Construct boundaries and place labels for multi-class scatterplots

Article *in* Journal of Visualization · September 2021

DOI: 10.1007/s12650-021-00791-x

Zeyu Li
Tianjin University
18 PUBLICATIONS **71** CITATIONS
SEE PROFILE

Wang Meng
Tianjin University
8 PUBLICATIONS **35** CITATIONS
SEE PROFILE

Jiawan Zhang
Tianjin University
162 PUBLICATIONS **2,500** CITATIONS
SEE PROFILE

Noname manuscript No.
(will be inserted by the editor)

Construct Boundaries and Place Labels for Multi-class Scatterplots

Zeyu Li · Teng Wang · Meng Wang · Jiawan Zhang

Received: date / Accepted: date

Abstract Drawing boundaries and appending text labels for each class of multi-class scatterplot are two common steps to help people perceive and understand class-level spatial and semantic infromation hidden in the scatterplot. However, massive data points, highly-overlapped classes, widespread outliers, extremely non-uniform density of data points lead to readability and scalability issues with existing methods. In this paper, we propose a set of methods that form a three-step framework to overcome these issues. We enable the boundary compact, readable, and controllable, and can find an ideal position that matches the human visual preference for each label. In the first step, we use a MST-based clustering algorithm to further divide classes into clusters and remove class-level outliers to avoid the distortion of boundaries. A stroke-based interaction is integrated into the algorithm, allowing the user to quickly correct the identified clusters or materialize the clusters in his or her mind. In the second step, we design a grid-based boundary construction pipeline which enables the user to tighten the boundary into the main distribution region of its corresponding class in a controlled manner by gradually filtering out cluster-level outliers. Gridding improves scalability at the scale of data points and helps users gain insights by generating different distributions of classes based on a relative or absolute density threshold. In the third step, by combining three factors: the boundary of the target cluster, the boundary of the label, and the density distribution of the target cluster, we can place

Zeyu Li, Teng Wang, Meng Wang
College of Intelligence and Computing, Tianjin University
E-mail: lzytianda@tju.edu.cn, wt0201@tju.edu.cn, meng.wang@tju.edu.cn

Corresponding author Jiawan Zhang
Tianjin cultural heritage conservation and inheritance engineering technology center and Key Research Center for Surface Monitoring and Analysis of Relics, State Administration of Cultural Heritage, College of Intelligence and Computing, Tianjin University
E-mail: jwzhang@tju.edu.cn

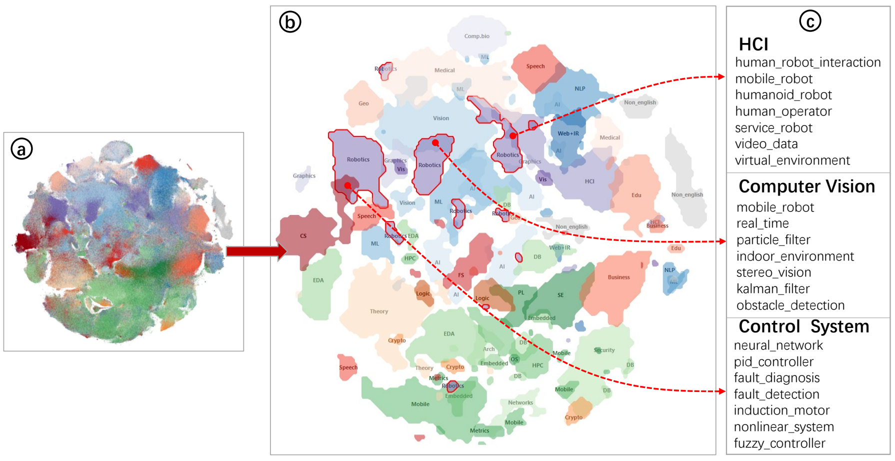

Fig. 1 An example of using our methods to transform a large-scale, unreadable multi-class scatterplot(a) into a map(b) with clear and rich class-level information. In the map, the distribution of each class can be clearly seen, which makes it possible to investigate the semantics(c) of overlaps between classes.

the label closer to its visually ideal position. Rich illustrations and two cases demonstrate the effectiveness of our methods.

Keywords multi-class scatterplot · visual abstraction · boundary · labeling

1 Introduction

Multi-class scatterplot is a common visualization to show patterns, trends, and outliers hidden in classes, as well as relationships between classes. However, for large-scale data and data whose classes overlap significantly, overdraw problem makes the scatterplot unreadable. In this paper, we focus on alleviating the overdraw and helping the user to identify and understand the important class-level information in multi-class scatterplots.

Data sampling, bin aggregation, scatterplot matrices, and boundary construction are four widely used visual abstraction approaches to alleviate overdraw. Among them, boundary construction has its own advantages. It provides a higher level of abstraction than data sampling approach because it hides points but explicitly highlights the scope of classes, which is consistent with our goal of highlighting class-level information. It scales better with the number of classes than bin aggregation approach, since the latter is limited by the small area of bins, which makes it difficult to present a data summary of many classes. Compared to scatterplot matrices, boundary construction needs less space.

Existing boundary construction methods have some overlooked and unsolved problems. Tessellation techniques (e.g., GMap [12]), hull techniques (e.g., Concave hull [36]), and set visualization techniques (e.g., Bubble sets [10]) simply encapsulate all points of a class with a single boundary. Consider a scenario where points belonging to the same class are located in several widely separated regions, the boundaries created by the three techniques cover almost the entire scatterplot, leading to a serious distortion. Second, outliers located between classes (class-level outliers) and around clusters (cluster-level outliers) make the boundaries very complex, resulting in significant visual clutter given the tight arrangement of clusters. Third, due to the lack of the concept of 'region', these three techniques do not allow the user to gain further knowledge by expanding or narrowing the scope of the class. Kernel Density Estimation (KDE)-based techniques (e.g., Scatterplots [42]) can naturally ignore class-level outliers by thresholding the density, but they cannot remove cluster-level outliers whose local density exceeds a set threshold. Moreover, in some scenarios, it is valuable to find relatively exclusive regions of a class or regions where a few classes all have a high proportion. However, KDE-based methods do not support such tasks, as they cannot form a boundary according to the local proportion of a class.

Similar to country names placed in corresponding borders on a map, concise text labels placed in corresponding classes supplement semantic information for scatterplots. Compared to legends placed in corners, labels placed directly on classes avoid frequent and long movements of our visual focus. In this paper, we hope to place labels in the visual center of the scattered points of clusters. Area-feature label placement algorithms [2,11,13,26,48] aim at finding a best placement location in an area object, such as a ploygon. However, they usually treat labels as particles and consider only the boundary of the area object, ignoring the other two important factors. One is the boundary of the label itself. An ideal position should make the outline, not just the center of the label far from the boundary of the area object. The other is the density distribution of the scattered points. An ideal position should be near regions with high density because they attract more visual attention.

