.v + C|}{\sqrt{A^2 + B^2}},\ L : At + Bv + C = 0,$$
$$A = r.v - l.v,\ B = l.t - r.t,\ C = r.t * l.v - l.t * r.v.$$

Proof. First, because the two points l and r are fixed, the largest triangle routine is equivalent to selecting the point in $\mathbb{P}$ that is farthest from the line L connecting l and r, i.e.,

$$p^* = \arg\max_{p \in \mathbb{P}} \Delta lpr = \arg\max_{p \in \mathbb{P}} d(p, L).$$

Next, we prove that the point in $CH(\mathbb{P})$ that is farthest from L must also be the farthest in $\mathbb{P}$, i.e.,

$$\arg\max_{p \in \mathbb{P}} d(p, L) = \arg\max_{p \in CH(\mathbb{P})} d(p, L).$$

We use proof by contradiction. Let $d^* = \max_{p \in \mathbb{P}}\{d(p, L)\}$. From $CH(\mathbb{P}) \subseteq \mathbb{P}$, we know that $d(p, L) \leq d^*,\ \forall p \in CH(\mathbb{P})$. Suppose that $\max_{p \in CH(\mathbb{P})}\{d(p, L)\} \neq d^*$, which leads to $d(p, L) < d^*,\ \forall p \in CH(\mathbb{P})$. Then the point $\arg\max_{p \in \mathbb{P}} d(p, L)$ will fall outside the convex hull of $\mathbb{P}$, which is a contradiction to the definition of convex hulls and completes the proof. □

Therefore from Equation 1 to Equation 2, we narrow the search space from $\mathbb{P}$ to $CH(\mathbb{P})$. Note that $|CH(\mathbb{P})|$ is smaller than $|\mathbb{P}|$.

Example 4. *Given $\mathbb{P} = \{p_1, \ldots, p_{1000}\}$ plotted as blue lines in Figure 5, the convex hull of $\mathbb{P}$ is a convex polygon bounded by black lines with vertices represented by hollow points. Given $\mathbb{P}$, l, and r, the largest triangle routine selects p_{986}, which is the farthest convex hull vertex from the anchor-floater line L, and forms the largest triangle area with l and r.*

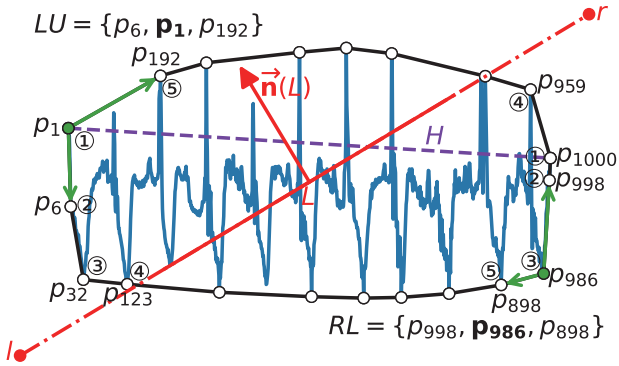


Fig. 5. Example of fast search on the convex hull

4.2 Fast Search on the Convex Hull

Next, we propose to speed up the search on the convex hull by terminating it early whenever possible. To achieve this, we first present a proposition for identifying the candidate farthest vertices. Then, we extract geometric connectivity from the chronologically organized vertices. Finally, we provide the complete fast search algorithm with the early stop mechanism.

4.2.1 Candidate Farthest Vertices. We identify potential farthest vertices by utilizing the normal vector $\vec{\mathbf{n}}(L) = (A, B)$ of the line $L : At + Bv + C = 0$. Recall that $L : At + Bv + C = 0$ is the fixed line connecting the left anchor point l and the right floater point r, as defined in Proposition 2.

Proposition 3 (extreme vertices along the normal vector). *For a convex hull vertex o_2, given its geometrically connected vertices o_1 and o_3, let*

$$s = sign(\overrightarrow{o_2o_1} \cdot \vec{\mathbf{n}}(L)) + sign(\overrightarrow{o_2o_3} \cdot \vec{\mathbf{n}}(L)),$$

where $sign(x)$ is the function returning the sign of x, i.e.,

$$sign(x) = \begin{cases} 1, & if\ x > 0, \\ -1, & if\ x < 0, \\ 0, & if\ x = 0. \end{cases}$$

If $s > 0$, o_2 is the lowest vertex in the direction of $\vec{\mathbf{n}}(L)$; if $s < 0$, o_2 is the highest vertex in the direction of $\vec{\mathbf{n}}(L)$; and if $s = 0$, o_2 is a non-extreme vertex along $\vec{\mathbf{n}}(L)$.

Proof. Table 1 enumerates all cases of the sum of signs. Here we prove the case for the sum of signs being zero, which happens when the signs are one positive and one negative. Both signs being zero means that o_1, o_2 and o_3 are on the same line, which is impossible for the convex hull vertices and thus will not happen. Imagine a coordinate system with o_2 as the origin and $\vec{\mathbf{n}}(L)$ as the y axis. Opposite signs of $sign(\overrightarrow{o_2o_1} \cdot \vec{\mathbf{n}}(L))$ and $sign(\overrightarrow{o_2o_3} \cdot \vec{\mathbf{n}}(L))$ mean that o_1 and o_3 are one higher and one lower than o_2 in the direction of $\vec{\mathbf{n}}(L)$. Therefore, o_2 is not an extreme vertex (i.e., neither highest nor lowest) along $\vec{\mathbf{n}}(L)$. Other cases can be similarly proved. □

According to Proposition 3, the convex hull vertex farthest from L must be among the extreme vertices along $\vec{\mathbf{n}}(L)$. That is, given the highest vertex $p_{hi} \in CH(\mathbb{P})$ and the lowest vertex $p_{lo} \in CH(\mathbb{P})$ along $\vec{\mathbf{n}}(L)$, we have

$$p^* = \arg\max_{p \in \mathbb{P}} \Delta lpr = \arg\max_{p \in CH(\mathbb{P})} d(p, L) = \arg\max_{p \in \{p_{hi}, p_{lo}\}} d(p, L). \tag{3}$$

Table 1. Enumeration of the sum of signs

$sign(\overrightarrow{o_2o_1} \cdot \vec{\mathbf{n}}(L))$	$sign(\overrightarrow{o_2o_3} \cdot \vec{\mathbf{n}}(L))$	s	Type of o_2
0	0	0	will not happen
-1	1	0	non-extreme
1	-1	0	non-extreme
0	1	1	lowest
1	0	1	lowest
1	1	2	lowest
0	-1	-1	highest
-1	0	-1	highest
-1	-1	-2	highest

Therefore, we can stop the search early once we find p_{hi} and p_{lo}. Again, from Equation 2 to Equation 3, we further reduce the number of points that need to be traversed.

Example 5. *As shown in Figure 5, for the convex hull vertex p_1, $\overrightarrow{p_1p_6}$ and $\overrightarrow{p_1p_{192}}$ form both obtuse angles with the normal vector $\vec{\mathbf{n}}(L)$, having $sign(\overrightarrow{p_1p_6} \cdot \vec{\mathbf{n}}(L)) + sign(\overrightarrow{p_1p_{192}} \cdot \vec{\mathbf{n}}(L)) = -2$. It means that p_1 is the highest vertex along $\vec{\mathbf{n}}(L)$ and thus might be the farthest point from L. Similarly, p_{986} might be the farthest point, as it is the lowest vertex along $\vec{\mathbf{n}}(L)$, having $sign(\overrightarrow{p_{986}p_{998}} \cdot \vec{\mathbf{n}}(L)) + sign(\overrightarrow{p_{986}p_{898}} \cdot \vec{\mathbf{n}}(L)) = 2$. Therefore, we terminate the search once the two extreme vertices are found, returning $p^* = \arg\max_{p \in \{p_1, p_{986}\}} d(p, L)$ without traversing the remaining vertices.*

4.2.2 Geometric Connectivity Reconstruction. Proposition 3 assumes the knowledge of geometric connectivity between vertices. However, we traverse the convex hull vertices in chronological order for storage representation reasons (see Section 5.2.2). Therefore, we need to reconstruct geometric connectivity while traversing.

