Pyramid-based Scatterplots Sampling for Progressive and Streaming Data Visualization

Xin Chen, Jian Zhang, Chi-Wing Fu, Jean-Daniel Fekete, Yunhai Wang

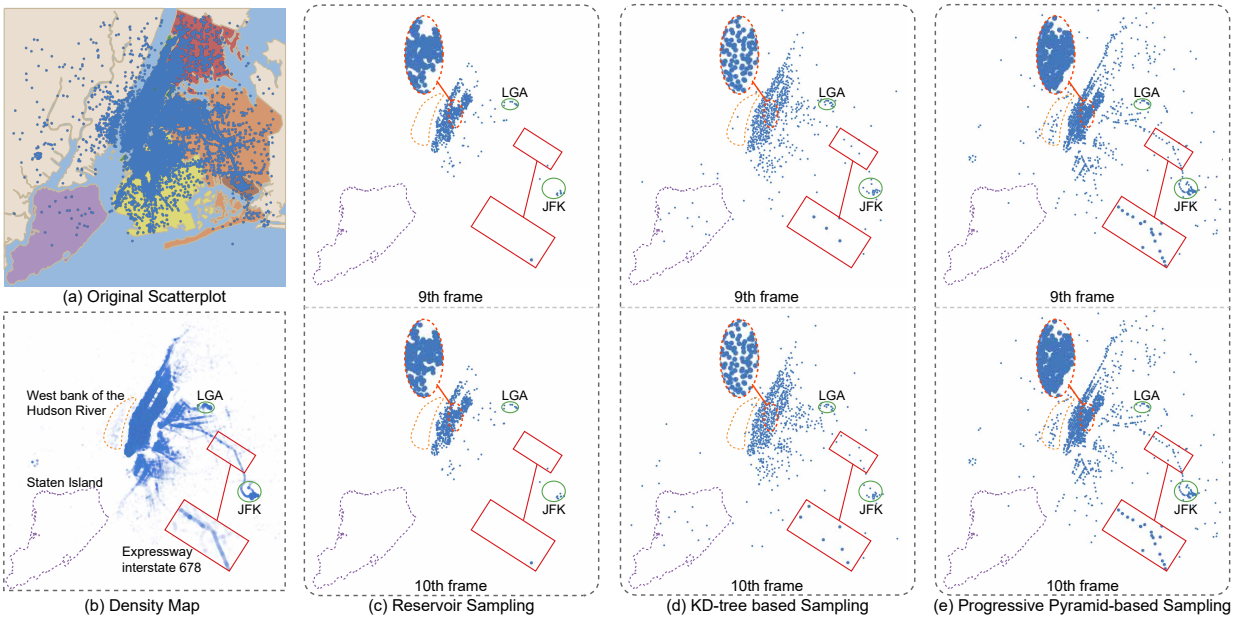


Fig. 1. Different sampling methods for presenting the "New York City TLC Trip Record" data with 2 million data points, which are partitioned into chunks, each of 100k data points. (a) The opaque scatterplot is overlaid on the New York map and rendered as (b) a transparent density map, where some major features are highlighted. (c,d,e) The top and bottom rows show the results of different sampling methods in the 9th and 10th frames, respectively, where each result has around 1K points sampled from the original data chunk. Comparing (c) reservoir sampling [28], (d) KD-tree-based sampling [10], and (e) our progressive pyramid-based sampling, we can find our method more consistent in preserving high-density areas (see the LGA and JFK airports circled in green) and low-density areas (see the Expressway interstate 678 labeled by a red rectangle), while maintaining the density difference between different regions (see the Staten Island and the west bank of the Hudson River labeled by the purple and yellow dashed boxes).

Abstract—We present a pyramid-based scatterplot sampling technique to avoid overplotting and enable progressive and streaming visualization of large data. Our technique is based on a multiresolution pyramid-based decomposition of the underlying density map and makes use of the density values in the pyramid to guide the sampling at each scale for preserving the relative data densities and outliers. We show that our technique is competitive in quality with state-of-the-art methods and runs faster by about an order of magnitude. Also, we have adapted it to deliver *progressive* and *streaming data* visualization by processing the data in chunks and updating the scatterplot areas with visible changes in the density map. A quantitative evaluation shows that our approach generates stable and faithful progressive samples that are comparable to the state-of-the-art method in preserving relative densities and superior to it in keeping outliers and stability when switching frames. We present two case studies that demonstrate the effectiveness of our approach for exploring large data.

Index Terms—Scatterplots, sampling, pyramid, progressive visualization, streaming visualization, scalability, big data

- *X. Chen, Y. Wang are with Shandong University. E-mail:* `{chenxin199634,cloudseawang}@gmail.com`.
- *J. Zhang is with CNIC, CAS, China. E-mail:* `zhangjian@sccas.cn`.
- *C.-W. Fu is with Dept. of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China. E-mail:* `cwfu@cse.cuhk.edu.hk`
- *J.-D. Fekete is with University Paris-Saclay, CNRS, Inria, LISN, France. E-mail:* `Jean-Daniel.Fekete@inria.fr`
- *Y. Wang is the corresponding author.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

1 Introduction

Visualization systems should have low latency to maintain the user's attention, and therefore to support effective data exploration. As data continues to grow in size, progressive visualization has emerged as a promising paradigm to control latency. By partitioning data into chunks, we can transfer data chunks and render a visualization step-by-step over a limited bandwidth channel, allowing analysts to explore intermediate results with a controlled latency. By doing so, computational scalability is ensured for big data visualization. This paradigm is similar to streaming data visualization [13], where the data is continuously generated and the visualization is incrementally updated.

Point-based visualizations are commonly used for showing various forms of data such as bivariate data, multidimensional projections, and points over a map; we refer to them as *scatterplots* for convenience. However, such scatterplots might not work effectively because, for

large data, they suffer from the overplotting issue, i.e., as the data density increases, the number of visual marks overlapping increases as well, thus decreasing the readability of the visualization. Even small data with non-uniform distributions suffer from this issue, limiting the perceptual scalability of scatterplots. In contrast, continuous density fields bypass this issue and can effectively reveal patterns in high-density regions. However, patterns in low-density regions (e.g., outliers) can become less visible [9, 10]. For example, the region labeled by the purple dashed box is void in Fig. 1(b), but it has a few points in the original scatterplot (see Fig. 1(a)).

Several approaches have been proposed to reduce overplotting in scatterplots, among which sampling is widely used. By carefully choosing a subset of the data for display, well-designed sampling methods [5] can faithfully maintain the visual perception of relative data densities and outliers. Some recent methods [9, 10] further attempt to preserve the relative class densities in multi-class scatterplots. All these methods, however, assume that the whole data is preloaded into memory, and thus they may not work well for progressive visualization, where the scatterplots are incrementally updated with new data coming. For example, the sampling results of successive frames generated by a KD-tree-based method [10] might not be temporally coherent; see the yellow lasso regions in the two frames shown in Fig. 1(d). Though the classic reservoir sampling [30] can select a fixed size of random samples from a data stream while preserving temporal coherence, it inherits the drawback of uniform random sampling that it misses the outliers in low-density regions; see the red box in Fig. 1(c).

In this article, we present a new sampling approach for visualizing scatterplots based on a density map pyramid. Our method is faster than the previous methods and is also enhanced to work in a progressive and streaming manner to remain interactive even when applied to very large datasets. In line with the design guidelines of visual sampling [6], progressive visualization [41], and streaming visualization [22], our approach is formulated based on the following three design requirements:

(i) **DR1:** maintaining the relative data densities and outliers on par with the state-of-the-art approaches [9, 10];
(ii) **DR2:** being sufficiently efficient to generate successive visualization frames under human latency limits [29]; and
(iii) **DR3:** preserving temporal coherence between successive visualization frames while capturing the data characteristics.

We first describe a static version of our approach and then a progressive and streaming version. Our approach is based on a pyramid representation [39] of the density map obtained from an input scatterplot, where each scale is sub-sampled to half the width and height of the previous scale. With this representation, the sampling at each level is first performed individually on each local region and then refined between adjacent regions of different parents in the pyramid to further meet DR1. Compared to the state-of-the-art KD-tree-based sampling [10], our pyramid-based sampling not only better preserves local characteristics in low-density regions (the red box in Figs. 1(d,e)) but also runs faster by about an order of magnitude.

For progressive and streaming visualization, in which the data is processed incrementally in chunks, we gradually update the sampled results with each updated density map while preserving the temporal coherence. Like the static pyramid-based sampling, the incremental update has two steps: it works first on each local region, then on adjacent regions to fix them. Both steps are performed level by level in the pyramid. In doing so, the temporal coherence between successive visualizations can be better preserved; see an example in Fig. 1(e).

We evaluate our static approach using 40 large datasets and quantitatively compare the quality and computation time with the existing methods. The results show that our method is comparable to the state-of-the-art sampling methods in preserving relative data density and outliers while running about an order of magnitude faster than them. Further, we evaluate our progressive/streaming sampling by comparing its results to the ones of the reservoir and our static sampling. Our code is available on GitHub [1]. The main contributions of this work are:

- We develop a pyramid-based scatterplot sampling approach that preserves the relative data densities and outliers; it runs an order of magnitude faster and has good quality, as the state-of-the-arts.
- Beyond the existing methods, we can adapt our method for progressiveness and streaming, making it even faster while maintaining the temporal coherence between successive frames; and
- We quantitatively evaluate our sampling results and present two case studies that highlight the usefulness of our approach.