In this paper, we combine clustering, boundary construction, and label placement into a three-step framework to highlight class-level information for large-scale multi-class scatterplots. As stated before, separate clusters of the same class may lead to distorted boundaries. Therefore, clustering is needed as a preliminary step to the boundary construction. In the clustering step, we select a MST-based clustering algorithm to identify clusters and remove class-level outliers. The algorithm can integrate a stroke-based interaction that allows the user to correct or rebuild identified clusters by quickly picking up arbitrary clusters from the scatterplot. In the first substep of the boundary construction, the plane is divided into grids and then all subsequent substeps are performed on grid-level. This improves the scalability of the boundary construction in term of the number of data points. Besides, gridding introduces the concept of 'region', making it possible for the user to create different scopes of a class by filtering the grids through a relative or absolute density threshold. In the following steps, diverse cluster-level outliers are detected and removed, such as points with sparse neighborhoods and islands that lie around the main distribution region of clusters. Removing outliers results in a slight loss of information, but it significantly improves the readability of boundaries by making them concise and tight to the main distribution region of clusters. The granularity of outlier removal can be controlled by the parameter of each substep. In the label placement step, we introduce a distance index and a density index to address the two ignored factors mentioned above. By adjusting a weight, the user can make a trade-off between moving the label away from the boundary of the target cluster and toward a high-density region.

We summarize our contributions as follows:

– We propose a three-step framework that highlights class-level information in large-scale multi-class scatterplots by first constructing boundaries and then placing text labels for classes.
– We design a stroke-based cluster refinement interaction that allows the user to quickly correct clusters identified by the algorithm, or materialize the clusters in his or her mind.
– We propose a grid-based and controllable boundary construction pipeline which alleviates the overdraw problem and allows the boundary to strike a balance between simplicity and accurate delineation of data distribution.

2 Related work

2.1 Visual Abstraction for Multi-class Scatterplots

In recent years, many methods have been proposed to improve the perception of multi-class scatterplots by providing a visual abstraction focusing on different aspects. Wang et al. [47] present an effective approach for color assignment based on a set of given colors that is designed to optimize the separability perception between multiple data groups. Lu et al. [30] extend [47] by not only assigning but also creating palettes and generalizing to other visualizations, such as line chart and bar chart. Winglets [31] enhance the classic scatterplot by improving the perception of association and uncertainty of points to their related cluster. Zhou et al. [50] describe an attribute-based abstraction method to present the associated attributes and simplify visual clutter for large-scale geographical points. Different from the goals of the above work, our paper focuses on giving the analyst a quick understanding of a multi-class scatterplot by explicitly presenting the distribution and semantics of each class.

Overdraw is a common concern for scatterplots [42]. The visual clutter it caused seriously hinder perception and understanding of data. Data sampling is one common visual abstraction approach to alleviate overdraw. For example, Chen et al. [9] present a non-uniform recursive sampling technique with a specific goal of faithfully presenting relative data and class densities, while preserving major outliers. Chen et al. [7] employ a hierarchical multi-class sampling technique to show a feature-preserving simplification. Hu et al. [18] provide a formulation of multi-class scatterplot sampling in terms of the Exact Cover Problem, which can be extended for outlier inclusion. Zhao et al. [52] design a new graph sampling approach to efficiently preserve minority structures that are rare and small in a graph but are very important in graph analysis. Yuan et al. [49] investigates the capability of seven typical sampling strategies in maintaining multiple data features, including region density, class density, outliers, and overall shape in the sampling results. Bin aggregation uses a summary strategy to reveal the distribution of data. Heimerl et al. [16] explore the space of visual designs for creating effective visual representations of these summaries. However, displaying summaries in a confined bin leads to poor scalability in the number of classes. Chen et al. [8] demonstrate that animation using flickering points is an effective way to help users identify dense regions. Obviously, this mathod is not suitable for static pages. Another common approach to alleviate the overdraw problem is to draw contours or boundaries for classes. For example, Li et al. [27] and Zhao et al. [51] apply contours to highlight clusters. We believe that explicit boundaries are more straightforward than sampled scatterplots in term of presenting the distribution of data, and that contours with different scopes can reveal more information, such as density gradients. We will discuss boundary construction methods in detail in the next subsection.

2.2 Boundary construction

Under the premise of constant distribution of scattered points, we divide the techniques for constructing the boundary of scattered points into four categories: tessellation techniques, hull techniques, set visualization techniques, and Kernel Density Estimation-based (KDE) techniques.

In geometric tessellation techniques, each point is contained in a geometric cell. Cells are then grouped if they represent similar points. For instance, GMap [12] adapts the Voronoi mesh to visualize clustered data by coloring the cells of all nodes of the same cluster in the same color. Using the same border construction technique, Cartograph [43] creates a thematic map for Wikipedia. Geometric hull techniques generate areas around groups of points of data in the view. For example, Stahnke et al. [45] and Bernard et al. [4] use hull to represent a group of data items, generated either interactively by the user or automatically by a clustering or classification algorithm. Liu et al. [29] use convex hull to encapsulate all points associated with an attribute vector in a two-dimensional projection of high-dimensional data. Set visualization techniques emphasize mebership of set elements while trying to avoid the intersection of outlines of different sets. To this end, Bubble sets [10] computes an energy map over the pixels in the convex hull containing the set elements and then applies the marching squares algorithm to compute the implicit surface from the map. Byelas et al. [6] first draw smooth boundaries borders for an area of interest by constructing a skeleton with texture splatting, and then performs a post-processing step to avoid overlaps by erasing elements that mistakenly fall within the target area of interest.

The above three types of techniques form precise boundaries that warp all given scattered points, regardless of potential outliers that can significantly reduce the readability of the multi-class scatterplot. Therefore, an upstream step to handle outliers is usually required. For example, Mashima et al. [33] remove most of the fragments shown in the map created by GMap by modifying the similarity between data points. Kobourov et al. [20] avoid discontinuous regions for GMap by preserving the given embedding and changing the cluster assignment of data points or just the opposite. Sen et al. [43] simply delete outliers using a signal-processing technique. However, in some scenarios, it is not allowed to change or delete the original data. Furthermore, these three types of techniques are all point-based, which introduces two additional problems. First, they cannot create different scopes for the class based on its data density. Second, they have scalability issues when faced with large datasets. Our proposed grid-based boundary construction pipeline introduces the concept of 'region' by gridding the 2D space, thus avoiding these two problems.