Given a set of data points $\mathbb{P}$ sorted in increasing order of time, the first and last points of $\mathbb{P}$ must be the vertices of the convex hull of $\mathbb{P}$. Thus, we can divide the convex hull of $\mathbb{P}$ into upper and lower parts by connecting the first and last points of $\mathbb{P}$ [8]. Any two chronologically neighboring vertices on the same part of the hull must also be geometrically connected, forming the endpoints of a side of the convex polygon.

Example 6. *As shown in Figure 5, the line H connecting the first point p_1 and the last point p_{1000} divides the convex hull into upper and lower parts. p_{898}, p_{986}, and p_{998} are chronologically neighboring vertices in the lower part. Hence, $\{p_{898}, p_{986}, p_{998}\}$ is a triplet of geometrically connected vertices.*

4.2.3 Fast Search Algorithm. Combining the previous two sections, Algorithm 2 shows the fast search algorithm on the convex hull. Every time a convex hull vertex comes, we reconstruct its geometric connectivity by adding it to the corresponding sliding triplet (line 8-15, as in Section 4.2.2), and check if the middle point of that triplet is an extreme vertex along $\vec{\mathbf{n}}(L)$ (line 16-20, as in Section 4.2.1). Once the highest and the lowest vertices are found (line 21), the search stops and returns the farther extreme vertex (line 23).

LL, LU, RL, and RU in Algorithm 2 are sliding triplets used for geometric connectivity reconstruction in the left lower, left upper, right lower, and right upper parts of the convex hull, respectively.

Combining line 6 and line 7 in Algorithm 2, the directions "increasing" and "decreasing" refer to traversing the convex hull vertices alternately in ascending and descending order of time. Recall that the line H connecting the first and last points divides the convex hull into upper and lower parts. Thus, with time progressing from left to right ("increasing") and from right to left ("decreasing"), combined with the upper and lower parts of the convex hull, there are four regions where the triplets LL, LU, RL, and RU slide.

Algorithm 2: Fast search on the convex hull

Input: a set of data points $\mathbb{P}$ sorted in increasing order of time, the left anchor point l, the right floater point r

Output: the point $p^* \in \mathbb{P}$ that forms the largest triangle area with l and r

determine the convex hull vertices $CH(\mathbb{P})$ sorted in increasing order of time, the dividing line H that connects the first and last points of $\mathbb{P}$, the normal vector $\vec{\mathbf{n}}(L)$
initialize LL counterclockwise and LU clockwise by the first two convex hull vertices
initialize RL clockwise and RU counterclockwise by the last two convex hull vertices
initialize the flags $highest = lowest = false$
while *true* **do**
  for $dir \in \{increasing, decreasing\}$ **do**
    $p \leftarrow$ the next convex hull vertex with dir timestamp
    if $dir = increasing$ *and* p *is above or on* H **then**
      update LU by sliding it to p
    if $dir = increasing$ *and* p *is below or on* H **then**
      update LL by sliding it to p
    if $dir = decreasing$ *and* p *is above or on* H **then**
      update RU by sliding it to p
    if $dir = decreasing$ *and* p *is below or on* H **then**
      update RL by sliding it to p
    $p \leftarrow$ the middle point of the updated triplets
    if p *is the highest vertex along* $\vec{\mathbf{n}}(L)$ **then**
      $highest = true$, $p_{hi} = p$
    if p *is the lowest vertex along* $\vec{\mathbf{n}}(L)$ **then**
      $lowest = true$, $p_{lo} = p$
  if $highest = true$ *and* $lowest = true$ **then**
    break
$p^* \leftarrow$ the point farther from L between p_{hi} and p_{lo} as in Equation 3
return p^*

Example 7. *Continuing the example in Figure 5, Figure 6 shows the step breakdown. LL, LU, RL, and RU are sliding containers, each holding a triplet of three geometrically connected convex hull vertices. At initialization,* $LU = \{p_6, p_1, null\}$, $LL = \{p_1, p_6, null\}$, $RU = \{p_{998}, p_{1000}, null\}$, $RL = \{p_{1000}, p_{998}, null\}$. *Then vertex* p_{32} *is traversed in increasing chronological order in the first round, and* LL *is updated as* $\{p_1, p_6, p_{32}\}$. *The middle point* p_6 *of the updated* LL *is a non-extreme vertex by Proposition 3. When vertex* p_{192} *is traversed in the third round as shown in Figure 6(d), the highest vertex* p_1 *along* $\vec{\mathbf{n}}(L)$ *is found in the updated triplet* $LU = \{p_6, p_1, p_{192}\}$. *Similarly, the lowest vertex* p_{986} *is found in* $RL = \{p_{998}, p_{986}, p_{898}\}$. *With* p_1 *and* p_{986} *found, the search terminates and outputs* p_{986}, *which is farther from L than* p_1. *The traversal of remaining vertices in the middle part is skipped.*

5 System Implementation

In this section, we describe the implementation details of our in-database ILTS. We start from the system deployment with the database as a black box. After that, we delve into the database architecture, presenting the physical storage of time series and convex hulls, as well as the sampling query processing logic.

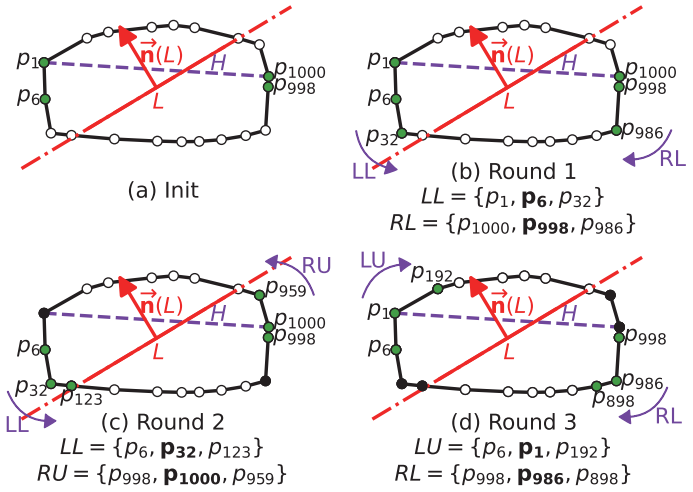


Fig. 6. Step breakdown of fast search on the convex hull

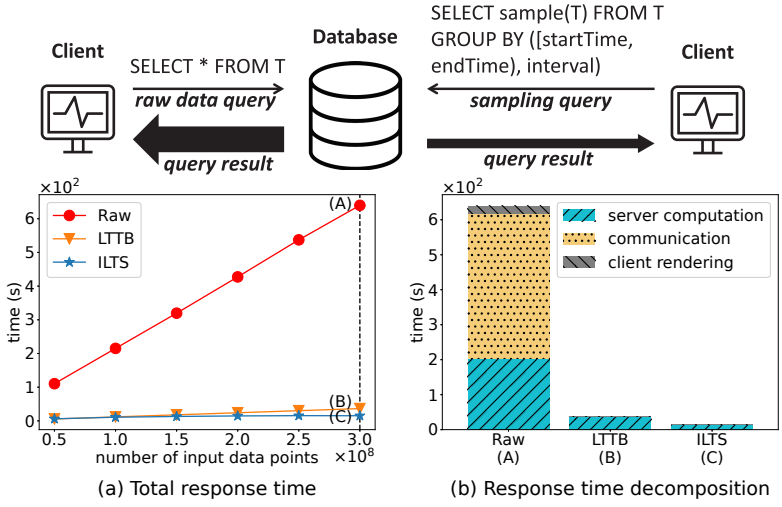


Fig. 7. Visualizing time series from a remote database without/with in-database sampling queries (baseline LTTB and our ILTS)

5.1 System Deployment

When a user requests an ad-hoc visualization of a time series, the visualization client sends an online query request to the database. The database executes the query and sends the query result set back to the client. Finally, the client renders the line chart on the screen. The sampling query that benefits from this work follows the format: `SELECT sample(T) FROM T GROUP BY ([startTime,endTime),interval)`, where `interval` is the time length of the sampling bucket. As indicated in the upper half of Figure 7, in-database sampling queries, as opposed to raw data queries, can greatly reduce the size of the query result set.