[1] https://github.com/ChenXin360104/ProgressiveWaveletSampling

2 Related Work

Our method relates to scatterplot sampling, as well as streaming and progressive visualizations. Below, we discuss these areas.

2.1 Scatterplot Sampling

Various approaches [19] have been proposed to address overplot in scatterplots, among which sampling is widely used. Sampling aims to faithfully represent the original data by carefully selecting a subset of the data for display. The simplest scheme is random sampling [15, 18], which treats all data points equally. However, it loses important data patterns (e.g., outliers) in low-density regions. To address this issue, a few perception-driven methods have been proposed for preserving various data characteristics such as density [24, 32], and outliers [5, 9, 10]. Recently, Yuan et al. [47] conducted an empirical evaluation of seven representative methods and provided guidance for selecting methods to use in different applications.

Among the existing perception-driven sampling methods, recursive-subdivision-based sampling [10] is the most relevant to our work. It proposes a customized KD-tree method for guiding multi-class scatterplot sampling, which explicitly characterizes the relative data densities, relative class densities, and major outliers. This method is as efficient as the above single-class sampling methods but performs best in balancing between the relative data densities and outliers. However, its customized KD-tree structure has to be built from the root to determine the split axes; doing so is not only very costly but also likely unstable to changes in the data. Thus, this method may not be applicable to streaming or progressive scenarios. In contrast, our proposed pyramid-based sampling method is much faster and allows local updates to preserve temporal coherence between successive frames.

So far, almost all existing sampling methods in visualization [47] assume that the whole data can fit into the memory, which might not be true for the analysis of massive amounts of data. In a similar vein, Provost et al. [35] propose progressive sampling for training a model that uses progressively larger samples until the model accuracy no longer improves. Because of its efficiency, progressive sampling has been used lately for association rules discovery [33], deduplication indexing [17], and mining of frequent items [36].

2.2 Streaming and Progressive Visualization

Streaming data is a continuous flow of data generated from various sources [1]. Visualizing it usually consists of collecting values in a *sliding window* of the data (e.g., the last second or minute), visualizing it, and iteratively collecting the data in the next time window and visualizing it again, frame by frame as an animation (the sliding windows of two adjacent frames can overlap). So a streaming scatterplot shows all the points in the last time window and forgets all the old ones.

Progressive data is generated when e.g., loading a file progressively through a slow network [45]. The points arrive chunk by chunk depending on the loading speed until the whole file has arrived. Each time a new chunk is loaded, the visualization can be updated to reflect the portion of the whole data that has been loaded. Contrary to streaming data, the past is not forgotten and the future is not infinite. Still, the visualization appears and improves frame by frame.

Streaming data visualization has recently gained a lot of attention. Gansner et al. [23] proposed combining the node-link diagram with a map metaphor for incrementally visualizing text streams in real-time. Tanahashi et al. [42] presented an efficient storyline generation algorithm from streaming data that uses the layouts of previous steps to position the incoming data points for preserving temporal coherence. Crnovrsanin et al. [12] developed an incremental layout algorithm for visualizing online dynamic graphs. Fujiwara et al. [22] extended an

incremental PCA [38] for visualizing streaming multidimensional data. All these techniques share the characteristic [13] that the streaming visualization needs to be updated with the incoming data while maintaining the temporal coherence.

Like streaming visualization, progressive visualizations also need to deliver intermediate results with low latency to allow analysts to better control the exploration process [20]. Although the intermediate results might not be accurate, several user studies [4, 21, 48] demonstrated that progressive visualizations show comparable performance to instantaneous ones in insight generation and perform better than the blocking ones. Stolper et al. [41] further show that they can be efficiently combined with visual analytics and propose *progressive visual analytics* (PVA). To provide design guidelines for developers, Schulz et al. [40] introduce an incremental visualization model with partitioned data and visualization operators for facilitating intermediate visualization updates. Angelini et al. [2] systematically characterize the requirements, benefits, and challenges of PVA systems, and Micallef et al. [31] further characterize PVA users in terms of their roles, tasks, and focus of analysis.

Several progressive systems and algorithms have been designed recently using high-dimensional projections visualized with density maps [20, 25, 27, 34, 44]. However, to the best of our knowledge, there are no progressive sampling approaches designed for scatterplots that satisfy the requirements of progressive and streaming visualization, and our approach fills this gap.

3 Preliminary

The goal of scatterplot sampling is to simulate the density distribution of the input data in a display with a limited number of visual pixels. In this section, we define the basic elements used in the design of our approach to achieving this goal. These include three maps (density map, visibility map, and assignment map) and three ratios (density ratio, visibility ratio, and assignment ratio). Before describing them, we denote *data samples* as the input samples from the original data in the scatterplot and *display samples* as the ones assigned to different regions for showing in the final scatterplot.

Maps. For a scatterplot to be shown on a given display, we define:

Density map D**:** $\mathbb{R}^2 \to \mathbb{R}$, where $D(\mathbf{x})$ is the number of data samples located at pixel $\mathbf{x}$.

Visibility map V**:** $\mathbb{R}^2 \to \{0,1\}$, where $V(\mathbf{x})$ is 0 if $D(\mathbf{x})$ is zero, and 1 otherwise. It records whether a density map pixel is empty.

Assignment map A**:** $\mathbb{R}^2 \to \mathbb{N}$. It records the number of samples that have been assigned to each region to eventually generate the display samples of our scatterplot sampling.

Our method first constructs D and V from the input data and then determines the assignment counts in A based on the algorithm in Sect. 4 to produce the final sampled scatterplot.

Ratios. Given two regions Ω_A and Ω_B of the same area on the display, the data *density ratio* between them is defined as:

$$\delta(\Omega_A,\Omega_B) = \frac{\sum_{\mathbf{x}\in\Omega_A} D(\mathbf{x})}{\sum_{\mathbf{x}\in\Omega_B} D(\mathbf{x})}.$$

Also, we can define the visibility ratio ν with visibility map V and the assignment ratio α with assignment map A using a similar formulation (δ stands for *density*, ν for *visibility*, and α for *assignment*). So, to preserve relative densities, $\alpha(\Omega_A,\Omega_B)$ of the assignment map should be close to $\delta(\Omega_A,\Omega_B)$ of the density map. However, simply enforcing this constraint would make most outliers disappear in the final scatterplot because, for outliers located in low-density regions, the assignment ratios of low- to high-density regions would become nearly zero according to the constraint. Since there is no clear definition of an outlier in scatterplots, we preserve outliers by requiring the assignment ratio $\alpha(\Omega_A,\Omega_B)$ in low-density regions to maintain a good balance between the density ratio $\delta(\Omega_A,\Omega_B)$ and the visibility ratio $\nu(\Omega_A,\Omega_B)$ in the final scatterplot. The reason is that the visibility map emphasizes the outliers by treating all the visible pixels equally.

Map Pyramid. We use a pyramid representation for maps D, V, and A. For a 2D map M, its pyramid representation is a sequence of maps $\{M^i\}_{i=0}^{n-1}$, where M^{i-1} is a lower-resolution version of M^i [37]. So, the finest level of the assignment map is similar to a visibility map, in which each pixel value is either zero or one. Yet, the assignment map is for display samples, whereas the visibility map is for data samples. There are multiple types of pyramids. We choose to use a simple one based on non-overlapping 2×2 blocks of pixels [39].

Given a 2^n by 2^n density map M, the pyramid G is built by recursively subdividing the map into four quadrants, similar to a complete quad-tree construction where each non-leaf node is the sum of its four children nodes. Doing so, we can construct a set of maps $G_M = \{M^0,\ldots,M^{n-1}\}$, where M^{n-1} corresponds to input map M and M^0 is the sum of all values in M^{n-1}.

4 Progressive Scatterplot Sampling

After loading a data chunk from the source (e.g., files in a local hard disk or server), our progressive sampling first builds two pyramids from the input density and visibility maps, followed by a static pyramid-based sampling to produce an assignment map that records if a pixel becomes a display sample. For the first data chunk, we directly select the display samples using the assignment map. For subsequent data chunks, we need to first detect the changed regions by comparing the pyramid coefficients of the latest density map with the previous one. For regions with enough changes, we need to update and re-select their display samples. Once the display samples are chosen, the sample selection of the current chunk is complete; we can perform the rendering (of the changed regions only) and then proceed to process the next data chunk.

From an input density map D, we first build the pyramids G_D and G_V based on D and its visibility map V, respectively. In the following, we will describe the detail of the two major stages of our approach, i.e., *pyramid-based sampling* and *incremental update*, and discuss the parameters and their influence.

4.1 Pyramid-based Sampling

Fig. 2 shows the pipeline of our pyramid-based sampling. Once pyramids G_D and G_V are built, a pyramid-based sampling is performed in a hierarchical way to generate an assignment map A^i at each level. To fulfill DR1, we need to identify outliers in the input data, yet the definition of an outlier can be ambiguous. Since low-density regions are often regarded as outliers [8], we sidestep this issue by classifying regions into high- and low-density and placing samples into regions of different densities using different strategies.