KDE-based techniques first build a density scalar field by a kernel function and then generate contours given a set of density thresholds. Splatterplots [34] is an example of KDE-based technique. It automatically groups dense data points into contours, while allowing continuous zoom to reveal abstracted details. Outliers are defined and handled differently between Splatterplots and our method. In Splatterplots, outliers are some points sampled from low-density regions and they are explicitly shown to be perceived by the analyst. Outliers are classified into two levels in our method: class level and cluster level. They violate the simplicity of the boundary, so they are identified and removed. Moreover, since Splatterplots can only determine contours(boundaries) by the absolute data density of individual classes, it does not support the task of finding the unique region of a class, namely the region with high proportion of the target class. While our method can accomplish this task, because it supports grid filtering by relative density among classes.

2.3 Label Placement

A large number of studies on label placement have focused on point-feature label placement, with different requirements and strategies. For example, Mote [37] focuses on efficiency, achieving tens of thousands of nodes labeled at a rate of several frames per second, using a novel geometric de-confliction approach along with a unique label candidate cost analysis. Been et al. [3] emphasize avoiding the distracting or jarring behavior of labels during continuous zooming and panning by making placement a continuous function of scale and letting label selection take the form of active ranges. Meng et al. [35] draw attention to clutter-aware and formulate a clutter model that uses four factors: confusion, visual connection, distance, and intersection. Kouřil et al. [22] point out that real-time selection of label level and representative instances is the key to interactive labeling in densely populated multi-scale and multi-instance environments. Mumtaz et al. [38] highlight outliers within scatterplots by labeling them only. In contrast to these efforts, which aim to place labels as much as possible while avoiding their overlaps, our goal is simply to choose a unique location for a group of scattered points. In this sense, our goal is closer to area-feature label placement [2].

In general, area-feature label placement is about finding a point within an area(polygon) that takes up as much space around itself as possible. Daniel et al. [13] design an iterative algorithm to find the largest incircle of the polygon, but it might fall into local optimality. With the same goal, Agafonkin [1] introduces an exhaustive search strategy that trades efficiency for precision while ensuring reliable precision growth during the search process. Elhami et al. [11] propose a geometry-based method that works by shrinking the polygon until the convex hull of the shrunken polygon fits completely inside the original polygon, and the position of the label is then taken as the centroid of the shrunken polygon. Instead of finding a single optimal point, Krumpe et al. [23] distort the label to span and fit the shape of the polygon. However, labels in vertical narrow regions are hard to read. Wu et al. [48] develop a grid algorithm where a maximal inclusive rectangle is searched for the numerical label of its corresponding polygon, the midpoint of which is considered the potential position. Pokonieczny et al. [40] and Li et al. [26] use neural networks to learn the complex and implicit features characteristics of label placement from manual operation or existing maps, and they show promising results. However, the above methods do not consider the boundary of the label or the density distribution of scattered points, both of which we believe are important to our goal.

3 Clustering and Cluster Refinement

For many multi-class scatterplots, it is not appropriate to build a boundary directly for each class. Consider the following two scenarios: 1. Massive points are scattered among conspicuous clusters like background noise (see Fig.2 and Fig.3 for examples). If they are treated as part of classes, boundaries may be extremely complex and can not tightly enclose the primary data distribution area of the corresponding class. 2. Data points of the same class fall into several widely separated regions. This often occurs in scatterplots generated by nonlinear dimensionality reduction algorithms, which is known as 'false tears' distortion [21, 24, 25]. In this case, the boundary of the class may cover almost the entire scatterplot. Therefore, it is necessary to remove the noisy data and further divide classes into clusters before the next boundary construction step. We use a clustering algorithm to perform such upstream step.

We emphasize that cluster refinement is frequently required. Consider the following three scenarios: 1. global parameters cannot adapt to classes with different data distribution, so several automatically identified clusters need to be corrected; 2. several identified adjacent clusters have consistent semantics, so they should be merged. 3. in contrast to 2, a identified cluster has multiple distinguishable centers with different semantics, so it should be further split.

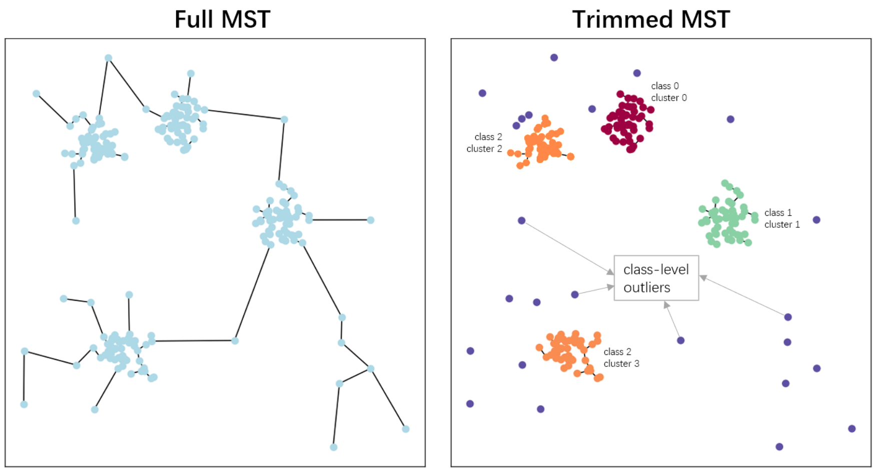

Fig. 2 After removing all edges whose length is larger than ϵ, subtrees are identified as clusters.

Recalling the general features of usual multi-class scatterplots and the common scenarios above, we summarized five requirements for an ideal clustering algorithm:

- **R1** Identify clusters that are visually significant.
- **R2** Remove background noise (class-level outliers).
- **R3** No need to specify the number of clusters in advance. Because it is difficult for the user to quickly determine the ideal number of potential clusters of a given class in a cluttered scatterplot with numerous classes.
- **R4** Adapt to clusters with arbitrary shape. Because clusters with irregular boundaries are common.
- **R5** Since clusters obtained by an automatic algorithm may not be ideal for some classes, the algorithm should have endogenous support for or has the ability to incorporate cluster refinement.

3.1 MST clustering algorithm

We considered a number of common clustering algorithms, and finally chose a Minimum Spanning Tree (MST)-based clustering algorithm [46]. R2, R3, and R4 exclude most classical algorithms, including Kmeans (R2, R3, R4), Spectral clustering (R2, R3), and Hierarchical clustering (R2). As shown in Fig.3, Meanshif often fails to provide satisfactory noise identification even with careful parameter tuning. Although DBSCAN supports the first four requirements well, its parameters are not intuitive, often leading to time-consuming trial-and-error. In contrast, the chosen MST-based clustering algorithm satisfies all five requirements, in particular it can support R5 by simply integrating a stroke-based interaction.