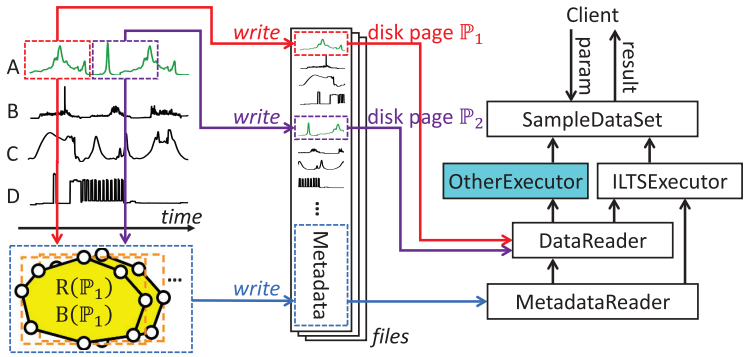


Fig. 8. ILTS and other baselines implemented in a time series database Apache IoTDB

The results in Figure 7 are obtained from real experiments, with experimental code and data available in [3]. The sampling queries (either LTTB or our ILTS) show significant speedup, compared to the non-sampling query encapsulating and transferring the entire raw data. The speedup is more obvious with large n (reaching 300 million points), given the much smaller sample size.

As shown in Figure 7, it is evident that in-database sampling queries, such as LTTB, can significantly reduce the data transfer time between the remote database and the visualization client, compared to the raw data queries. Instead, query execution time in the database server becomes the bottleneck. Hence our work contributes to accelerating such in-database sampling queries to further decrease the response time of visualization applications. With hundreds of millions of input points, our ILTS with convex hull acceleration turns out to achieve a 3–5× speedup compared to the brute force LTTB approach.

5.2 Database Architecture

Figure 8 gives an overview of our implementation in an open-source time series database Apache IoTDB [36]. The source code of ILTS and other baseline methods have been committed to the GitHub repository [2] of Apache IoTDB by the system developers.

5.2.1 Time Series Storage. Note that high concurrency data writes are common in IoT scenarios, where 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 senor) are first buffered in memory. When the limited memory buffer is full, the buffered segment is flushed to the disk, stored 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. For instance, as shown in Figure 8, the time series "A", highlighted in green, is stored as two pages $\mathbb{P}_1$ and $\mathbb{P}_2$ across dispersed disk locations. We maintain a convex hull for each disk page and store these convex hulls in the metadata. Metadata remains in memory until the file is closed, at which point it is flushed contiguously to the end of the file, making access to the metadata I/O friendly.

5.2.2 Convex Hull Representation. For compact metadata storage, we use a bitmap to represent the convex hull of a disk page $\mathbb{P}_j$, with each bit indicating whether the corresponding point in chronological order is a convex hull vertex or not. That is,

$$B(\mathbb{P}_j) = (b_1 b_2 \dots)_2,$$

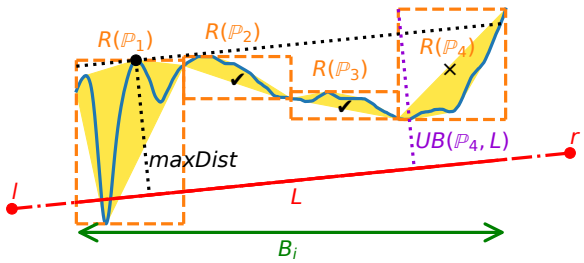


Fig. 9. Bounding box of convex hull for I/O Pruning

where for $p_i \in \mathbb{P}_j$,

$$b_i = \begin{cases} 0, & \text{if } p_i \text{ is not a vertex of the convex hull of } \mathbb{P}_j, \\ 1, & \text{otherwise.} \end{cases}$$

Note that the number of vertices of a convex hull is not known in advance, and using bitmap representation for convex hulls helps to keep the metadata space overhead manageable. There exist many algorithms to construct the planar convex hull, including Jarvis March [25], Graham Scan [24], Quickhull [11], and Chan's algorithm [13]. In the implementation, Quickhull is used to compute the convex hull for each flushed disk page. As we will show later in the experiments, the effort of precomputing convex hulls pays off in terms of improved query efficiency.

5.2.3 Pruning by Bounding Box. Note that the bitmap representation requires scanning the corresponding disk page to obtain convex hull vertices, as the bitmap only indicates the locations of vertices within the disk page. Therefore, we propose to use the bounding boxes [33] of convex hulls to further prune I/O of disk pages.

Formally, the bounding box of a disk page $\mathbb{P}_j$ is $R(\mathbb{P}_j) = \{p \mid p.t \in \{t_{min}, t_{max}\}, p.v \in \{v_{min}, v_{max}\}\}$, where $t_{min} = min(\{p.t \mid p \in \mathbb{P}_j\})$, $t_{max} = max(\{p.t \mid p \in \mathbb{P}_j\})$, $v_{min} = min(\{p.v \mid p \in \mathbb{P}_j\})$, $v_{max} = max(\{p.v \mid p \in \mathbb{P}_j\})$. We have the following proposition for the loose upper bound on the point-to-line distance.

Proposition 4 (distance upper bound). *Given a disk page $\mathbb{P}_j$ and the fixed anchor-floater line L, the perpendicular distance from the point in $\mathbb{P}_j$ to L has an upper bound $UB(\mathbb{P}_j, L) = max(\{d(p, L) \mid p \in R(\mathbb{P}_j)\})$.*

Example 8. *As Figure 9 shows, points in bucket B_i are stored into four pages, i.e., $\mathbb{P} = \{p \in T \mid p.t \in B_i\} = \{\mathbb{P}_1, \ldots, \mathbb{P}_4\}$. For each page, its convex hull is colored yellow, and its bounding box is plotted as an orange dashed rectangle. The distance from $\mathbb{P}_4$ to L is upper bounded by the distance from the upper left corner of $R(\mathbb{P}_4)$ to L.*

Therefore, we explicitly store the bounding box $R(\mathbb{P}_j)$ of each convex hull in the metadata and leverage them to prune I/O and further speed up query processing. As shown in Figure 9, given the largest distance *maxDist* currently found from the processed pages, for the following page $\mathbb{P}_j$, if $UB(\mathbb{P}_j, L) \leq maxDist$, then $\mathbb{P}_j$ can be pruned as a whole (e.g., $\mathbb{P}_2$ and $\mathbb{P}_3$). The pruned disk pages do not need to be loaded and searched, because they must not contain the point whose distance from L exceeds *maxDist* according to Proposition 4. Otherwise, when the pruning fails, we load the corresponding page (e.g., $\mathbb{P}_4$) from disk and apply fast search on its convex hull as in Section 4 to update *maxDist*.

5.2.4 Complexity Analysis. Let n be the number of input data points and m be the number of sampled points. Suppose each disk page contains c data points where c is a constant, and each number occupies 64 bits. We analyze below that our method has a space complexity that is linear with respect to the number of input points, and a time complexity that is linear in the number of sampled points and logarithmic in the number of input points.

Space Complexity. According to Section 5.2.2 and Section 5.2.3, we store n bits for the convex hull bitmap representation and $n/c \times 256$ bits for the bounding boxes of convex hulls. Therefore, the space complexity is linear with respect to n.

Time Complexity. The number of disk pages in a bucket is approximately $\frac{n}{m \cdot c}$. In the best case, we only need to access the bounding box of the convex hull of each disk page in a bucket, i.e., traversing $4 \cdot \frac{n}{m \cdot c}$ points. Enumerate c from 1 to $\frac{n}{m}$, i.e., from one point per page to all points in a bucket per page, then the average number of traversed points in a bucket is $4\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n/m}\right)$. Therefore, according to the harmonic number, the time complexity for a single bucket is $\log(\frac{n}{m})$. The total time complexity for m buckets is thus $m \cdot \log(\frac{n}{m})$. Given that m is usually much smaller than n, we have $m \cdot log(\frac{n}{m}) \approx m \cdot \log n$. Thus, the overall time complexity of ILTS is $m \log n$.

5.2.5 Sampling Query Processing. As shown in Figure 8, the core query processing logic of ILTS is implemented in `ILTSExecutor`, which relies on `MetadataReader` to access convex hull metadata $B(\mathbb{P}_j)$ and $R(\mathbb{P}_j)$, and `DataReader` to access disk pages $\mathbb{P}_j$. During each iteration of Algorithm 1, the `ILTSExecutor` reuses the previously sampled point in a bucket whenever possible by iterative pruning (line 7 of Algorithm 1). For the unpruned bucket, the executor computes the largest triangle routine (line 11 of Algorithm 1), which is equivalent to searching among disk pages of the bucket for the farthest point from the anchor-floater line. To accelerate this, `ILTSExecutor` utilizes the bounding boxes $R(\mathbb{P}_j)$ to prune pages that must not contain the farthest point (Section 5.2.3). For the unpruned page, I/O operations are inevitable; that is, the page $\mathbb{P}_j$ is loaded from disk to memory. However, `ILTSExecutor` can reduce the CPU cost of traversing points by convex hull pruning (Algorithm 2). Finally, `SampleDataSet` is used to consume results from `ILTSExecutor` and encapsulate them to return to the database client. While compact metadata is loaded into memory as a whole at the start of the query, dispersed disk pages are loaded only when necessary.