We initialize A^0 as the total number of pixels in V; this number is essentially the *sample budget*, the upper bound on the number of display samples filled in the assignment map. Hence, the goal of the top-down sampling process is to distribute this number over the four subregions recursively, while preserving the relative data densities and outliers. Starting from the top-level, i.e., A^0, we classify the four subregions of A^1 into high- and low-density regions, assign samples to each of these regions with different objectives, and then refine the assignment map to further maintain the density ratios between each pair of adjacent regions. This top-down process iteratively refines the assignment maps, such that the number of assigned display samples gradually decreases for preserving the relative densities. The user can specify a certain level (called *stopLevel*) to stop this procedure and directly assign the present sample budget to subregions until the last level based on the visibility ratios. As outlined in Algorithm 1, the sampling at each level involves four major components: *region classification*, *bilateral assignment*, *direct assignment*, and *sampling refinement*. With the final assignment map, we obtain the final sampling result by randomly choosing one data sample for each grid cell.

Region Classification. For each region at level i in pyramid G_D, we compute the density ratios among its four subregion nodes as follows. Suppose the density values in these subregions are $\{d_1,d_2,d_3,d_4\}$, the

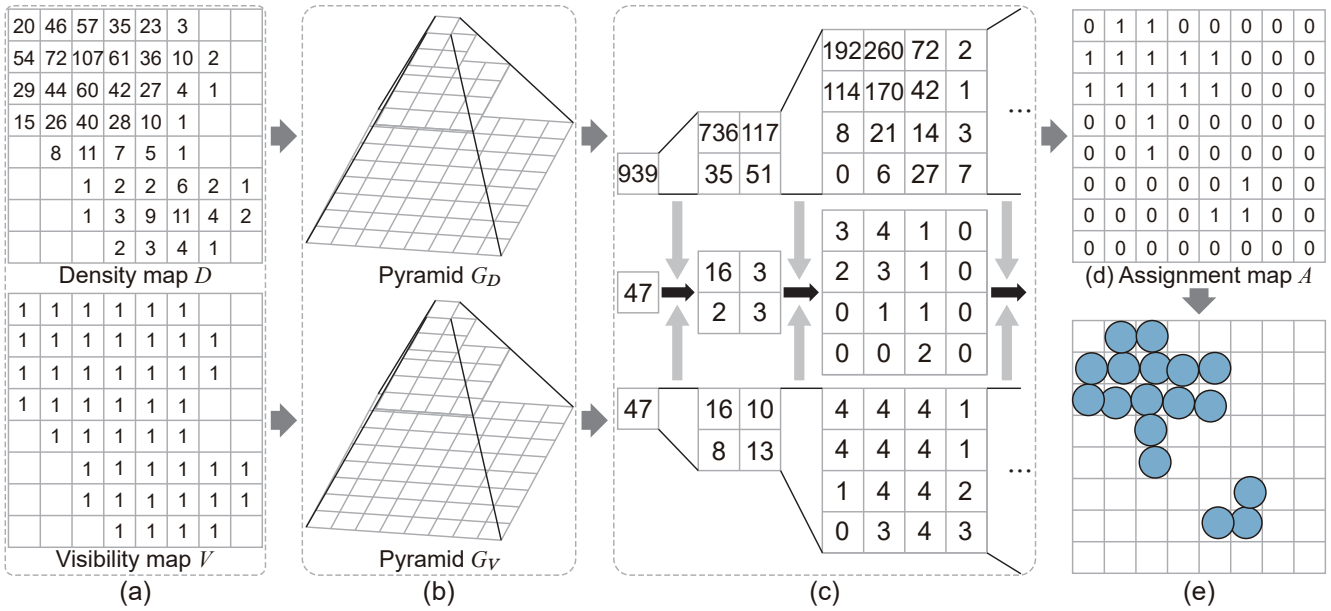


Fig. 2. Overall pipeline of our static pyramid-based sampling: (a) the input map D and its corresponding visibility map V; (b) pyramids G_D and G_V constructed from D and V, respectively; (c) the pyramid-based sampling process that fuses G_D (upper) and G_V (lower) level by level to assign the sample budget to the next level assignment map $[A^0,...,A^{n-1}]$ (middle), where the last level of G_D and G_V equal D and V, respectively; (d) the final assignment map $A = A^{n-1}$; and (e) the visualization with randomly selected samples based on A.

Algorithm 1 Pyramid-based sampling

Input: density map D, visibility map V, and map size $2^l \times 2^l$
Output: assignment map A

```
function PYRAMID-BASED SAMPLING(D, V, l)
    [G_D, G_V] = construct one pyramid for D and another for V
    A^0 = V^0
    i = 0
    while i < l − 1 do
        for each node j in V^i do
            if i < stopLevel then
                [R_high , R_low] = ClassifyRegions(G_D, j)
                AssignToHighDensityRegions(A^i, G_D, G_V, R_high)
                if R_low exists then
                    AssignToLowDensityRegions(A^i, G_D, G_V, R_high, R_low)
                end if
            else
                AssignDirectly(A^i, G_D, G_V)
            end if
        end for
        if i > 0 then
            RefineBoundary(A^{i+1}, G_D, G_V)
        end if
        i++
    end while
return A^{l−1}
end function
```

density ratio δ_k for the kth subregion is defined as

$$\delta_k = d_k / \max_j d_j \in [0,1]. \quad (1)$$

If ϕ_k is smaller than a threshold (say λ), the corresponding region is regarded as a low-density region; otherwise, it is a high-density region. In our experiment, we set λ to 0.1, indicating that the density value of dense regions is 10 times larger than that of sparse regions.

Bilateral Assignment. To address the discrepancy between the number of data samples and the number of visible pixels, we need to choose a proper number of display samples for each region to preserve the relative densities and outliers. Based on the region classification, we fulfill DR1 by assigning the sample budget to high- and low-density regions using two different strategies. For a high-density region Ω_h, we aim to preserve its data density ratio with the adjacent regions as much as possible (i.e., the assignment ratio should be close to the associated data density ratio), while for low-density regions Ω_l, we aim to preserve the outliers in the final scatterplot (i.e., the associated assignment counts should not be always zero to keep *some* of the outliers).

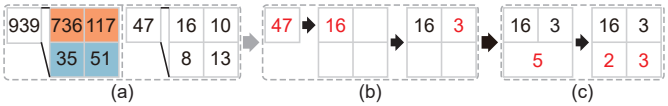


Fig. 3. Illustrating the bilateral assignment process for high- and low-density regions. (a) The density map (left) and visibility map (right) at the root whose four subregions in the density map are classified as high density (in orange) and low density (in blue); (b) two steps to assign the sample budget to high-density subregions: first the subregion with the largest density, then the other high-density subregion(s) based on the density ratio; (c) two steps to assign sample budget to low-density subregions: count the number of display samples in the whole region, then allocate it to each low-density subregion based on the visibility ratio.

Suppose we work with the jth region at the ith level, we assign the sample budget first to its high-density children nodes in four steps:

1. Find the kth subregion with the maximal density among its four children subregions (nodes);
2. Compute the number of display samples a_k for the kth subregion;
3. Compute the sampling ratio $\eta = a_k/d_k$; and
4. Compute the number of display samples for the other high-density regions by multiplying their data densities $d_h (h \neq k)$ by η.

In step 2, a_k is obtained by allocating A^i_j in terms of visible pixels:

$$a_k = \text{ceil}(A^i_j * V^{i+1}_k / V^i_j). \quad (2)$$

where V^i_j is the number of visible pixels in the corresponding region of the jth subregion at level i. Namely, the proportion of assignment for the largest density is based on the visibility map and the rest is proportionally sized by the density map. In doing so, the relative data density is preserved while respecting the sample budget as much as possible. Fig. 3(b) shows a running example.

After the sample assignment is done for the high-density regions Ω_h, we count the number of display samples assigned to Ω_h, say A^{i+1}_h, and then deal with low-density regions Ω_l. To balance the density ratio and outliers in low-density regions, we first compute the density ratio $\delta(\Omega_l, \Omega_h)$ between Ω_l and Ω_h. Likewise, we compute the visibility ratio $\nu(\Omega_l, \Omega_h)$. Then, we find a proper number of samples to assign to

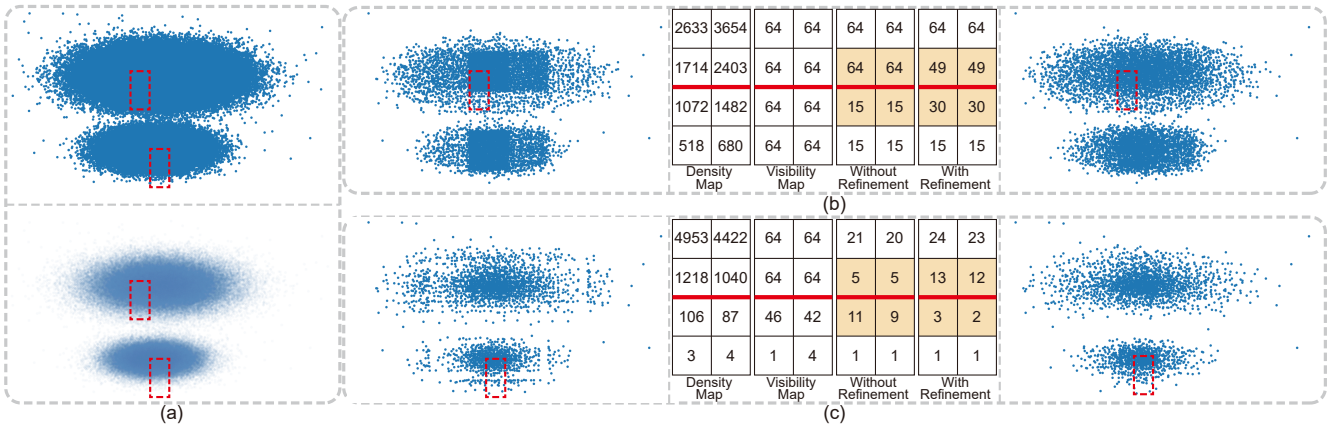


Fig. 4. Illustrating the sampling refinement process that repairs the assignment map to alleviate the blocking artifacts. (a) The input scatterplot (top) made up of two Gaussians and its transparent visualization (bottom); (b,c) Results generated with different *stopLevel*: 5 for Case (i) shown in (b) and 9 for Case (ii) shown in (c). From left to right: the sampled scatterplots generated without refinement (left), the maps related to the highlighted regions (middle), and the scatterplots after the refinement (right), in which the blocking artifacts are almost removed. The red bold lines indicate the boundary between subregions of different parents, whereas the orange backgrounds indicate the subregions involved in the refinement.

