A COMPLEXITY ANALYSIS OF DP

We analyze below that the DP (Dynamic Programming) solution (Algorithm 1) for the ADMIN (Areal Difference MINimization) problem has a time complexity of $O(n^3)$ and a space complexity of $O(n^2)$.

Time Complexity. Let us first consider the computation of the $ad2$ table (Line 1). Computing $AD(T[1:n], \{p_1, p_n\})$ has a time complexity of $O(n)$, which relies on the Shoelace formula by Definition 1. Computing the entire $ad2$ table involves summing the time complexities for each row, which can be expressed as: $(1+\cdots+n)+(1+\cdots+(n-1))+\cdots+(1+2)+1 = \sum_{i=1}^{n} i\cdot(n-i+1) = \frac{n(n+1)(n+2)}{6}$. Thus, the total time complexity for computing the $ad2$ table is $O(n^3)$. It is important to note that, due to the nature of areal difference computation, traditional techniques, such as cumulative sums for fast partial sum computation, cannot be applied in the same way. Consequently, each $ad2[r][i]$ must be computed from scratch. Next, from Line 4 to Line 11, the time complexity is $O(m\cdot n^2)$, which can be easily derived from the three nested loops. In the backward tracing phase (from Line 12 to Line 17), the time complexity is $O(m)$. To sum up, the overall time complexity of DP for ADMIN is $O(n^3)$, dominated by the $ad2$ table computation.

Space Complexity. The space complexity is $O(n^2)$, dominated by the size of the $ad2$ table. Since m is usually much smaller than n, the size of the dp or $path$ tables is smaller than that of $ad2$.

B COMPLEXITY ANALYSIS OF EBUG

In this section, we analyze the time complexity of the eBUG sampling algorithm (Algorithm 2), which mainly consists of two parts: the complexity of significance calculation and the complexity of maintaining the min-heap structure.

Given the input data size as n, and the target number of sampled points as m, eBUG requires a bottom-up cumulative elimination of $n-m$ points. Let k $(1\le k\le n-m)$ denote the number of points eliminated in each round. In each round, the eliminated points are logically removed through marking, requiring no adjustment to the heap structure, resulting in a time complexity of $O(1)$. When updating the significance of the left (or right) neighboring points, the total time complexity complexity$_{Sig}$ for calculating point significance across all rounds is a piecewise function of the algorithm parameter e, as shown below.

$$\text{complexity}_{\text{Sig}} = \begin{cases} 3(n-m), \text{ if } e=0, \\ 2\left(\sum_{k=1}^{e}(k+3)+\sum_{e+1}^{n-m}(e+3)\right), \text{ if } 1\le e<n-m, \\ 2\sum_{k=1}^{n-m}(k+3), \text{ if } e\ge n-m. \end{cases}$$
$$= \begin{cases} 3(n-m), \text{ if } e=0, \\ -e^2+e+2(n-m)(e+3), \text{ if } 1\le e<n-m, \\ (n-m)^2+7(n-m), \text{ if } e\ge n-m. \end{cases}$$

The above result is because (1) When $e=0$, each round only requires calculating the area of a triangle, involving computations for three points. (2) When $1\le e<n-m$, the range of k is divided into two parts: $[1,e]$ and $[e+1,n-m]$. For the former, since $k\le e$, each round involves computations for at most $k+3$ points; for the latter, since $k>e$, each round involves computations for at most $e+3$ points. (3) When $e\ge n-m$, the range of k is $[1,n-m]$, and since $k\le e$, each round involves computations for at most $k+3$ points. Finally, when $e\ge 1$, calculating the areal difference requires finding the intersection point and computing the polygon area, involving two traversal processes. Thus, the time complexity is multiplied by a constant factor of 2.

The other steps mainly involve reinserting the updated points into the min-heap and maintaining the heap structure, with this part of the total time complexity being $O((n-m)\log n)$. In summary, the time complexity of the eBUG algorithm for sampling m points from n points is $O((n-m)\cdot(\min(e,n-m)+\log n))$. It can be observed that the algorithm complexity is primarily determined by the algorithm parameter e, where smaller values of e keep the algorithm within the log-linear complexity range, while larger values of e cause the algorithm to degrade to quadratic complexity. Additionally, since the algorithm employs a bottom-up elimination strategy, the larger the number of sampled points m, the fewer the number of points that need to be eliminated (i.e., $n-m$), resulting in higher algorithm efficiency.

C BASELINES

In the comparative experiments, we employed three different settings for the parameter e of **eBUG**: $e=0$, $e=5000$, and $e=\infty$, which correspond to not using eliminated points at all, using up to 5,000 most recently eliminated points, and using all points to calculate the importance of a point, respectively. Note that Visval [26] is a specific case of the eBUG algorithm when $e=0$.

BU-L1/L2/L∞ are approximate algorithms for the vertical distance difference minimization segmentation problem under different norms (L1/L2/L∞), employing a basic bottom-up greedy strategy. The stopping condition for these algorithms can be either reaching a preset number of sampled points or stopping when the minimum merging error exceeds a preset error threshold. **SWAB** (Sliding Window and Bottom-up) combines the aforementioned BU series of bottom-up algorithms with the classical sliding window method. In our experiments, we configured SWAB to be based on the L2 norm and used linear interpolation (i.e., connecting the start and end data points of a segment) as the segment approximation model. The sampled points are the segment endpoints.