As indicated by `OtherExecutor` (in blue box) in Figure 8, other baselines are also implemented natively in the time series database Apache IoTDB, for a fair comparison. Each sampling method is executed by a distinct `Executor` on the server, with `DataReader` serving as a common infrastructure for accessing disk pages. The difference is that `MetadataReader` for convex hulls is used exclusively by the `ILTSExecutor`.

6 Experiments

In this section, we first compare our proposed ILTS with other baseline methods to verify the performance gains of our method in terms of both visual quality and query efficiency. Then we perform an ablation study to understand the contribution of each component of ILTS. All experiments are conducted on the machine from Alibaba Cloud with Intel(R) Xeon(R) Platinum 8369B CPU @ 2.70GHz processors, 32GB of memory, and 250GB SSD. Experimental code, data, and scripts are available in [3] for reproducibility.

6.1 Experimental Settings

6.1.1 Baselines. We compare ILTS with baselines in four categories, i.e., (1) area based: our ILTS, LTTB [34], MinMaxLTTB [18], Visvalingam-Whyatt (Visval) [35], (2) extrema based: MinMax, M4

[27], Uniform, (3) segment based: Feasible Space Window (FSW) [30], Sim-Piece [28], (4) transform based: Discrete Fourier Transform (DFT) [6], Principal Component Analysis (PCA) [26].

For a fair comparison, all methods are calibrated to produce similar numbers of sampled points m. Specifically, for methods using buckets (ILTS, LTTB, MinMaxLTTB, MinMax, M4, and Uniform), we adjust the number of buckets based on the number of points sampled per bucket. For example, M4 samples four points per bucket, leading to $m/4$ buckets. For segment-based methods controlled by error thresholds (FSW and Sim-Piece), we conduct a binary search beforehand to identify the threshold that yields approximately m sampled points. For Visval, DFT, and PCA, we retain the m most significant points or components, based on area, amplitude, and variance, respectively.

In addition to the sampling algorithms above, we also compare our approach in two other application scenarios. OM3 [37] precomputes a multi-level representation of time series to support interactive progressive visualization. MinMaxCache [31] provides error-bounded adaptive caching as a middleware for user operations.

6.1.2 Datasets. We first experiment on the UCR datasets [14, 15] to demonstrate the significance of improvement in terms of visual accuracy. Since each series in the UCR datasets is relatively short, we concatenate all series within each class to form longer series. In total, we experiment on 120 long series from the 128 classes in the UCR datasets, excluding those containing null values.

Then we experiment on real-world long time series from the NIST campus photovoltaic (PV) arrays and weather station [12]. This station collects photovoltaic irradiance data and meteorological data, with a collection frequency of 1 second, spanning four years from 2015 to 2018. Specifically, we use four datasets in the experiments: **Qloss** for battery charge, **Pyra1** for solar irradiance, **WindSpeed** for wind speed, and **RTD** for panel temperature.

6.1.3 Canvas Size and Sampling Size. We provide details about the experimental evaluation regarding the width and height of the visualization canvas in pixels. Overall, most of the number of M4 buckets $m/4$ is less than the number of pixel columns w in our experiments. The reason is that the area-based sampling methods (LTTB and our ILTS) work on a relatively small sample size, where the error-free condition of M4 does not apply. It is always promising to sample and transfer a smaller set of points. This is particularly important in resource-constrained scenarios (such as energy and bandwidth). Given the limited battery capacity of end devices, a smaller sample size can reduce energy consumption, thereby extending the battery life of these devices. To validate that, we conduct a battery life experiment in Section 6.5.1.

6.1.4 Visual Quality Metrics. In our visual quality experiments, in addition to direct visual inspection of line charts, we use SSIM [19, 27, 31, 37, 38] to quantitatively measure the perceived image distortion. The Structural SIMilarity (SSIM) index [38] is proposed to account for variations in luminance, contrast, and structural components that are valued by human perception. SSIM yields a score ranging from -1 to 1, where values closer to 1 indicate less perceived distortion and higher visual quality. We use the standard Mean SSIM (MSSIM) [5, 38] implementation, which first calculates SSIM within a sliding window across the entire image, and then averages the SSIM values from all windows to obtain the overall image quality score.

6.1.5 Database Performance Metrics. For experiments related to query efficiency, we measure the database query execution time.

6.2 Accuracy Comparison

We first compare the visualization accuracy, measured by SSIM, of the 11 sampling methods on the UCR dataset, with a fixed canvas size of 1000×250 and 800 sampled points. The results are presented in Figure 10 using a critical difference diagram based on the Wilcoxon-Holm test [17, 21].

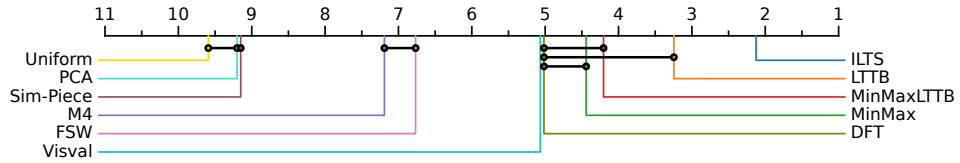


Fig. 10. Critical difference diagram with Wilcoxon-Holm test on UCR datasets

A lower rank indicates better performance, and methods with no significant pairwise difference in accuracy form a clique (illustrated by a horizontal black line connecting the dots representing the methods in the clique). The results show that our ILTS ranks first, achieving the highest SSIM and demonstrating a significant difference compared to other methods.

6.3 Visualization Comparison

Figure 11 presents the line chart visualizations generated by various sampling algorithms on a fixed 250×150 canvas. Each column corresponds to a specific dataset covering approximately one month of data (2.5 million points). By default, 100 points are sampled per dataset, except for the WindSpeed dataset, where 200 points are sampled to account for its more random fluctuation patterns. The first row, colored in green, displays the line charts of the raw time series. Each subsequent row shows the line charts by a different sampling method, with MMLTTB short for MinMaxLTTB. The SSIM score is annotated on top of each chart to quantitatively assess its visual quality, with the first row serving as the ground truth. Within each dataset column, ILTS with the highest SSIM value (bolded) is highlighted by a blue solid-line rectangle. Additionally, visual defects are emphasized with red dotted rectangles.

Next we take a closer look at Figure 11. In the panel temperature dataset **RTD**, the line chart generated by LTTB captures the daily periodic variations and the long-term declining trend (due to the winter season). However, regarding the micro-fluctuations throughout the day, as indicated by the red dotted rectangle in Figure 11, LTTB portrays the trend that starts slowly and then rises quickly as a uniformly increasing trend, hindering overheating risk identification and thermal designs. The visual defects of MinMax stem from jaggedness caused by alternating vertical extrema and the artificial lines formed by neighboring extrema of the same polarity [19]. Sim-Piece generates disjoint segments whose endpoints may not exist in the raw data. Thus, visual artifacts are produced from the connected lines of disjoint ends, leading to poor visualization. As discussed in [23], DFT and PCA focus on lower bounding the Euclidean distance while smoothing out perceptually critical points, resulting in poor visual quality.

Apart from using the SSIM metric to evaluate visual quality, we conduct a user study to investigate people's preferences for visualization results in Figure 11. The user study involves 20 participants, all with some experience in data analysis, including one who is a product manager at Apache IoTDB. Following the same line of [34], we ask participants to select the four charts they consider closest to the raw chart, from the 11 charts for each dataset shown in Figure 11. The results show that ILTS receives the most votes, aligning with the SSIM metric.

6.4 Parameter Evaluation

In this section, we quantitatively compare the methods in terms of visual quality, query efficiency and scalability. We exclude sampling methods with nonlinear time complexity, such as DFT, as well as those with clearly poor visual quality, such as Uniform. The default number of input points in this section is 10 million, covering approximately four months of data.