Ω_l, say A_l^{i+1}, by minimizing the following objective:

$$\underset{A_l^{i+1}}{\operatorname{argmin}} \left[(1-\omega)\big(\alpha-\delta(\Omega_l,\Omega_h)\big)^2+\omega\big(\alpha-v(\Omega_l,\Omega_h)\big)^2\right], \quad (3)$$

where $\alpha = A_l^{i+1}/A_h^{i+1}$ and ω is the weight between these two terms.

Setting the derivative of Equation 3 with respect to A_l^{i+1} to zero yields the following solution:

$$A_l^{i+1} = A_h^{i+1} \times \big((1-\omega)\delta(\Omega_l,\Omega_h)+\omega v(\Omega_l,\Omega_h)\big), \quad (4)$$

where ω controls how much we want to preserve the data density ratio. Putting ω to 0 solely keeps the high-density regions, so all outliers will disappear. Setting it to 1 keeps all the outliers but will heavily destroy the relative density, and overplotting will persist since too many data samples will be kept. In our experiment, ω is empirically set to 0.2 by default to put more emphasis on preserving the data density ratio. We then allocate the samples into each region in proportion to the number of non-empty pixels in V^{i+1}, like 8 and 13 in Fig. 3(a).

Direct Assignment. If the current level is higher than the given *stopLevel*, we perform a direct assignment for each subregion in this level, where the sample budget is kept during the assignment. Thus, we can adjust the details of high levels and control the number of points in the final result. For the jth subregion at level i, we conduct the assignment in two steps. We first sort its four subregions at level $i+1$ in descending order of the data density ratio δ_k^{i+1}. Based on the ordering, we assign the number of display samples to the kth subregion as

$$A_k^{i+1} = \min(\text{ceil}(A_j^i * V_k^{i+1}/V_j^i), r) \quad (5)$$

where r is the number of unassigned display samples. At last, r is zero and the total sum of display samples assigned to the four subregions equals A_j^i, ensuring no loss in display samples after the assignment.

Sampling Refinement. Applying different sample assignment strategies to different kinds of regions might destroy the relative densities among the boundary regions and introduces blocking artifacts in the final sampling results; see Figs. 4(b,c). After carefully examining the results, we found that these artifacts are caused by the discontinuity between adjacent regions with different parents in the pyramid. To alleviate this issue while avoiding large computational overhead, we propose to refine the assignment map A^i by requiring every pair of adjacent regions of different parents at each level in the tree to preserve the relative data densities as much as possible.

Suppose two adjacent subregions of different parents at level i are l and h, where the density value D_l^i of l is smaller than the density value D_h^i of h. There are two possible cases that violate the data density ratios: (i) $\frac{D_l^i}{D_h^i} > \frac{A_l^i}{A_h^i}$ and (ii) $(D_h^i - D_l^i)(A_h^i - A_l^i) < 0$. Case (i) is caused by the direct assignment, which respects the visibility ratio of each local region, whereas case (ii) is caused by the bilateral assignment for the two different kinds of regions. In both cases, A_l^i and A_h^i need to be fixed. Figs. 4(b,c) show an example for each case, where the scatterplot is generated with different *stopLevel* values. The maps of the highlighted regions in Fig. 4(b) indicate that the density ratios between two adjacent regions become larger in the assignment map, for example, the ratio $1072/1714 \approx 0.625$ is changed to $15/64 \approx 0.234$. In contrast, the relative density is changed in the opposite manner as in Fig. 4(c), for example, the ratio $106/1218 \approx 0.087$ is changed to $11/5 = 2.2$.

For case (i), we re-allocate the total samples $ns = A_h^i + A_l^i$ in these two regions in proportion to the data densities:

$$A_h^i = D_h^i \frac{ns}{D_h^i + D_l^i}. \quad (6)$$

In doing so, the number of assigned display samples in the other (low-density) regions is $ns - A_h^i$.

For case (ii), we re-allocate the total samples in these two regions by balancing density ratio $\delta(\Omega_l,\Omega_h)$ and visibility ratio $v(\Omega_l,\Omega_h)$. By formulating this goal with Equation 3 and differentiating it, we obtain the number of samples nh assigned to the high-density region A_h^i:

$$nh = \frac{ns}{(1-\omega)\frac{D_l^i+D_h^i}{D_h^i} + \omega\frac{V_l^i+V_h^i}{V_h^i}}. \quad (7)$$

where ns is the total number of samples $A_h^i + A_l^i$. Once nh is obtained, we set A_l^i to $ns - nh$.

As shown on the left of Fig. 4(c), the assignment map generated after the refinement better preserves the density ratios. Since the refinement in the coarse level will diffuse to subsequent levels in the pyramid-based sampling process, the artifact can almost be removed in the final sampling result (see the right of Fig. 4(c)).

Time Complexity. As shown in Algorithm 1, the time complexity of our method depends only on the size of the density map $n = 2^l \times 2^l$. The level of the pyramid is therefore l and we need to classify each node (subregion), and then assign and refine the sample budget for each node at each level. Overall, the time complexity of our method is $O(n)$ ignoring the construction time of the density map.

4.2 Incremental Update

When the first data chunk is loaded, we produce the first assignment map A with the static pyramid-based sampling presented in Sect. 4.1. Since A is an approximation of the density map, we need to incrementally update it for each newly arriving data chunk. Given $\bar{D}$ as the latest density map associated with all the data chunks that have arrived so far, we generate a new assignment map $\bar{A}$ by performing a static pyramid-based sampling on $\bar{D}$. Then, we detect the regions from pyramid G_A whose relative densities are changed in $G_{\bar{D}}$ level by level and update these regions in A with the values in $\bar{A}$. Finally, we further refine the updated A to ensure the relative data densities between the updated

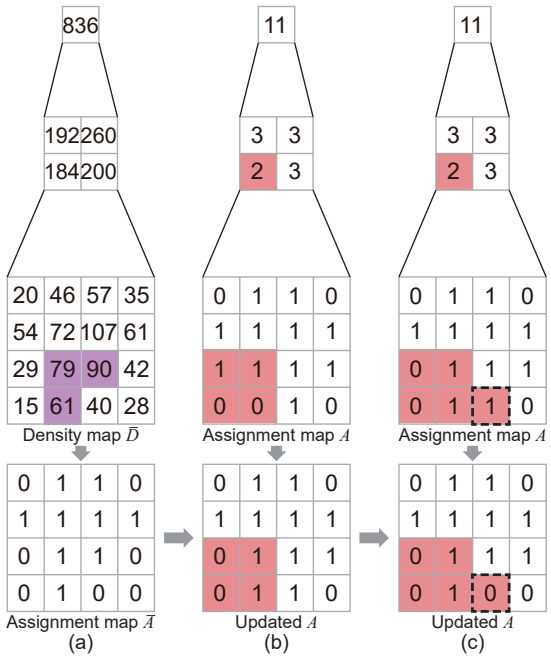


Fig. 5. Illustrating the incremental update process using the top-left 4×4 region in Fig. 2 (d). Our method updates the assignment map for a new data chunk in three steps: (a) update density map $\bar{D}$, where the values of three purple cells are changed from 44, 60, and 26 to 79, 90, and 61, and then compute a new assignment map $\bar{A}$; (b) perform a *local region update* by identifying the changed subregions against the previous assignment map A highlighted in red and updating these regions with the corresponding values in $\bar{A}$; and (c) perform an *adjacent region refinement*, where the cell with the dashed black border in assignment map A is changed and its value is further updated with the one in $\bar{A}$.

regions and their adjacent ones are preserved, during which the regions to be changed are also detected level by level. Fig. 5 illustrates these three steps with the 4×4 region on the top left of Fig. 2(d), where the regions to be updated in A are highlighted in red. In the following, we describe the last two steps in detail.

Local Region Update. Given A and $\bar{D}$, we compute two pyramids for them and detect the regions with the changed relative densities from coarse level to fine level. For every region at the ith level, we check if the density ratio in A^i has a large difference from the one in $\bar{D}^i$. Suppose the node index is j at the $(i-1)$-th level, and A^{i-1}_j and $\bar{D}^{i-1}_j$ are both nonzero, the density ratio difference is defined as

$$\mu^{i-1}_j = \frac{1}{4}\sum_{k=1}^{4} \left| \frac{A^i_{4j+k}}{A^{i-1}_j} - \frac{\bar{D}^i_{4j+k}}{\bar{D}^{i-1}_j} \right|. \tag{8}$$

If μ^{i-1}_j is larger than a given threshold ε, we regard node j at the $(i-1)$-th level as "changed" and stop checking its descendant regions in finer levels. For the node, if A^{i-1}_j or $\bar{D}^{i-1}_j$ becomes zero, we also label its region as "changed." Once the finest level is done, we update all these changed regions in A with the values in $\bar{A}$.