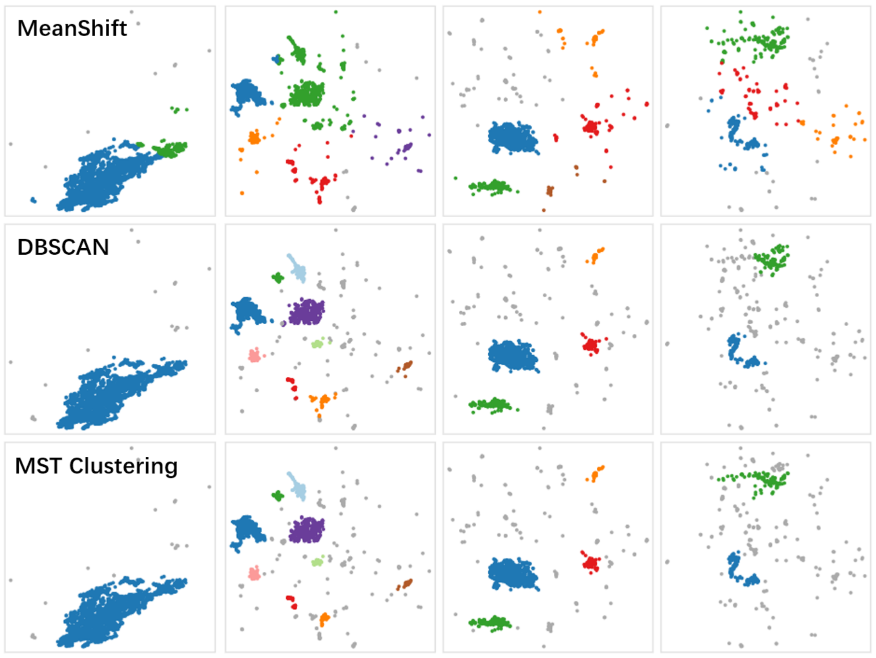

Fig. 3 Comparison between three common clustering algorithms. MeanShift often groups noisy data into normal clusters.

The MST clustering algorithm first builds the Euclidean Minimum Spanning Tree (EMST) of the points to be clustered (Fig.2 (a)), and then trims the edges whose length is larger than a threshold ϵ (Fig.2 (b)). Then, each unconnected subtree can be considered as a cluster (**R1**). When ϵ is increased, adjacent clusters tend to be merged; conversely, more small clusters are created. Users can filter out small clusters by setting a threshold representing the minimum cluster size (**R2**). We call these filtered clusters class-level outliers. Accordingly, we refer to the remaining local outliers that lie around a cluster's main distribution region as cluster-level outliers. These outliers cannot be filtered out by further decreasing the ϵ because small ϵ will tear ideal clusters to shreds. We leave the detection and handling of cluster-level outliers to the following boundary construction step. Obviously, the MST clustering algorithm does not need to specify the number of clusters in advance (**R3**), and it is suitable for clusters of arbitrary shape (**R4**). Moreover, it has the following two advantages:

– The two parameters (ϵ and threshold of minimum cluster size) have a clear and intuitive physical meaning. Essentially, the goal of the MST-based clustering algorithm is to find a partition of data points in which the single-link distance between each pair of clusters is greater than ϵ. This means that the physical interpretation of ϵ can be more intuitive. That is, if the distance between two groups of points is larger than ϵ, they are recognized as two clusters, which is consistent with the way our visual system perceives clusters. Therefore, even users without computer science knowledge can quickly master the adjustment of ϵ.

– Only the ‘build the EMST’ step requires many computations. The time complexity of this step can be $O(n \log n)$ using the Kd-tree [32]. It takes about 2.1s CPU time to build the EMST of 100,000 data points on an regular PC (Intel i5-7200U processor, 8 GB RAM) with a C++ implementation[1]. Note that this step is preformed on the individual class, not the entire dateset, and it is a one-time step which can be performed offline in advance. Its following step - determining the clusters by cutting edges - can be done in near real-time even for millions of points. Hence, the user can obtain new clusters immediately after ϵ changes.

In the next sub-section, we show how the MST clustering algorithm supports cluster refinement (**R5**) with a stroke-based interaction.

3.2 Stroke-based Cluster Refinement

Although most clustering algorithms can change the results by adjusting their parameters, approximating the desired results can still be difficult or even uncontrollable. In fact, the user can detect potential clusters by visual perception (e.g., noticing gaps or hotspots) and/or interactive semantic inspection (e.g., using lens-based interaction techniques to check local semantics [17, 27]). The goal at this point is to quickly materialize the clusters in the user’s mind, i.e., to pick them up directly in the scatterplot. Lasso interaction is the most common solution, but it is not efficient in our scenario. Imagine that two adjacent clusters have narrow gap and complex boundaries, it is not easy to lasso one of them accurately, especially given the fact that mouse is not suitable for precise drawing. Moreover, in the case of merging two adjacent clusters, lasso interaction requires profiling the complete outline of the merged cluster, which is time-consuming and tedious. In our paper, we extend the chosen MST-based clustering algorithm by integrating a stroke-based interaction for refining clusters. The interaction allows users to customize clusters by simply drawing two types of strokes - join-curve and cut-curve - on scatterplots.

As shown in Fig.4, to aggregate the blue points into a single cluster, the user only needs to draw a curve roughly across their coverage. We first take interval samples (red points) of the curve and then find the closest point (green points) of each sample among the target blue points to serve as a seed. Given the trimmed MST, the subtree in which each seed resides is known. Then, we merge the data points of corresponding subtrees of all seeds with union operator. In this way, we can get a customized cluster with a single stroke. We call this type of stroke join-curve, since adjacent regions (subtrees) crossed by a join-curve will be joined.

[1] https://github.com/AndrewB330/EuclideanMST

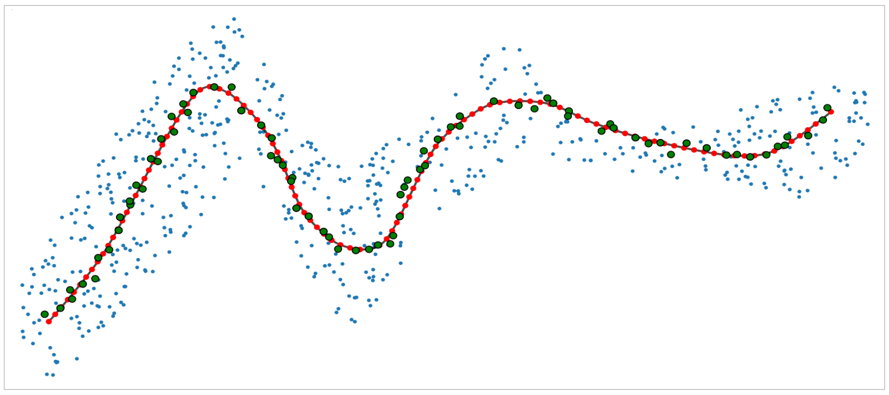

Fig. 4 Principle of the join-curve. A join-curve is drawn to aggregate the blue points into a single cluster. The red points represtent samples and the green points are the closest points to the samples and serve as seeds.