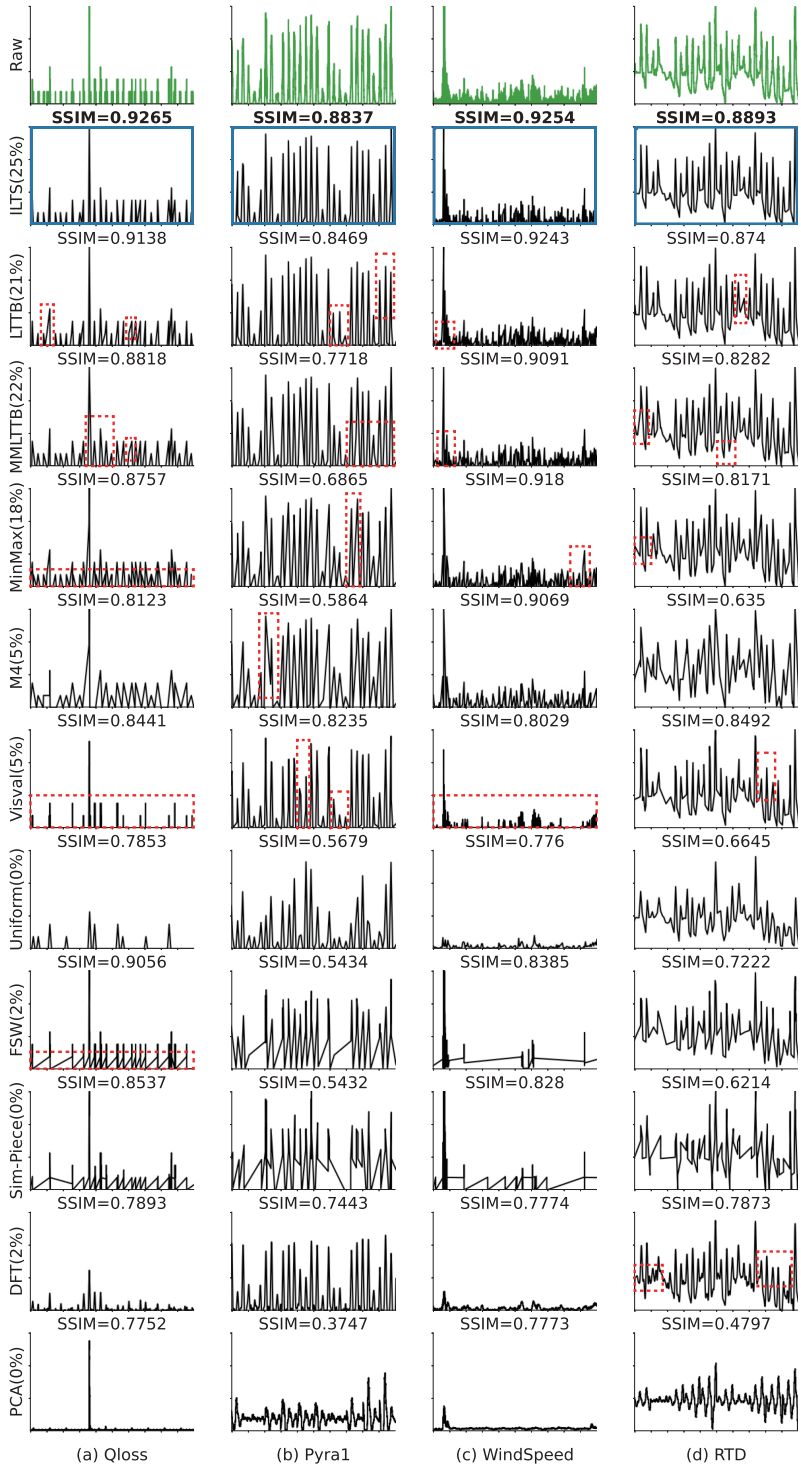


Fig. 11. Visualizations of sampled time series. Percentages in parentheses are the votes in the user study.

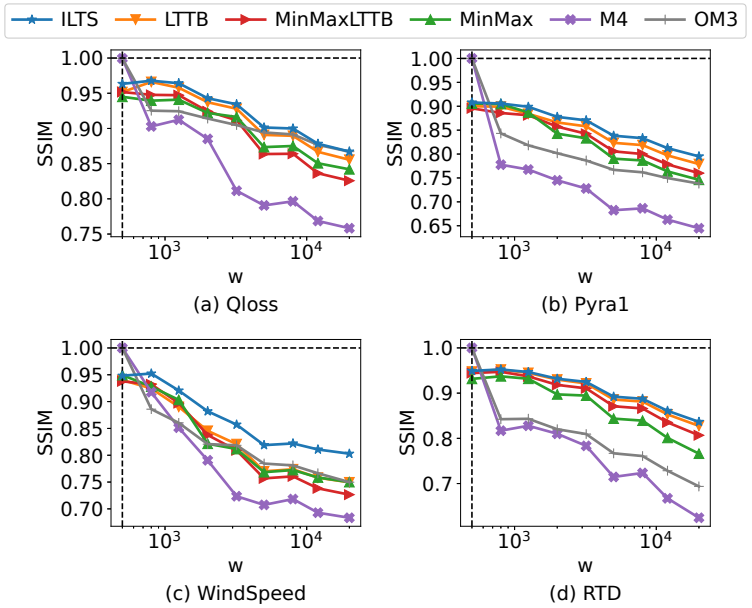


Fig. 12. Effectiveness on the number of pixel columns w. The black vertical dashed line represents $w = m/4$.

6.4.1 Varying the Number of Pixel Columns w. Figure 12 presents the SSIM comparison results as the number of pixel columns w varies, with a fixed number of sampled points $m = 2000$. M4 operates under its ideal condition of $m = 4w$ when $w = m/4 = 500$, achieving a perfect SSIM score of 1, the highest among all methods. However, as w increases, M4's SSIM decreases rapidly, becoming the worst among all methods. In contrast, ILTS demonstrates more robust performance, consistently achieving the highest SSIM in most cases (except for the above $w = m/4$).

OM3 exhibits a higher SSIM than M4 as it returns more sampled points than M4 given the same number of buckets [37]. Despite this, ILTS still achieves a higher SSIM than OM3 in most cases.

6.4.2 Varying the Number of Sampled Points m. Figure 13 shows the SSIM results under various numbers of sampled points m on a fixed 2500×1500 canvas. Again, ILTS has the highest SSIM among all methods in most cases, attributed to the area-based routine and the iterative refinement.

Figure 14 shows the query latency comparison results under different m. The query latency of ILTS increases with m, because a larger m means more buckets specified in the query and hence fewer disk pages per bucket, reducing the I/O pruning effectiveness. The query latency of other methods remains nearly constant with m, because they read and traverse all input points anyway.

We increase the number of sampled points m to meet $m = 4w$ in Figures 13 and 14 so that the case for M4 buckets to align with their corresponding pixel column time intervals is included in our experiments. The SSIM of M4 under $m = 4w$ is higher than others, but it drops sharply when the sample size m is slightly decreased or increased, as also observed in [27].

We include OM3 in the experiments in Figures 13 and 14. Again, this very recent work exhibits a higher SSIM than M4, because it actually returns more sampled points than M4 given the same number of buckets [37]. The time cost of OM3 is higher than that of M4 in Apache IoTDB implementation, due to the power-of-two length padding [37]. Note that the time cost comparison of OM3 may be unfair, since Apache IoTDB does not support to build an index as the original PostgreSQL implementation of OM3. We store the OM3 coefficient table (schema: id, minc, maxc)

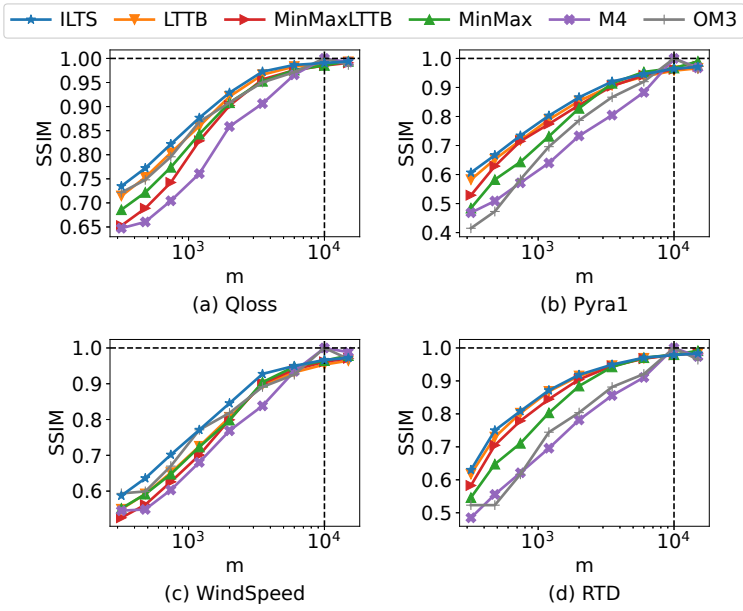


Fig. 13. Effectiveness on the number of sampled points m. The black vertical dashed line represents $m = 4w$.