Adjacent Region Refinement. The first step individually refines each local region, so it might not preserve the relative densities between adjacent regions, resulting in visible artifacts. Like the sampling refinement step in the static sampling, we also further check whether the updated regions and their adjacent regions respect the corresponding data density ratio in $\bar{D}$. With the updated A, we re-compute the pyramid again and then find the adjacent regions that need to be further updated from the coarse to fine levels.

Suppose the changed region j and its adjacent region k are at level i, we compute the density ratio difference μ in A^i and $\bar{D}^i$ as

$$\mu = \left| \frac{A^i_j}{A^i_k} - \frac{\bar{D}^i_j}{\bar{D}^i_k} \right|. \tag{9}$$

We mark the adjacent region as "changed" if μ is larger than threshold ε, and stop checking its descendant regions in finer levels. Like local region update, we refine all changed regions in A with the values in $\bar{A}$.

As shown in Fig. 5(a), the arriving data chunk produces three changed regions in the latest density map $\bar{D}$ highlighted in purple. Then, the bottom left region in the second level of the assignment map is labeled as "changed" and then all its subregions in the finer level are updated in Fig. 5(b). Last, the subregion with the black border in Fig. 5(c) is further updated by the adjacent region refinement.

4.3 Parameter Analysis

All the parameters we describe below affect the quality and stability of the sampling, but they can be changed at any time in an interactive environment so the user can tune them according to the data at hand for producing the desired results. More discussions about the influence of parameter choices can be found in the supplemental material.

Density threshold λ & Outlier weight ω. These two parameters jointly affect the number of outliers to be preserved. Parameter λ determines which regions are classified as low density and weight ω determines how many data points in low-density regions we want to keep. As shown in Fig. 6, a large λ leads to more sparse regions (see the orange boxes in Fig. 6(a,b,c)) and a large ω preserves more data points in low- and medium-density regions (see the orange and red boxes in Fig. 6(d,b,e)). When λ and ω are both large, more outliers are preserved, whereas the relative densities may not be correct in some regions. In our experiments, we set them to 0.1 and 0.2, respectively.

StopLevel. Parameter *stopLevel* determines the level of details to preserve, thus affecting the number of samples to keep. As shown in Fig. 6(b,g,h), a small *stopLevel* leads to more details in low-density regions and more samples, while a large *stopLevel* reduces the number of samples to preserve the relative densities in fine level. We suggest setting it to the last level by default and automatically search one level as *stopLevel* where the corresponding total sum of display samples is close to the number of samples specified by users.

Ratio threshold ε. Parameter ε influences the temporal coherence between successive visualizations; see Fig. 7. A large ε tends to keep more samples selected from the previous frames and helps to maintain stability, but might not reflect the relative data densities in the latest frame. In contrast, a low ε better preserves the relative data density but may introduce large changes on the display samples. We empirically set it to 0.25, which works well for most tested data.

5 Evaluation

We implemented our method in C++ and tested it on a PC with an Intel Core i5-4590 3.3GHz CPU and 24GB memory. To confirm that our method can meet our three design requirements (DR1, DR2, and DR3), we performed two quantitative comparisons with the state-of-the-art methods in two settings of static sampling and progressive sampling. Also, we conducted two case studies to demonstrate the effectiveness of our method on real datasets. The full evaluation results, including the screenshots and the corresponding scores of different metrics for each dataset, can be found in the supplemental material.

5.1 Comparative Evaluation of Static Sampling

We compare the sampling results produced from various methods in terms of sampling quality and runtime performance, in which all the scatterplots are displayed in 1600×900 pixels.

Methods. We compare our method with seven sampling methods: random sampling (RS), blue noise sampling (BNS) [11], density-based sampling (DBS) [32], non-uniform sampling (NUS) [5], outlier biased density-based sampling (OBDBS) [46], multi-view z-order sampling (MVZS) [24], and KD-tree-based sampling (KBS) [10]. Though KBS and MVZS are designed for multi-class sampling, we use them on single-class data without multi-class constraints. For all methods, we use the Python implementations provided by Yuan et al. [47] for comparing the sampling quality. We set the parameters of our method to be $\lambda = 0.1$, $\omega = 0.2$, and *stopLevel* to the last level, and then adjust the

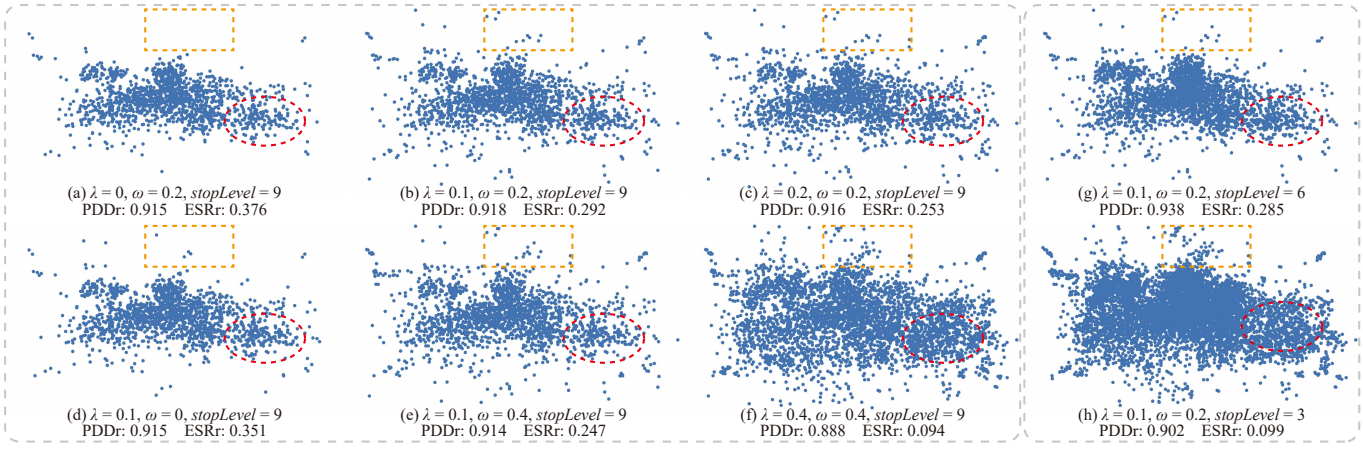


Fig. 6. Parameter analysis on the *Person Activity* dataset [16] with the associated PDDr and ESRr scores (see Section 5). The orange dashed boxes and red dashed circles represent typical low- and medium-density regions, respectively. (a,b,c) A large λ results in more regions classified as low-density regions; (d,b,e) a large ω results in more display samples in low-density regions; (f) when λ and ω are both large, almost all low-density regions in the original scatterplot are kept but the data density ratios cannot be maintained; (b,g,h) decreasing *stopLevel* introduces more display samples; outliers in low-density regions can be shown more clearly but overplotting will have resulted.

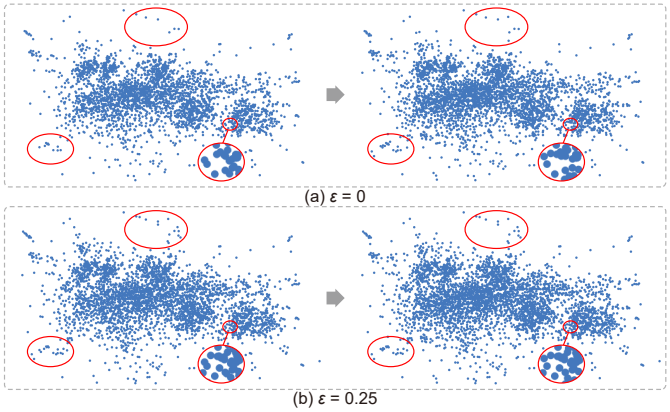


Fig. 7. The scatterplots of the 7th and 8th frames under different ε. The red circles show that larger ε can help prevent unnecessary changes of display samples in low-density regions.

parameters of other methods to generate similar numbers of samples. The details can be found in the supplemental material.

Datasets. For a comprehensive evaluation, we collected 40 datasets with substantial variations in data distributions and data size ranged from 4K to 2M. Among them, 12 synthetic datasets were generated by mixing Gaussian distributions and a uniform distribution, and 28 real datasets were collected from the UCI data repository [16] and Kaggle [26]. The details can be found in the supplemental material.

Metrics. We employ two numerical measures proposed by Bertini and Santucci [5]: *Perceived Data Densities ratio* (PDDr) and *Erased Sample Regions ratio* (ESRr), both based on the division of the pixel display into a set of non-overlapping, equal-sized regions. PDDr measures how much the density ratio is preserved for each pair of regions:

$$\text{PDDr} = \frac{\sum_i \sum_{j<i} \chi_{ij} \sigma(\text{sgn}(D(\Omega_i) - D(\Omega_j)), \text{sgn}(A(\Omega_i) - A(\Omega_j)))}{\sum_i \sum_{j<i} \chi_{ij}}, \quad (10)$$

where $\chi_{ij} = D(\Omega_i) + D(\Omega_j)$ (number of data samples in regions Ω_i and Ω_j), $\text{sgn}(v)$ is a sign function, and $\sigma(v_1, v_2)$ returns one if v_1 equals v_2, otherwise zero. The range of PDDr is $[0,1]$ and a large value indicates better preservation of relative data densities.