FSW (Feasible Space Window) is a heuristic algorithm based on the sliding window method. Under the constraint of the user-specified maximum vertical distance (i.e., y-value difference) error, it aims to maximize the length of the current segment as much as possible. The vertical distance is the distance from the original data point to the sampling line segment in the y-axis direction, while the perpendicular distance is the shortest distance from the point to the line segment. Under the constraint of the maximum perpendicular distance error, **Rdp** (Ramer–Douglas–Peucker) adopts a top-down greedy strategy, recursively selecting the point with the maximum error for segmentation until the error of all points is less than the preset error threshold. Although **MinMax-freq** and **MinMax-width** use different bucketing strategies, both methods sample the points with the minimum and maximum values within each predefined bucket.

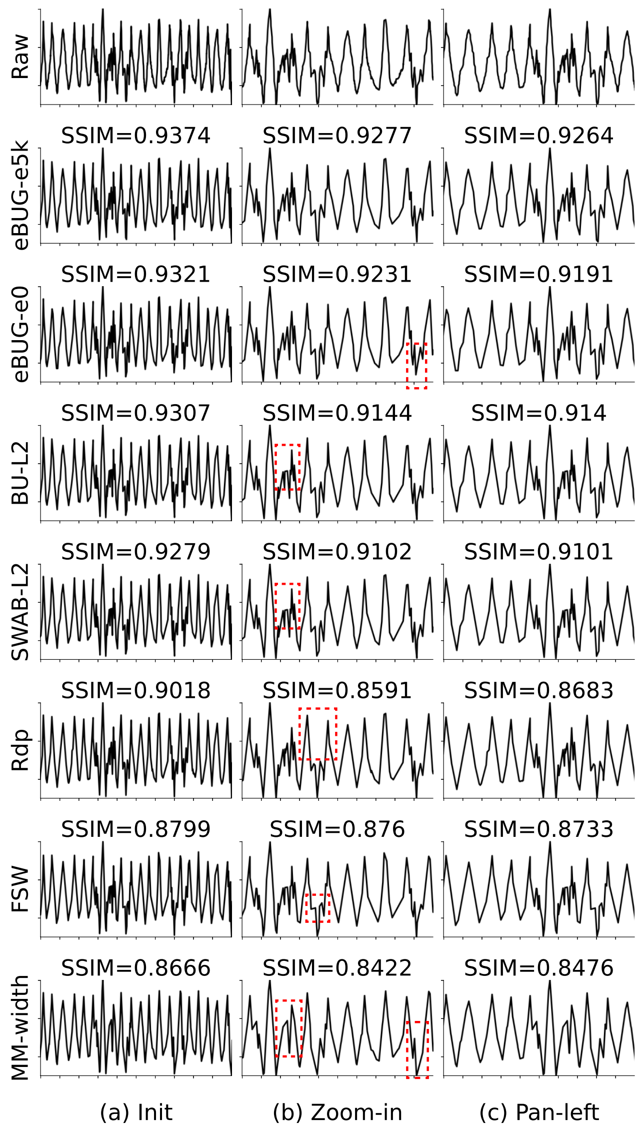

Figure 18: Zooming and Panning Visualization of Sampled Time Series

D E-SSIM METRIC

The sequence of random pan-and-zoom operations is generated following the methodology described in MinMaxCache [20]. Specifically, the initial view is set to the latest 10% of the data interval, and then a number of random pan and zoom operations are performed. Each operation, relative to the previous query view, either enlarges or reduces the time range of the view by no more than 10 times (zoom operation), or shifts the view by a distance of 10-50% of the view's time span (pan operation). Given that the exploration starts from the latest data, the probability of shifting towards historical data (i.e., left) is higher than shifting towards recent data (i.e., right).

Each pan or zoom operation, at its core, maps the data points within a specific time range to the full x-axis range of the canvas. For the y-axis, we automatically scale it based on the raw data points in each visible time range, which is a strategy commonly used by data visualization tools such as Grafana [4]. The formal definition of Exploration SSIM (E-SSIM) is given as follows.

Definition 4 (E-SSIM). *Given the raw time series T, the sampled time series T', and the sequence of visible time ranges $\mathcal{R} = \{[t_s^1, t_e^1], ..., [t_s^c, t_e^c]\}$ corresponding to c pan-and-zoom operations, E-SSIM is defined as*

$$E - SSIM(T, T', \mathcal{R}) = \frac{1}{c}\sum_{i=1}^{c} SSIM(I(T[t_s^i, t_e^i]), I(T'[t_s^i, t_e^i])),$$

where $T[t_s^i, t_e^i] = \{P \mid P \in T, P.t \in [t_s^i, t_e^i]\}$ represents the subsequence of T within the time range $[t_s^i, t_e^i]$, and $I(T[t_s^i, t_e^i])$ denotes the rendered line chart image of $T[t_s^i, t_e^i]$.

Note that the sampled time series T' is retrieved in a single query from the database server. The zooming and panning visualization is then performed locally at the client side based on on T', without further database queries.

E VISUALIZATION COMPARISON

Figure 18 demonstrates the zooming and panning visualization results on the SwedishLeaf time series from the UCR dataset. This sequence reflects the evolutionary process of plant leaf contours over time. From left to right, the figure shows the initial view, zoomed-in view, and left-shifted view in a continuous process. The first row displays the visualization of the original time series, serving as the ground truth. Each subsequent row presents the visualization results of a sampling algorithm along with its SSIM accuracy compared to the original data image, with visual defects highlighted by red dashed rectangles. These algorithms lie on the Pareto front in Figure 12. It can be observed that the eBUG method proposed in this paper effectively preserves the key visual features of the original sequence during interactive visualization: rapid fluctuation regions reflect the detailed changes in leaf edges, while smooth transition regions retain the gradual features of the leaf center.