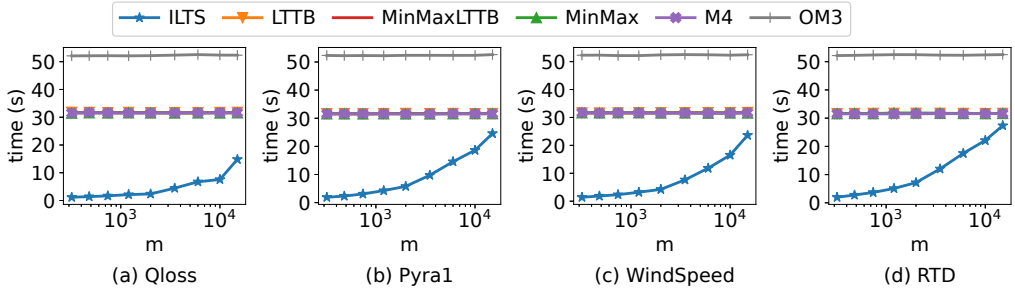


Fig. 14. Efficiency on the number of sampled points m

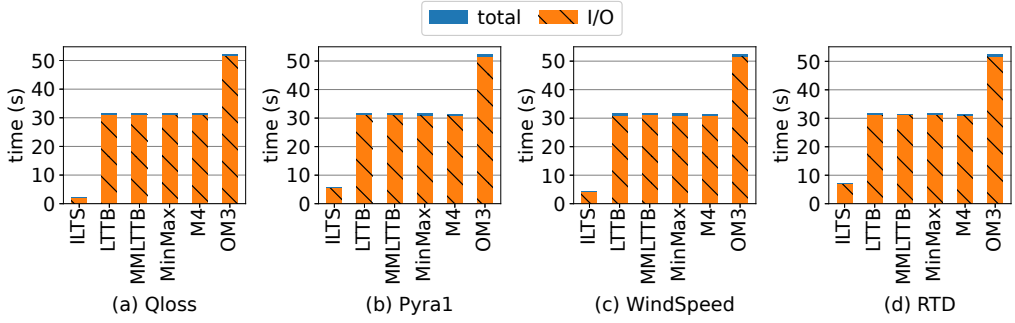


Fig. 15. Query latency breakdown of Figure 14

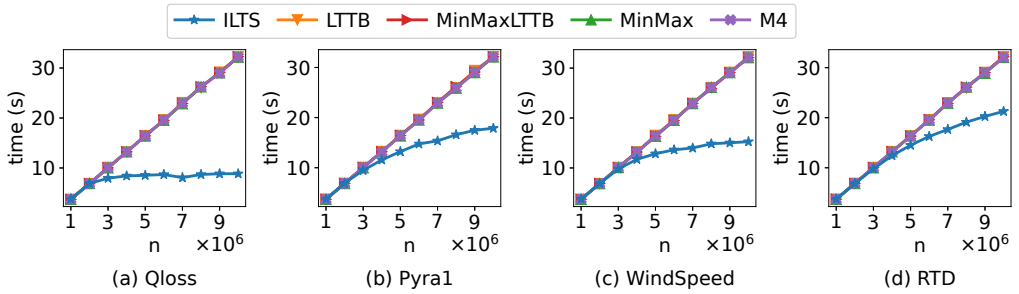


Fig. 16. Scalability on the number of input points n by varying the dataset size

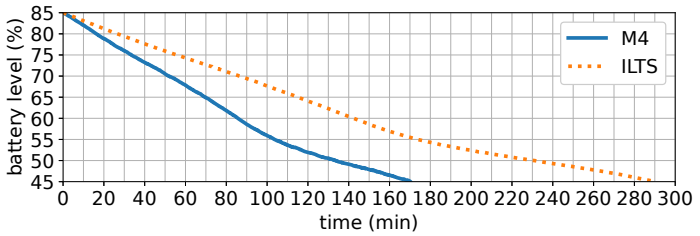


Fig. 17. Raspberry Pi battery level changes over time

as two time series (with id as timestamps and minc, maxc as values) in Apache IoTDB. Unlike PostgreSQL's index-based queries on id, IoTDB queries rely on time filters without index support.

The reason for the runtime results of baselines to be similar is that they read and traverse all input points. To provide further insights, we include an additional experiment that breaks down the query latency of the baseline runtime under $m = 2000$ of Figure 14 into I/O and CPU time, as shown in Figure 15. ILTS exhibits the lowest I/O time, attributed to the convex hull acceleration.

6.4.3 Varying the Number of Input Points n by Dataset Size. We vary the dataset size n in Figure 16 (varying query selectivity performing similarly in [4]), while fixing the number of sampled points to 8k. As n increases, each bucket contains more disk pages, thereby increasing the opportunity of disk pages being pruned. As a result, the query latency of ILTS scales sublinearly with n. The query latency of other methods increases linearly with n, as they process all input points.

6.5 Application Evaluation

6.5.1 Battery Life Experiment. Recall that in Figures 13 and 14, ILTS achieves reasonably good SSIM scores (e.g., >0.85) with fewer sampled points and lower query latency compared to M4. To validate the energy saving benefits of ILTS in resource-constrained scenarios, we test the battery life of end devices with querying and transmitting workloads. We install the database, with both ILTS and M4 implemented, on a Raspberry Pi 4B device. Over each 200-second period, the database executes 10 concurrent queries and transmits the compressed query results to a remote client. The battery levels for ILTS sampling 2k points and M4 sampling 10k points are shown in Figure 17. During the drop in battery level from 85% to 45%, M4 works only for 170 mins, while our ILTS lasts for 290 mins. Consequently, ILTS handles 1.7 times more query requests than M4, processing 870 and 510 requests, respectively.

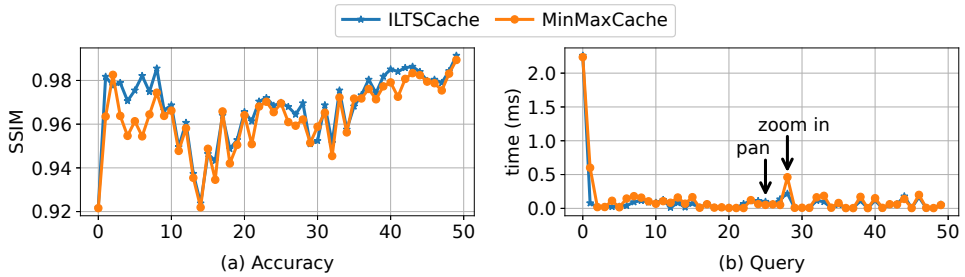


Fig. 18. ILTSCache v.s. MinMaxCache

Table 2. Ablation study settings

Acceleration component	$ILTS_a$	$ILTS_b$	$ILTS_c$	$ILTS_d$
Iterative Pruning (Section 3.2)		✓	✓	✓
Convex Hull Pruning (Section 4)			✓	✓
I/O Pruning (Section 5.2.3)				✓