ESRr measures the proportion of lost outliers in low-density areas by computing the ratio of void regions caused by the sampling:

$$\text{ESRr} = \frac{\sum_i \sigma(A(\Omega_i), 0)}{N}, \quad (11)$$

where N is the number of non-empty regions in the original scatterplot. ESRr also ranges $[0,1]$ but a small value indicates better outlier preservation. We set the size of the region Ω to 40×40 pixels.

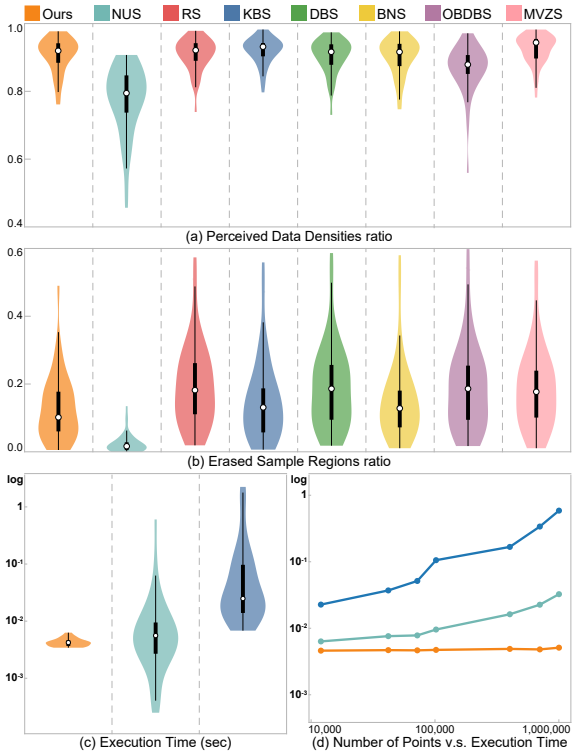


Fig. 8. Results of quantitative comparison. (a,b) These violin plots summarize the values of PDDr (a) and ESRr (b) over all the tested datasets, where a larger PDDr score is better and a smaller ESRr score is better; (c) the violin plot shows the log-scale computational times of three most efficient methods with C++ implementations tested over all the datasets; and (d) the curves show the relationship between the execution time (running the sampling procedure) and the data size of different methods by using synthetic datasets.

Results. Screenshots of the original scatterplots and sampled results generated by all methods on various datasets with complete scores can be found in the supplemental material. The violin plots in Figs. 8(a,b) summarize the PDDr and ESRr scores of each method on all datasets.

Fig. 8(a) shows that PDDr of our method is very close to those of RS, KBS, DBS, BNS, and MVZS. OBDBS is slightly worse than our method, while NUS is the worst. In contrast, NUS performs better than the other methods in ESRr, while our method is ranked as the second, as shown in Fig. 8(b). Though NUS has the lowest ESRr, our method outperforms it in PDDr clearly. This result confirms that our method

balances well the preservation of relative data densities and outliers.

Runtime. We only implemented KBS and NUS in C++ for a fair runtime comparison, since the other methods involve expensive computation and are slower as shown in previous work [10]. Fig. 8(c) summarizes the execution time of all tested datasets for KBS, NUS, and our method. We can see from the violin plot that our method is faster than NUS and even more than ten times faster than KBS on average. Fewer outliers and lower variance indicate that the runtime of our method is rather stable, regardless of the data set size.

To further examine the time performance of our method, we generated a set of synthetic data with a gradually increasing number of data samples. Fig. 8(d) plots the execution time for running different sampling methods on these datasets. We can see that our method is twice faster than NUS and around ten times faster than KBS for the data with more than 100k points. The reason for the almost constant running time of our method is that its time complexity depends only on the resolution of the density map. Note that we did not take into account the pre-processing time such as computing the density map.

5.2 Comparative Evaluation of Progressive Sampling

Competitive analysis [7] is a widely-used approach for evaluating online algorithms, by comparing the performance with an equivalent offline algorithm. In our progressive setting, users will rely on intermediate results to make early decisions and thus we measure the performance of each frame, following the practice in Jo et al. [25].

Method. We conducted a competitive analysis of the progressive version ($\varepsilon = 0.25$) and the static version (without incremental update) of our method, and the reservoir sampling method [28]. Our method uses the same parameter settings as the one in the first experiment.

Data. To learn how our method works in practice, we use two datasets *activity* and *census*, which have different characteristics as shown in Table 1. For *activity* and *census*, their partitioned chunk sizes are 10,000 and 100,000, respectively.

Dataset	# points	# samples	chunk size	variance
activity	164,860	$\sim$ 5,500	10,000	high
census	2,000,000	$\sim$ 2,100	100,000	low

Table 1. Datasets employed in the competitive analysis.

Measures. We hypothesized that the progressive version is more stable than others since there should be fewer changed samples. Thus, we further compute the number of changed points (NCP) between every pair of consecutive frames, besides PDDr and ESRr. For each frame, the PDDr and ESRr scores are based on the comparison between the assignment map and the density map of the whole data.

Results. Figs. 9(a,b) show the NCP values for each frame of these two datasets. In the first frame, all points are newly added and thus its NCP is the highest. Later, all methods quickly converge to stable values but the NCP values of our progressive are smaller than the others. Because of the large density variation, all the methods show more fluctuations on the *activity* dataset, whereas our progressive method is the most stable. We conclude that our progressive version is more stable than the static version and the reservoir sampling.

Figs. 9(c,d) show the PDDr scores of each frame of these two datasets, which have different performances. The PDDr scores of the three methods in the *activity* dataset gradually increase to a value of 0.96. Although the scores of our progressive methods are less than reservoir sampling at the beginning, they quickly converge to similar values. In contrast, the scores of the three methods on the *census* dataset are quite stable in all frames. After examining the data and each sampling result, we found that each data chunk contains similar spatial distribution. Since all scores of our progressive method are around 0.9, it is still good enough to preserve relative data densities.

Figs. 9(e,f) show the ESRr scores of each frame of these two datasets. We can see that our progressive method performs the best, followed by our static method, while reservoir sampling is the worst. The reason for our progressive method performing better than the static method is

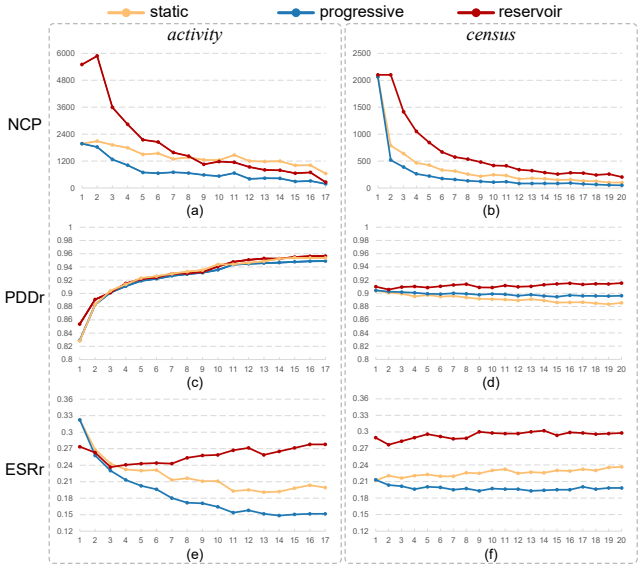


Fig. 9. These line charts show how the number of changed points (a,b), PDDr (c,d), and ESRr (e,f) evolve over iterations for loading the two tested datasets (*activity* and *census*) for sampling by the three methods being compared: static and progressive versions of our method vs. reservoir sampling. Our progressive method is the most stable and makes a good balance between preserving relative data densities and outliers.

that its incremental update step pays more attention to the regions with significant data density changes, thereby encouraging the retention of outliers in low-density regions. Conversely, the static method updates the whole density map and might lose some outliers for better balancing the preservation of relative density and outliers. In this configuration, by chance, retaining the outliers produced a slightly better result transiently. Moreover, the difference in ESRr between our progressive method and reservoir sampling gradually increases to 0.12 for the *activity* dataset, while the difference almost stays the same (0.11) for the *census* dataset. Thus, we conclude that our progressive method also maintains a good balance between preserving relative data densities and outliers.

Runtime. Regarding the time performance, our progressive method is similar to our static method but a bit slower than the reservoir sampling. The average runtime of our method and reservoir sampling for the two datasets are 0.024s vs. 0.013s and 0.15s vs. 0.13s; yet, our method is sufficiently fast for supporting interactive analysis.

Based on the results of the three measures and time performance, we can conclude that our method well satisfies the three design requirements (DR1-DR3) for progressive visualization of large scatterplots.

5.3 Case studies

We conducted case studies with two real-world datasets about astrophysics and stock prices, corresponding to the two settings of progressive and streaming visualizations, respectively. Note that the density map is accumulative in progressive visualization, and it is based on the data within the current time window in streaming visualization.

Hertzsprung Russell diagram. Astronomers often study stellar evolution by exploring the relationship between the star temperature and observed luminosity using scatterplots; such plots are commonly referred to as the *Hertzsprung-Russell diagram* [14]. Here, we used a subset of the Gaia Data Release 2 [43] collected by the European Science Agency. The subset contains 1,322,033 stars within 200 parsecs from The Sun; two typical attributes of the stars are luminosity and temperature. Fig. 10(a) shows the input scatterplot (top) and the associated density map (bottom); each pair of the overlaid red and green lines corresponds to an unresolved binary system of two identical stars [3].