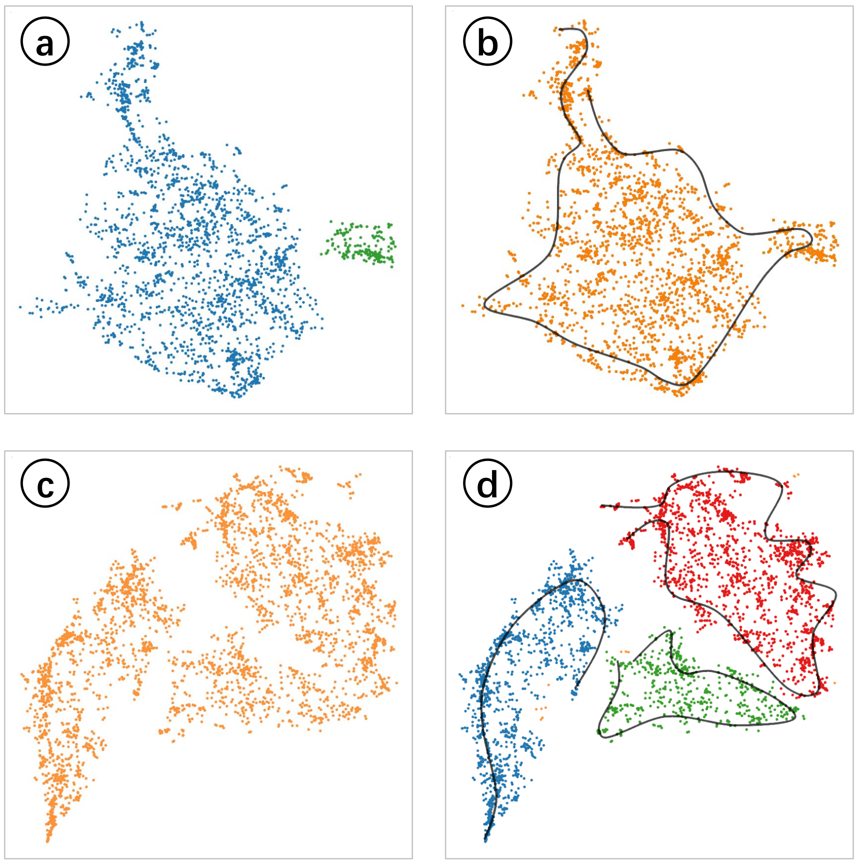

Fig. 5 Two examples of interactive cluster pickup with join-curve. ⓐ → ⓑ: merge two clusters; ⓒ → ⓓ: split one cluster into three.

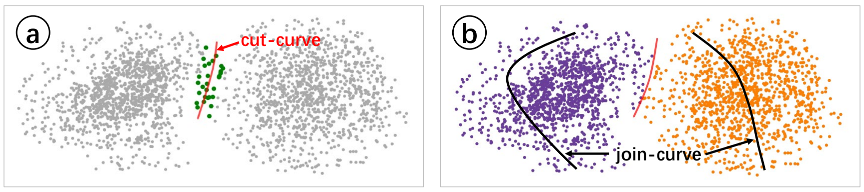

Fig. 6 Example of applying our cut-curve. ⓐ shows two potential clusters with distinct centers and a vague dividing line. A red cut-curve is drawn to separate them. The green points are those near the cut-curve that are used to create an artificial gap. ⓑ shows two picked clusters by then using two join-curves. The green points in ⓐ are assigned to one of the picked clusters using a KNN-like neighborhood voting method.

We demonstrate interactive cluster customization using two examples in which we merge or split clusters identified by MST clustering. Note that our method is not limited to rearranging the identified clusters, but can materialize **arbitrary** clusters in the user's mind.

Merge To merge the two clusters shown in Fig.5 (a), the user only needs to sketch their coverage with a single join-curve, as shown in Fig.5 (b). If the user increases the value of ϵ, a very short join-curve that just connects the target clusters is sufficient. This is much easier and more convenient than lasso interaction, as it it doesn't require precise boundary delineation.

Split To split the cluster shown in Fig.5 (c) into three clusters, the user first outlines one of the desired clusters with a join-curve. The algorithm immediately feeds back a identified cluster. If the identified cluster is larger than expected, the user can reduce ϵ, otherwise increase ϵ. In either case, it is helpful to redraw the stroke to more densely and accurately traverse the distribution region of the desired cluster. Each time the user adjusts ϵ or redraws the stroke, the identified cluster of the current stroke is automatically updated. Points that have been clustered previously will not be picked up by subsequent strokes. In this way, the user can draw one stroke at a time, and each stroke will pick up a desired cluster.

Join-curve has a drawback. It is severely affected by the single-linkage effect, i.e., it cannot distinguish two clusters if some points connect them like a bridge, even though the two clusters are visually distinct or far apart. To solve this problem, we introduce another kind of stroke - cut-curve (Fig.6). The user can draw a cut-curve along the intersection of two potential clusters. We obtain the points near the curve by an idea similar to Fig.4: sampling on the curve and then finding the points within a certain distance from the samples. These points are then removed from their corresponding subtrees. This is equivalent to creating an artificial gap that cuts off the original connection(bridge). In this way, our cut-curve can overcome the single-linkage effect and the potential clusters can then be picked up by join-curve (Fig.6). As for the assignment of the removed points, we provide two strategies: 1. ignore these points, that is, do not assign them to any cluster; 2. use a KNN-like neighborhood voting method to assign them to one of the identified clusters one by one. Cut-curve simply acts as an assistant to join-curve and is specialized to deal with the single-linkage effect. On its own, it cannot perform the cluster refinment.

All cluster refinement interactions are near real-time. Merging subtrees and deleting nodes from a subtree are both set operations. The efficiency of finding the nearest points and the points with a certain distance to samples can be greatly improved by data structure such as Quadtree or Kd-tree. In tests, for a class with 100,000 points, all interactions can be responded in less than 1 second on a regular PC.

4 Boundary Construction

We construct boundaries for each cluster for two purposes: as a necessary input to the next step - label placement and to help the reader perceive the spatial distribution of clusters. As mentioned in Related Work, most existing boundary construction methods for scattered points are point-based, whose goal is to wrap all points with one or more boundaries. However, in some scenarios, the reader focuses on perceiving the main distribution of the class and does not care about insignificant outliers. Moreover, outliers not only increase the complexity of the boundary, leading to visual clutter, but also cause the boundary to include many regions that do not belong to the target cluster, leading to a misinterpretation of data. The existing methods do not handle the outliers themselves, but rely on external processings. But these processings often can not detect multiple types of outliers and they handle outliers only in one way - filtering, which is not suitable for some cases. Besides, due to the lack of the concept of 'region', the existing methods do not allow the user to check the exclusive or core regions of classes with different scopes. We develop a grid-based and controlled boundary construction pipeline. It solves the above problems by partitioning space and data into grids and incorporating multiple outlier detection and processing strategies. Thus, the boundary can strike a balance between maintaining simplicity and delineating the data distribution accurate.