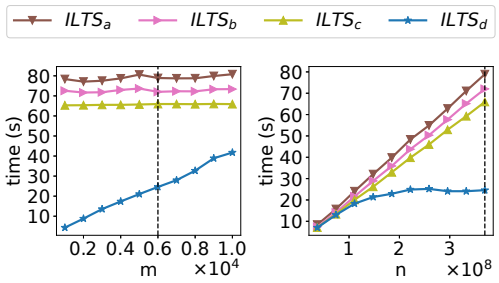


Fig. 19. Query latency results of the ablation study

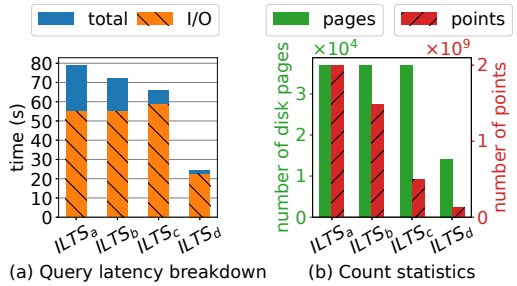


Fig. 20. Query latency breakdown and count statistics

6.5.2 Interactive Visual Exploration with Caching. MinMaxCache [31] considers an interesting scenario of user operations (e.g., pan, zoom). While our proposal is not originally designed for such interactive visual exploration, we can reuse the previously fetched sampled points in such operations (which we refer to as ILTSCache). To understand ILTS's applicability in real-time visualization scenarios, we conduct an experiment in Figure 18, comparing MinMaxCache and ILTSCache (both built on top of Apache IoTDB) in terms of visual quality (SSIM) and query response time in the case of user operations. For MinMaxCache, we adopt the experimental parameters from [31], including an error bound of 5% and an initial aggregation factor (AF) of 4. For ILTSCache, AF is fixed at 8, twice the initial AF of MinMaxCache, as the latter aggregates two values (min and max) per group. The error evaluation steps in MinMaxCache are omitted in ILTSCache. The results show that the performance of ILTSCache is comparable to MinMaxCache.

6.6 Ablation Study

To understand the contribution of each acceleration component of ILTS, we perform an ablation study on the RTD dataset with a larger scale, totaling 370 million data points, recording panel temperature from multiple locations spanning four years. The ablation settings are shown in Table 2.

Table 3. Overview of related sampling methods, where n is the number of raw data points, m is the number of sampled points, and ϵ is the maximum error threshold.

Method	Category	Time Complexity	Control
ILTS (our)	area-based	$O(m \log n)$	m
LTTB [34]	area-based	$O(n)$	m
MinMaxLTTB [18]	area-based	$O(n)$	m
Visval [35]	area-based	$O(n \log n)$	m
MinMax	extrema-based	$O(n)$	m
M4 [27]	extrema-based	$O(n)$	m
Uniform	extrema-based	$O(n)$	m
FSW [30]	segment-based	$O(m \cdot n)$	ϵ
Sim-Piece [28]	segment-based	$O(n + m \log m)$	ϵ
DFT [6]	transform-based	$O(n \log n)$	m
PCA [26]	transform-based	$O(m^2 \cdot n)$	m

Three components are gradually added from $ILTS_b$ to $ILTS_d$, and $ILTS_a$ is the brute force approach without using any of them. Iterative pruning (Section 3.2) reduces buckets to be resampled by identifying reuse opportunities between iterations; convex hull pruning (Section 4) reduces points to be traversed by narrowing down the search space from raw points to convex hull vertices and terminating the search when possible; I/O pruning (Section 5.2.3) reduces disk pages to be loaded by pruning pages that must not contain the farthest point.

The results are shown in Figure 19 and Figure 20. Figure 19 presents the query latency under various m and n. Fixing the number of sampled points m and input points n (as indicated by dashed lines in Figure 19), Figure 20 provides a detailed investigation of the query latency breakdown, as well as count statistics including the number of disk pages accessed and points traversed.

As also observed in previous experiments, the query latency of $ILTS_d$ (i.e., final ILTS) increases linearly with m and sublinearly with n. The query latency of $ILTS_a$, $ILTS_b$, and $ILTS_c$ remains constant with m and increases linearly with n. The latency breakdown and count statistics reveal that all three acceleration components contribute to reducing points traversed, thereby reducing CPU time. The I/O pruning component further reduces disk pages accessed, thus reducing I/O time.

The time complexity of the method with I/O pruning is $O(m \log n)$ as discussed in Section 5.2.4, while the time complexity of the method with only CPU processing acceleration (iterative pruning and convex hull pruning) is linear with respect to the data size n. As a result, I/O pruning significantly reduces I/O time.

7 Related Work

While there is research on lossy compression for machine learning tasks [10, 20, 22, 32], our work focuses on sampling for visualization tasks. Table 3 lists the four categories of sampling methods investigated in our paper. Below we focus on research addressing time-domain sampling methods.

7.1 Area-based Sampling

Visvalingam-Whyatts (Visval) [35] eliminates the point with the least effective area in a bottom-up manner. Linear-time Largest-Triangle-Three-Buckets (LTTB) [34] sequentially samples points from left to right buckets. Largest-Triangle-Dynamic (LTD) [34] further adjusts buckets through splitting and merging, and then applies LTTB on the buckets. Jonas Van Der Donckt et al. [18] confirm that LTTB excludes vertical extrema and propose MinMaxLTTB to solve the problem. However, our experimental results indicate that the visual quality of MinMaxLTTB lies between that of MinMax and LTTB. In contrast, our ILTS can surpass LTTB and achieve superior visual quality.

7.2 Extrema-based Sampling

Uniform, MinMax, and M4 [27] are extrema-based methods as they select either horizontal extrema (Uniform), vertical extrema (MinMax), or both (M4) from each bucket. M4 is designed for perfect binary line chart visualization, ideally requiring the sampling buckets to align with their corresponding pixel column time intervals.

OM3 [37], inspired by M4, proposes a multi-level tree structured representation for time series. It traverses the tree layer by layer to enable progressive interactive visualization, pruning redundant inter-column line segments along the way. However, OM3 requires a specialized plotting client and does not accommodate non-uniform timestamps.

MinMaxCache [31] focuses on locality-based exploration, using caching for interactive visualization. It calculates the maximum potential pixel errors and dynamically adjusts the granularity of cached min-max aggregates based on the error bound.

When it comes to accuracy guarantees for visualizations, M4 ensures error-free visualizations by modeling the rendering semantics of line rasterization as query-level aggregation. OM3 has no accuracy guarantee for intermediate visualizations but can eventually reach error-free visualizations in a progressive way. MinMaxCache provides approximate visualizations with user-controllable pixel error-bound guarantees. Although ILTS does not provide error guarantees, our ILTSCache performs comparably to MinMaxCache, as shown in Figure 18.

7.3 Segment-based Sampling

Feasible Space Window (FSW) [30] relaxes the segmentation trigger condition from detecting when a point falls outside the feasible space to waiting until the space becomes empty. Sim-Piece [28] is one of the most recent works on Piecewise Linear Approximation (PLA). It can achieve a high compression ratio even under a small error threshold. However, as the experiments show, Sim-Piece does not produce coherent visualizations due to the disjoint segments.

8 Conclusion

In this paper, we study the area-based Largest Triangle Sampling for time series visualization. We propose ILTS, a novel Iterative Largest Triangle Sampling algorithm enhanced with in-database convex hull acceleration. This algorithm progressively refines the sampling results by bidirectionally propagating the influence of points—heuristically forward within each iteration and iteratively backward between iterations. For better query performance, we improve the algorithm to prune redundant sampling in a bucket by identifying unchanged results. Moreover, we utilize precomputed convex hulls to prune the search space for the local optimal point, significantly reducing unnecessary traversal of points and I/O of disk pages. Experiments show that ILTS achieves increased visual quality and these acceleration techniques significantly improve the query efficiency. For future work, we plan to explore dynamic bucketing, evaluating its strengths and limitations in the context of visualization tasks as well as database querying. Dynamic bucket-based sampling is a double-edged sword, in that it could use fewer sampled points for low-variation data but may also cause overly dense buckets within a single pixel column.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China (62021002, 62232005, 92267203, 62072265), the National Key Research and Development Plan (2021YFB3300500), Beijing National Research Center for Information Science and Technology (BNR2025RC01011), and Beijing Key Laboratory of Industrial Big Data System and Application. Shaoxu Song (https://sxsong.github.io/) is the corresponding author.