By partitioning the data into chunks, each of 100,000 stars, Figs. 10(b,c) show four frames in progressive visualization generated by reservoir sampling and by our method with $stopLevel = 6$ (10 for the total), respectively. From the 8th frame, the patterns highlighted in

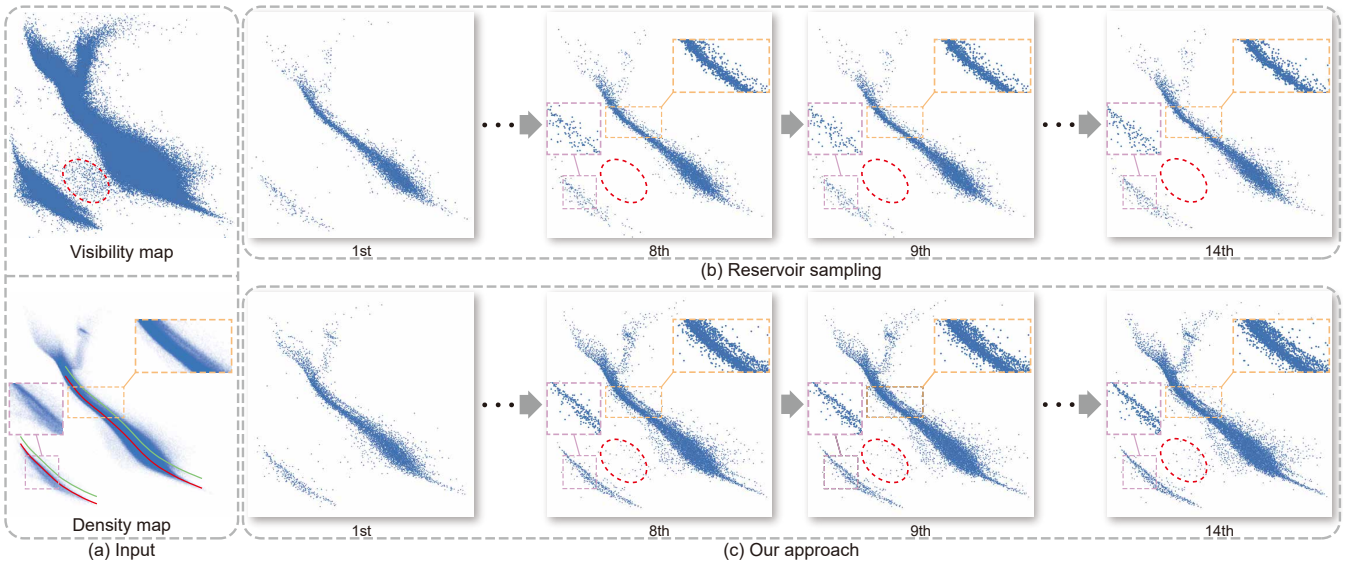


Fig. 10. Progressive sampling on the Gaia Data Release 2. (a) the scatterplot of the input data (top) and the density map (bottom); (b,c) the results of the intermediate frames generated by reservoir sampling (b) and our method (c).

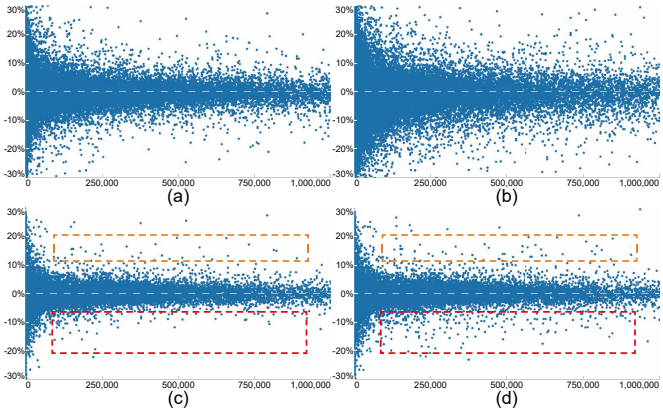


Fig. 11. Scatterplots that show the relationship between stock volume (horizontal) and stock percentage change (vertical) for two different time ranges: before the Sep. 11 attacks (left column) and the whole Sep. 2001 (right column). (a,b) the overplotted scatterplots of the original data; and (c,d) streaming visualization results of our method from (a,b), showing that our method can produce faithful visualizations.

Fig. 10(c) gradually become clear. Among them, the patterns in the purple and yellow boxes have similar structures as the ones at the bottom of Fig. 10(a), while the outliers in the red circle are better preserved. In contrast, all these structures cannot be clearly revealed in Fig. 10(b). Compared to the 8th and 9th frames, the number of changed points in both methods is small. Overall, our method can preserve relative data densities and outliers well, while maintaining the temporal coherence.

Since reservoir sampling is not designed for progressive visualization, we admit that the comparison is not entirely fair, given that our method supports progressive visualization. Yet, we would like to highlight that no progressive visualization methods have been designed for scatterplots sampling.

Stock Prices. We used the historical stock market datasets from Kaggle [26] from Jan. 1, 1996 to Aug. 7, 2020, and explored the relationship between the stock volume and stock percentage change. To simulate the setting of streaming data visualization, we set the time step as one day and the time window as 30 days. In doing so, our method loads the data chunk containing records of the next day for the incremental update while discarding the records of the day one month earlier. To reveal the major patterns, we remove the data items with a stock volume larger than 1 million or with stock percentage change larger than 30%.

To see how the Sep. 11 attacks affected the stock market, we compared two scatterplots (see Figs. 11(a,b)) generated from the dataset for two different time-ranges: (i) from Aug. 11, 2001, to Sep. 10, 2001 (see Fig. 11(a)) and (ii) the whole Sep. 2001 (see Fig. 11(b)). We can see that the stock percentage change has much larger variations in Fig. 11(b) than the ones shown in Fig. 11(a) and more points with negative stock changes can be observed than those with positive changes. Based on these observations, we see that the Sep. 11 attacks resulted in a negative effect and many stocks show large fluctuations. However, the streaming sampling results in Figs. 11(c,d) shows that the major trend is almost preserved in both ranges. Further comparing the samples in the red boxes reveals more samples with negative stock change and large volumes in Sep. 2001. After checking the density maps shown in the supplemental material, we conclude that our streaming sampling method produces faithful visualizations.

6 Conclusion

In this article, we proposed a scatterplot sampling method based on a pyramid-based decomposition of the density map for progressive and streaming data visualizations. In the static setting, it can perform similarly to state-of-the-art methods in maintaining relative densities and preserving outliers but it is faster. Based on a pyramid representation of the density map, our progressive sampling is achieved by first performing a local region-based sampling, then a refinement between adjacent regions and an incremental update for progressive and streaming visualization. The quantitative evaluation and case studies demonstrate the effectiveness of our method for exploring large and streaming data.

Our approach still has some limitations. First, our bilateral assignment may still produce blocking artifacts; we will develop a smooth nonlinear function to alleviate this issue. Second, it is hard to find proper parameters (λ, ω, and *stopLevel*) showing meaningful patterns, while improper ones might lead to bad visualizations. For example, the PDDr scores in Figs. 6(f,g) are 0.888 and 0.938, while the ESRr scores in Figs. 6(a,f) are 0.376 and 0.094. In the future, we will explore automatic methods for setting such parameters for better balancing the preservation of relative data densities and outliers. Last, we plan to conduct a user study as Yuan et al. [47] for investigating how our results align with human perception in both static and progressive settings. In the future, we will extend our approach to deal with multi-class datasets.

Acknowledgments

This work is supported by the grants of the NSFC (61772315, 61861136012), the Open Project Program of State Key Laboratory of Virtual Reality Technology and Systems, Beihang University (No.VRLAB2020C08), and the CAS grant (GJHZ1862).

REFERENCES

[1] T. Akidau, S. Chernyak, and R. Lax. *Streaming Systems: The What, Where, When, and How of Large-Scale Data Processing*. O'Reilly Media, Inc., 2018.

[2] M. Angelini, G. Santucci, H. Schumann, and H.-J. Schulz. A Review and Characterization of Progressive Visual Analytics. *Informatics*, 5(3):31, 2018. doi: 10.3390/informatics5030031

[3] C. Babusiaux, F. van Leeuwen, M. Barstow, C. Jordi, A. Vallenari, D. Bossini, A. Bressan, T. Cantat-Gaudin, M. Van Leeuwen, A. Brown, et al. Gaia Data Release 2 - Observational Hertzsprung-Russell diagrams. *Astronomy & Astrophysics*, 616:A10, 2018. doi: 10.1051/0004-6361/201832843

[4] S. K. Badam, N. Elmqvist, and J.-D. Fekete. Steering the Craft: UI Elements and Visualizations for Supporting Progressive Visual Analytics. In *Computer Graphics Forum*, vol. 36, pp. 491–502. Wiley Online Library, 2017. doi: 10.1111/cgf.13205

[5] E. Bertini and G. Santucci. By Chance is Not Enough: Preserving Relative Density through Nonuniform Sampling. In *Proceedings of the International Conference on Information Visualisation*, pp. 622–629, 2004. doi: 10.1109/IV.2004.1320207

[6] E. Bertini and G. Santucci. Give Chance a Chance: Modeling Density to Enhance Scatter Plot Quality through Random Data Sampling. *Information Visualization*, 5(2):95–110, 2006. doi: 10.1057/palgrave.ivs.9500122

[7] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 2005.