4.1 Pipeline of Boundary Construction

As shown in Fig.7, we take six steps for each target cluster to get a boundary that wraps its main distribution region.

Step1 Gridding. We fill the grids in which the number or the proportion of data points of target cluster exceeds a certain threshold (the first parameter: *point_num*, default: 10; or *proportion*, default: 0.1). Gridding means that the minimal processing unit in subsequent steps is no longer a single point but an aggregation of points, which greatly improves scalability. The time required

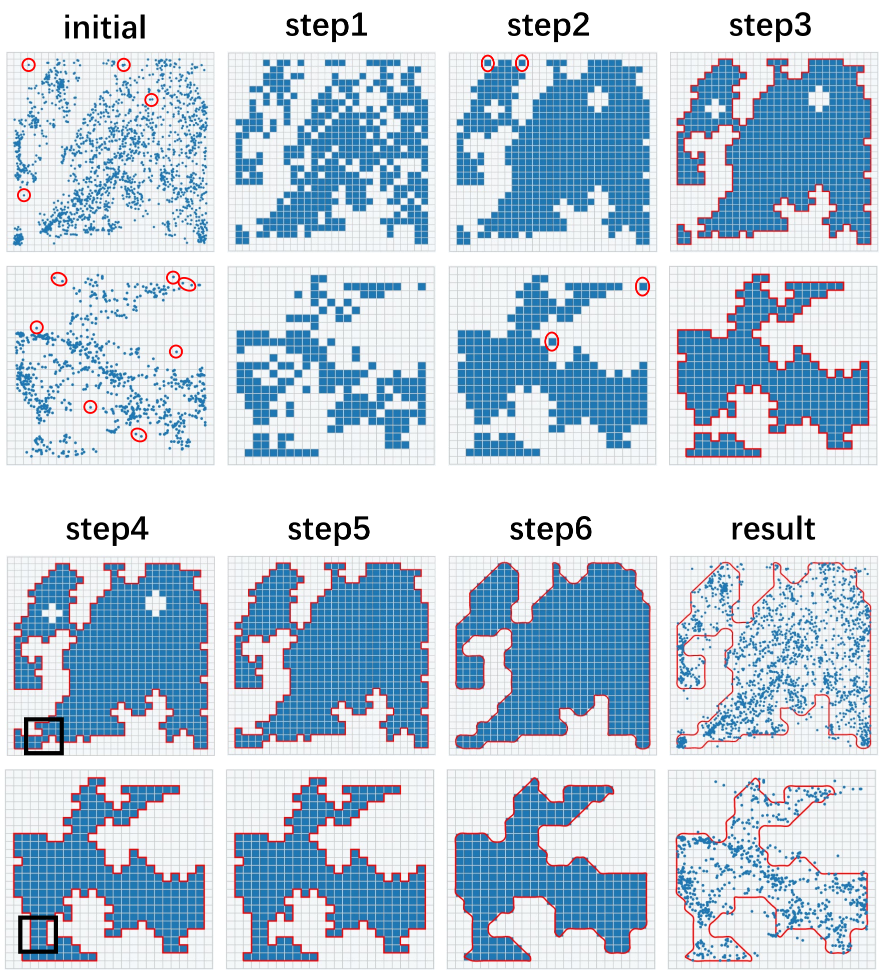

Fig. 7 Six steps of boundary construction pipeline. The red circles highlight the outliers that are removed in the next step. The black boxes highlight the merges of adjacent continents.

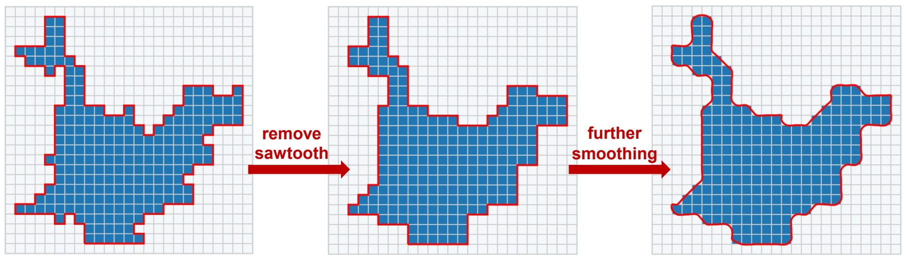

Fig. 8 Two steps of boundary smoothing.

for boundary construction depends almost only on the number of grids and does not explode with the increase of data points.

Step2 Eliminate the discrete blank grids that are interspersed with the filled grids. The discretization introduced by gridding prevents the formation of a continuous region that provides an overview of data distribution. If we proceed directly to the next step - continent identification, a large number of scattered continents will be formed, resulting in overly complex and meaningless boundaries. We introduce morphology methods in image processing to solve this problem. Specifically, we perform a close operation to fill the inside scattered blank grids, followed by an open operation (optional) to remove the scattered filled grids around the main distribution region of the target cluster. The structuring element we used in the open/close operation is a shape of cross of length 3. The number of iterations is set by the user. (the second parameter: *iterations*, default: 1). To keep the boundary native, it should be as small as possible.

Step3 Identify continents, filter continents, and determine the boundary of continents. First, an undirected graph is created according to the distribution of filled grids. Each filled grid is considered as a node and an edge is formed between two nodes if their corresponding grids have a common edge. Then, the identification of continents is equivalent to the detection of connected components of the undirected graph. After identification, continents containing only a small number of grids can be ignored (the third parameter: *grid_num*, default: 3). Finally, we apply the algorithm Moore-Neighbor Tracing [14] to determine the boundary of each continent.

Step4 Merge adjacent continents. Merging adjacent boundaries also reduces visual complexity. First, we consider pairs of continents reachable in n-jumps as adjacent continents (fourth parameter: *n_jump*, default: 3). Then, we merge adjacent continents by filling the grids on n-jump paths. Finally, we rebuild the boundaries of the merged continents. The black boxes in Fig.7 mark where the merge occurs.

Step5 Fill small holes. If the hole is small, it should be filled, since small holes are usually insignificant but increase visual complexity. Otherwise, it should be left to show the true distribution of data. The user can determine whether the hole should be filled by setting a threshold that represents minimum acceptable size (the number of grids the hole occupies, fifth parameter: *hole_size*, default: 3).

Step6 Smooth the boundary. As shown in Fig.8, this step consists of two substeps: 1. Sawtooth removal (optional): we detect bulges and depressions formed by a single grid, and then remove or fill them; 2. Further smoothing: we take the midpoint of each segment of the current boundary to form the final boundary, and smooth it further using Catmullrom Curve.