References

[1] [n. d.]. https://en.wikipedia.org/wiki/Triangle_wave.
[2] [n. d.]. https://github.com/apache/iotdb/tree/research/LTS-visualization.
[3] [n. d.]. https://github.com/LeiRui/vis-triangle.
[4] [n. d.]. https://github.com/LeiRui/vis-triangle/blob/main/supplement.pdf.
[5] [n. d.]. https://scikit-image.org/docs/stable/api/skimage.metrics.html.
[6] Rakesh Agrawal, Christos Faloutsos, and Arun N. Swami. 1993. Efficient Similarity Search In Sequence Databases. In *Foundations of Data Organization and Algorithms, 4th International Conference, FODO'93, Chicago, Illinois, USA, October 13-15, 1993, Proceedings (Lecture Notes in Computer Science, Vol. 730)*, David B. Lomet (Ed.). Springer, 69–84. doi:10.1007/3-540-57301-1_5
[7] Wolfgang Aigner, Silvia Miksch, Heidrun Schumann, and Christian Tominski. 2011. *Visualization of Time-Oriented Data.* Springer. doi:10.1007/978-0-85729-079-3
[8] Alex M. Andrew. 1979. Another Efficient Algorithm for Convex Hulls in Two Dimensions. *Inf. Process. Lett.* 9, 5 (1979), 216–219. doi:10.1016/0020-0190(79)90072-3
[9] Fred Attneave. 1954. Some Informational Aspects of Visual Perception. *Psychological review* 61, 3 (1954), 183.
[10] Joseph Azar, Abdallah Makhoul, Mahmoud Barhamgi, and Raphaël Couturier. 2019. An energy efficient IoT data compression approach for edge machine learning. *Future Gener. Comput. Syst.* 96 (2019), 168–175. doi:10.1016/J.FUTURE.2019.02.005
[11] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. 1996. The Quickhull Algorithm for Convex Hulls. *ACM Trans. Math. Softw.* 22, 4 (1996), 469–483. doi:10.1145/235815.235821
[12] Matthew Boyd. 2016. NIST Weather Station for Photovoltaic and Building System Research. doi:10.6028/NIST.TN.1913
[13] Timothy M. Chan. 1996. Optimal Output-Sensitive Convex Hull Algorithms in Two and Three Dimensions. *Discret. Comput. Geom.* 16, 4 (1996), 361–368. doi:10.1007/BF02712873
[14] Hoang Anh Dau, Anthony J. Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn J. Keogh. 2019. The UCR time series archive. *IEEE CAA J. Autom. Sinica* 6, 6 (2019), 1293–1305. doi:10.1109/JAS.2019.1911747
[15] Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, Yanping Chen, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, Gustavo Batista, and Hexagon-ML. 2019. The UCR Time Series Classification Archive. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.
[16] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. 2008. *Computational geometry: algorithms and applications, 3rd Edition.* Springer. https://www.worldcat.org/oclc/227584184
[17] Janez Demsar. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *J. Mach. Learn. Res.* 7 (2006), 1–30. https://jmlr.org/papers/v7/demsar06a.html
[18] Jeroen Van Der Donckt, Jonas Van Der Donckt, Michaël Rademaker, and Sofie Van Hoecke. 2023. MinMaxLTTB: Leveraging MinMax-Preselection to Scale LTTB. In *2023 IEEE Visualization and Visual Analytics (VIS), Melbourne, Australia, October 21-27, 2023.* IEEE, 21–25. doi:10.1109/VIS54172.2023.00013
[19] Jonas Van Der Donckt, M. Jeroen Van Der Donckt, Michaël Rademaker, and Sofie Van Hoecke. 2023. Data Point Selection for Line Chart Visualization: Methodological Assessment and Evidence-Based Guidelines. *CoRR* abs/2304.00900 (2023). doi:10.48550/ARXIV.2304.00900 arXiv:2304.00900
[20] Frank Eichinger, Pavel Efros, Stamatis Karnouskos, and Klemens Böhm. 2015. A time-series compression technique and its application to the smart grid. *VLDB J.* 24, 2 (2015), 193–218. doi:10.1007/S00778-014-0368-8
[21] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. 2019. Deep learning for time series classification: a review. *Data Min. Knowl. Discov.* 33, 4 (2019), 917–963. doi:10.1007/S10618-019-00619-1
[22] Abdur Rahim Mohammad Forkan, Yong-Bin Kang, Felip Martí Carrillo, Abhik Banerjee, Chris McCarthy, Hadi Ghaderi, Breno G. S. Costa, Anas Dawod, Dimitrios Georgakopoulos, and Prem Prakash Jayaraman. 2024. AIoT-CitySense: AI and IoT-Driven City-Scale Sensing for Roadside Infrastructure Maintenance. *Data Sci. Eng.* 9, 1 (2024), 26–40. doi:10.1007/S41019-023-00236-5
[23] Tak-Chung Fu, Korris Fu-Lai Chung, Robert Wing Pong Luk, and Chak-man Ng. 2008. Representing financial time series based on data point importance. *Eng. Appl. Artif. Intell.* 21, 2 (2008), 277–300. doi:10.1016/J.ENGAPPAI.2007.04.009
[24] Ronald L. Graham. 1972. An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. *Inf. Process. Lett.* 1, 4 (1972), 132–133. doi:10.1016/0020-0190(72)90045-2
[25] Ray A. Jarvis. 1973. On the Identification of the Convex Hull of a Finite Set of Points in the Plane. *Inf. Process. Lett.* 2, 1 (1973), 18–21. doi:10.1016/0020-0190(73)90020-3
[26] Ian T. Jolliffe. 1986. *Principal Component Analysis.* Springer. doi:10.1007/978-1-4757-1904-8

[27] Uwe Jugel, Zbigniew Jerzak, Gregor Hackenbroich, and Volker Markl. 2014. M4: A Visualization-Oriented Time Series Data Aggregation. *Proc. VLDB Endow.* 7, 10 (2014), 797–808. doi:10.14778/2732951.2732953

[28] Xenophon Kitsios, Panagiotis Liakos, Katia Papakonstantinopoulou, and Yannis Kotidis. 2023. Sim-Piece: Highly Accurate Piecewise Linear Approximation through Similar Segment Merging. *Proc. VLDB Endow.* 16, 8 (2023), 1910–1922. doi:10.14778/3594512.3594521

[29] Zhilin Li. 1995. An Examination of Algorithms for the Detection of Critical Points on Digital Cartographic Lines. *The Cartographic Journal* 32, 2 (1995), 121–125. doi:10.1179/caj.1995.32.2.121

[30] Xiaoyan Liu, Zhenjiang Lin, and Huaiqing Wang. 2008. Novel Online Methods for Time Series Segmentation. *IEEE Trans. Knowl. Data Eng.* 20, 12 (2008), 1616–1626. doi:10.1109/TKDE.2008.29

[31] Stavros Maroulis, Vassilis Stamatopoulos, George Papastefanatos, and Manolis Terrovitis. 2024. Visualization-aware Time Series Min-Max Caching with Error Bound Guarantees. *Proc. VLDB Endow.* 17, 8 (2024), 2091–2103. https://www.vldb.org/pvldb/vol17/p2091-maroulis.pdf

[32] Manuel Milling, Shuo Liu, Andreas Triantafyllopoulos, Ilhan Aslan, and Björn W. Schuller. 2024. Audio Enhancement for Computer Audition - An Iterative Training Paradigm Using Sample Importance. *J. Comput. Sci. Technol.* 39, 4 (2024), 895–911. doi:10.1007/S11390-024-2934-X

[33] Hanan Samet. 2006. *Foundations of multidimensional and metric data structures.* Academic Press.

[34] Sveinn Steinarsson. 2013. *Downsampling time series for visual representation.* Master's thesis. University of Iceland.

[35] Mahes Visvalingam and J. Duncan Whyatt. 1993. Line Generalisation by Repeated Elimination of Points. *Cartographic Journal* 30 (1993), 46–51. https://api.semanticscholar.org/CorpusID:54049050

[36] Chen Wang, Jialin Qiao, Xiangdong Huang, Shaoxu Song, Haonan Hou, Tian Jiang, Lei Rui, Jianmin Wang, and Jiaguang Sun. 2023. Apache IoTDB: A Time Series Database for IoT Applications. *Proc. ACM Manag. Data* 1, 2 (2023), 195:1–195:27. doi:10.1145/3589775

[37] Yunhai Wang, Yuchun Wang, Xin Chen, Yue Zhao, Fan Zhang, Eugene Wu, Chi-Wing Fu, and Xiaohui Yu. 2023. OM3: An Ordered Multi-level Min-Max Representation for Interactive Progressive Visualization of Time Series. *Proc. ACM Manag. Data* 1, 2 (2023), 145:1–145:24. doi:10.1145/3589290

[38] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Process.* 13, 4 (2004), 600–612. doi:10.1109/TIP.2003.819861

Received July 2024; revised September 2024; accepted November 2024