[8] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying Density-Based Local Outliers. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, vol. 29, p. 93–104. Association for Computing Machinery, New York, NY, USA, May 2000. doi: 10.1145/335191.335388

[9] H. Chen, W. Chen, H. Mei, Z. Liu, K. Zhou, W. Chen, W. Gu, and K.-L. Ma. Visual Abstraction and Exploration of Multi-class Scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1683–1692, 2014. doi: 10.1109/TVCG.2014.2346594

[10] X. Chen, T. Ge, J. Zhang, B. Chen, C.-W. Fu, O. Deussen, and Y. Wang. A Recursive Subdivision Technique for Sampling Multi-class Scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):729–738, Jan 2020. doi: 10.1109/TVCG.2019.2934541

[11] R. L. Cook. Stochastic Sampling in Computer Graphics. *ACM Transactions on Graphics*, 5(1):51–72, 1986. doi: 10.1145/7529.8927

[12] T. Crnovrsanin, J. Chu, and K.-L. Ma. An Incremental Layout Method for Visualizing Online Dynamic Graphs. *Journal of Graph Algorithms and Applications*, 21(1):55–80, 2017. doi: 10.1007/978-3-319-27261-0_2

[13] A. Dasgupta, D. L. Arendt, L. R. Franklin, P. C. Wong, and K. A. Cook. Human Factors in Streaming Data Analysis: Challenges and Opportunities for Information Visualization. In *Computer Graphics Forum*, vol. 37, pp. 254–272. Wiley Online Library, 2018. doi: 10.1111/cgf.13264

[14] C. de Jager, H. Nieuwenhuijzen, and K. A. van der Hucht. Mass loss rates in the Hertzsprung-Russell diagram. *Astronomy and Astrophysics Supplement Series*, 72:259–289, 1988.

[15] A. Dix and G. Ellis. *By Chance* Enhancing Interaction with Large Data Sets through Statistical Sampling. In *Proceedings of the International Conference on Information Visualisation*, pp. 167–176. ACM, 2002. doi: 10.1145/1556262.1556289

[16] D. Dua and C. Graff. UCI Machine Learning Repository. `https://archive.ics.uci.edu/ml`, 2017.

[17] P. Efstathopoulos, F. Guo, and D. Shah. Progressive sampling for deduplication indexing, Nov. 13 2012. US Patent 8,311,964.

[18] G. Ellis and A. Dix. Density Control Through Random Sampling: an Architectural Perspective. In *Proceedings of the International Conference on Information Visualisation*, pp. 82–90, 2002. doi: 10.1109/IV.2002.1028760

[19] G. Ellis and A. Dix. A Taxonomy of Clutter Reduction for Information Visualisation. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1216–1223, 2007. doi: 10.1109/TVCG.2007.70535

[20] J.-D. Fekete and R. Primet. Progressive Analytics: A Computation Paradigm for Exploratory Data Analysis. *arXiv preprint arXiv:1607.05162*, 2016.

[21] D. Fisher, I. Popov, S. Drucker, and M. Schraefel. Trust Me, I'm Partially Right: Incremental Visualization Lets Analysts Explore Large Datasets Faster. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1673–1682, 2012. doi: 10.1145/2207676.2208294

[22] T. Fujiwara, J.-K. Chou, S. Shilpika, P. Xu, L. Ren, and K.-L. Ma. An Incremental Dimensionality Reduction Method for Visualizing Streaming Multidimensional Data. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):418–428, 2019. doi: 10.1109/TVCG.2019.2934433

[23] E. R. Gansner, Y. Hu, and S. North. Interactive Visualization of Streaming Text Data with Dynamic Maps. *Journal of Graph Algorithms and Applications*, 17(4):515—-540. doi: 10.7155/JGAA.00302

[24] R. Hu, T. Sha, O. van Kaick, O. Deussen, and H. Huang. Data Sampling in Multi-view and Multi-class Scatterplots via Set Cover Optimization. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):739–748, 2019. doi: 10.1109/TVCG.2019.2934799

[25] J. Jo, J. Seo, and J.-D. Fekete. PANENE: A Progressive Algorithm for Indexing and Querying Approximate k-Nearest Neighbors. *IEEE Transactions on Visualization and Computer Graphics*, 26(2):1347–1360, 2020. doi: 10.1109/TVCG.2018.2869149

[26] Kaggle Inc. Kaggle. `https://www.kaggle.com/`.

[27] J. K. Li and K.-L. Ma. P5: Portable Progressive Parallel Processing Pipelines for Interactive Data Analysis and Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):1151–1160, 2019. doi: 10.1109/TVCG.2019.2934537

[28] K.-H. Li. Reservoir-Sampling Algorithms of Time Complexity O(n(1 + log(N/n))). *ACM Transactions on Mathematical Software*, 20(4):481–493, 1994. doi: 10.1145/198429.198435

[29] Z. Liu and J. Heer. The Effects of Interactive Latency on Exploratory Visual Analysis. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2122–2131, 2014. doi: 10.1109/TVCG.2014.2346452

[30] A. I. McLeod and D. R. Bellhouse. A Convenient Algorithm for Drawing a Simple Random Sample. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 32(2):182–184, 1983. doi: 10.2307/2347297

[31] L. Micallef, H.-J. Schulz, M. Angelini, M. Aupetit, R. Chang, J. Kohlhammer, A. Perer, and G. Santucci. The Human User in Progressive Visual Analytics. In *EuroVis (Short Papers)*, pp. 19–23, 2019. doi: 10.2312/evs.20191164

[32] C. R. Palmer and C. Faloutsos. Density Biased Sampling: An Improved Method for Data Mining and Clustering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 82–92, 2000. doi: 10.1145/335191.335384

[33] S. Parthasarathy. Efficient Progressive Sampling for Association Rules. In *Proceedings of IEEE International Conference on Data Mining*, pp. 354–361. IEEE, 2002. doi: 10.1109/ICDM.2002.1183923

[34] N. Pezzotti, B. Lelieveldt, L. van Der Maaten, T. Höllt, E. Eisemann, and A. Vilanova. Approximated and User Steerable tSNE for Progressive Visual Analytics. *IEEE Transactions on Visualization and Computer Graphics*, 23(7):1739–1752, 2016. doi: 10.1109/TVCG.2016.2570755

[35] F. Provost, D. Jensen, and T. Oates. Efficient Progressive Sampling. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 23–32, 1999. doi: 10.1145/312129.312188

[36] M. Riondato and E. Upfal. Mining Frequent Itemsets through Progressive Sampling with Rademacher Averages. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1005–1014, 2015. doi: 10.1145/2783258.2783265

[37] A. Rosenfeld. *Multiresolution Image Processing and Analysis*, vol. 12. Springer Science & Business Media, 2013.

[38] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang. Incremental Learning for Robust Visual Tracking. *International journal of computer vision*, 77(1-3):125–141, 2008. doi: 10.1007/s11263-007-0075-7

[39] H. Samet. The Quadtree and Related Hierarchical Data Structures. *ACM Computing Surveys*, 16(2):187–260, 1984. doi: 10.1145/356924.356930

[40] H.-J. Schulz, M. Angelini, G. Santucci, and H. Schumann. An Enhanced Visualization Process Model for Incremental Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(7):1830–1842, 2015. doi: 10.1109/TVCG.2015.2462356

[41] C. D. Stolper, A. Perer, and D. Gotz. Progressive Visual Analytics: User-Driven Visual Exploration of In-Progress Analytics. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1653–1662, 2014. doi: 10.1109/TVCG.2014.2346574

[42] Y. Tanahashi, C.-H. Hsueh, and K.-L. Ma. An Efficient Framework for Generating Storyline Visualizations from Streaming Data. *IEEE Transactions on Visualization and Computer Graphics*, 21(6):730–742, 2015. doi: 10.1109/TVCG.2015.2392771

[43] G. Team. Gaia Data Release 2 - Summary of the Contents and Survey Properties. *Astronomy & Astrophysics*, 616:id.A1, Aug. 2018. doi: 10.1051/0004-6361/201833051

[44] C. Turkay, P. Filzmoser, and H. Hauser. Brushing Dimensions - A Dual Visual Analysis Model for High-Dimensional Data. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2591–2599, 2011. doi: 10.1109/TVCG.2011.178

[45] C. Turkay, N. Pezzotti, C. Binnig, H. Strobelt, B. Hammer, D. A. Keim, J.-D. Fekete, T. Palpanas, Y. Wang, and F. Rusu. Progressive Data Science: Potential and Challenges. *arXiv preprint arXiv:1812.08032*, 2018.

[46] S. Xiang, X. Ye, J. Xia, J. Wu, Y. Chen, and S. Liu. Interactive Correction of Mislabeled Training Data. In *Proceedings of the IEEE Conference on Visual Analytics Science and Technology*, pp. 57–68, 2019. doi: 10.1109/VAST47406.2019.8986943

[47] J. Yuan, S. Xiang, J. Xia, L. Yu, and S. Liu. Evaluation of Sampling Methods for Scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1720–1730, 2021. doi: 10.1109/TVCG.2020.3030432

[48] E. Zgraggen, A. Galakatos, A. Crotty, J.-D. Fekete, and T. Kraska. How Progressive Visualizations Affect Exploratory Analysis. *IEEE Transactions on Visualization and Computer Graphics*, 23(8):1977–1987, 2016. doi: 10.1109/TVCG.2016.2607714