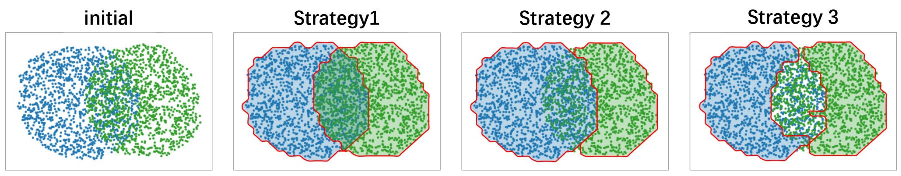

Fig. 9 Three strategies for adapting boundaries for overlaps.

The time required for boundary construction depends mainly on the number of grids and the number of clusters identified in the clustering step. We test the performance on a regular PC (Intel i5-7200U processor, 8 GB RAM) using the dataset shown in Case2, which consists of 4.1 million data points, 314 cluster, and 200*200 grids. The total time required for all steps is about 6.6 seconds with our python implementation.

4.2 Boundary Control

Our boundary construction pipeline can detect several types of outliers and process them in different ways. The first step filters out points whose neighborhood is absolutely sparse (using *point_num*) or relatively sparse(using *proportion*). The second step removes the spikes(thin lines) and the islands that occupy only a small amount of filled grids. If we consider the blank grids interspersed with filled grids as outliers, this step also removes these outliers by turning them into filled grids. The third step filters out the larger islands left over from the second step. The fourth step incorporates the relatively large islands that are not far from main continents into the main continents, while treating the remaining islands as individual continents. The effect of each step can be percepived by comparing the adjacent insets in Fig.7. In general, when the value of parameter (except the fourth parameter: *n_jump*) is increased, more outliers are removed and boundaries are more concise and tend to tighten towards the core region of data points.

It may be difficult for the user to find which step removes too much or too little outliers by only checking the final constructed boundaries. Therefore, introducing visualization into the parameter tuning process is a good idea. As shown in Fig.7, the output of each step should be presented on several example clusters, allowing the user to tune each parameter in the order of the steps. We believe that all other parameters are simple and straightforward, except for the parameter *iterations* in Step 2. The principle of tuning this parameter is that if there are still a large number of interspersed blank grids can be observed, the value of this parameter or the granularity of grids should be increased.

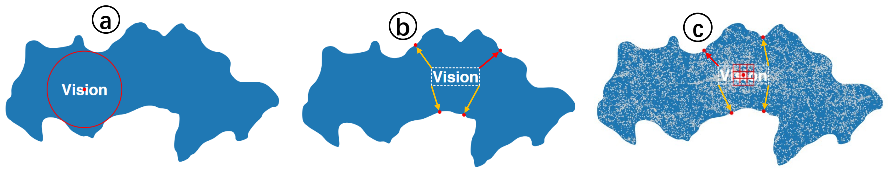

Fig. 10 Position of the label given ⓐ the boundary of the cluster; ⓑ the boundary of both the cluster and the label; ⓒ both the two boundaries and the density distribution of the data points. The circle in ⓐ is the largest inscribed circle of the boundary. The four arrows in ⓑ point from the corner of the bounding box of the label to the nearest point on the cluster boundary. The red grids in ⓒ indicate the region to be considered when calculating the density index for a candidate position.

4.3 Three strategies for overlaps

Dividing the space of the scatterplot into grids is essentially grouping the data into regions. This makes it possible to create different scopes of classes based on their local relative and absolute density, which can be valuable in some scenarios. For example, in a document space represented as a multi-class scatterplot, where each class represents a topic, the reader can understand intersections between topics by inspecting overlapped regions, or understand the characteristics of a topic by inspecting its core regions. Considering that boundary intersection can significantly increase visual complexity, we propose three strategies for adapting boundaries for overlaps (Fig.9):

– **Strategy1: overlaps are enclosed by two boundaries** In this strategy, the boundary of each cluster is constructed independently and the overlap between clusters is explicitly expressed by the intersection between boundaries. This strategy highlights the overlap but may lead to visual clutter when a mass of intersections prestent.
– **Strategy2: overlaps are enclosed by one of the two boundaries** This strategy reduces the visual clutter by eliminating the intersecion between boundaries, but the reader almost loses the perception of overlaps. It is suitable for situations where overlaps are not significant or there is no requirement for the perception of overlaps.
– **Strategy3: overlaps are not enclosed by any boundary** This strategy can be achieved simply by increasing the value of the parameter *proportion* in the first step. It is a compromise between the two strategies above, avoiding the visual clutter caused by boundary intersections while maintaining the perception of overlaps.

5 Label Placement

Placing a concise text label in the visual center of each cluster can help readers quickly grasp the semantics of the scatterplot. Area-feature label placement algorithms [2, 11, 13, 26, 48] are helpful for this task. Usually, they pick the position that farthest from the boundary of the target cluster (area object) as the ideal position, for example, the center of the largest inscribed circle of the boundary (Fig.10 (a)). But they ignore two other important factors: The shape and size, namely, the boundary of the label itself and the density distribution of the scattered points of the target cluster. Knowing the boundary and subordinate grids of the target cluster makes it easier to find a location near the visual center. We simply choose the center of the grid with the best integrated performance in all three factors as the final placement location.

We believe that the position that makes the label boundary farther away from the cluster boundary is closer to the visual center of the cluster. Therefore, we establish a distance index by combining the two boundaries. For a candidate grid, we align its center with the label center and then compute the distance between the four corners of the label bounding box and the boundary of the target cluster. We take the minimum value of the four distances as the score of the candidate grid under this index. The shift in label position between Fig.10 (a) and Fig.10 (b) proves that the boundary of the label matters.

Since denser areas are more likely to attract visual attention, we expect labels to be placed at a position with high point density. Therefore, we create a density index for each candidate grid by computing the average density of the local area in which it is located. The shift in label position between Fig.10 (b) and Fig.10 (c) proves that the density distribution of data points is another factor worth considering. The density index becomes important when there is a region whose density is extremely high compared to others, especially when the region is far from the optimal location determined by the distance index alone.

Candidate grids are usually all child grids of the continent corresponding to the target cluster. For each candidate grid, scores for both indexs are calculated. For both indexs, grids with higher scores are preferred. After converting the two scores into Z-scores, we simply add them with a weight to get the final score. The default weight of the distance index is 0.8 and can be adjusted by the user. The center of the grid with the highest final score gives the placement position.

6 Evaluatation

6.1 Qualitative Evaluation

First, we compare our boundary construction pipeline with other three representative methods by visually checking the quality of constructed boundaries on a sample multi-class scatterplot. The three representative methods are choosen from three types of boundary construction techniques for scatter points that are mentioned in Related Work, they are: Concave hull [4,29,45] (a representative of geometric hull techniques), Bubble sets [10] (a representative of set visualization techniques), and GMap [12] (a representative of geomet-

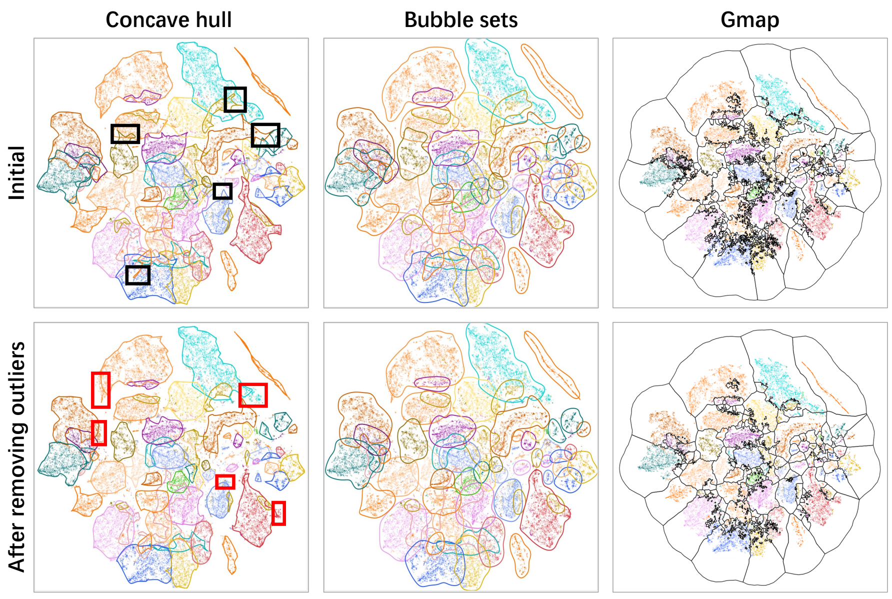

Fig. 11 Boundaries produced by the three representative methods with and without preforming outlier removal. The black boxes highlight several removed spikes by filtering out some outliers with COPOD [28]. The red boxes indicate a side effect of the outlier removal: some points are incorrectly identified as outliers and excluded from the boundaries.

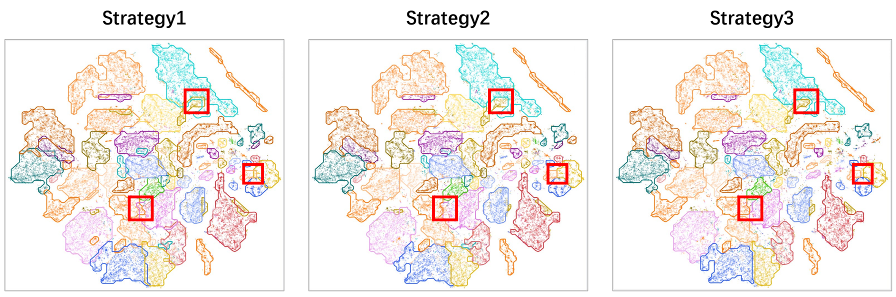

Fig. 12 Boundaries generated by our method with the three strategies mentioned in section 4.3. The red boxes present three groups of instances shown in Fig.9.

ric tessellation techniques). The sample multi-class scatterplot has 46278 data points and 30 classes (98 clusters).

The first row of Fig.11 shows the boundaries produced by the three representative methods. These boundaries are complex, leading to serious visual clutter. The boundaries produced by Concave hull and Bubble Sets have many spikes and intersections, making some regions almost unreadable. GMap creates massive fragments. Moreover, Bubble Sets and GMap cannot tightly enclose the data points. To verfiy whether the poor quality of the boundaries is simply caused by significant outliers, we introduce COPOD [28], a parameter-

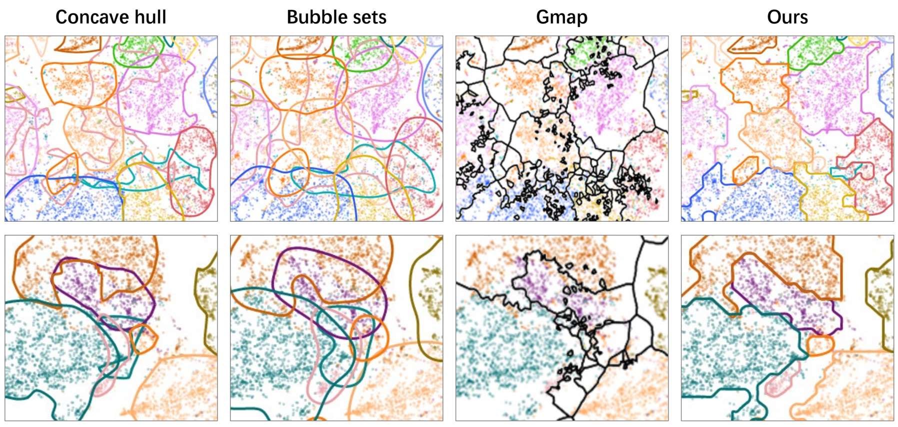

Fig. 13 Two local regions show obvious differences in the boundaries obtained by different methods.

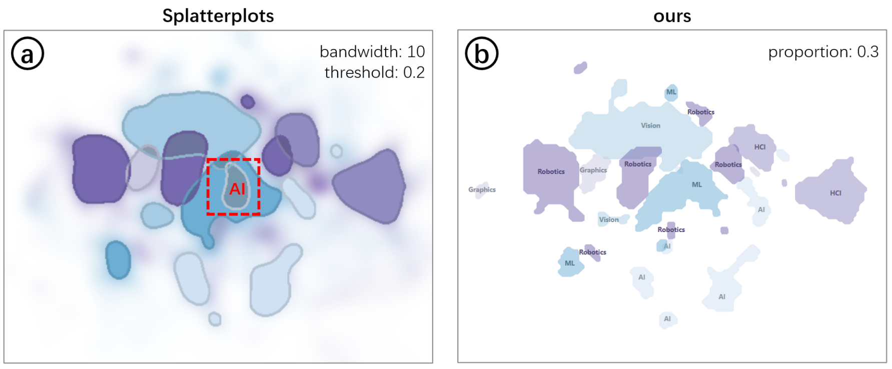

Fig. 14 Comapre splatterplots with our method. The AI area is retained in ⓐ because its absolute density is higher than the current density threshold. It is removed in ⓑ because its relative density is less than the current proportion threshold.

free probabilistic outlier detection algorithm, to preform outlier removal. We have verified that COPOD provides the best detection results for the current dataset compared to several classical algorithms, including HBOS [15], LODA [39], and VAE [19]. The second row of Fig.11 shows the bodundaries obtained after performing outlier removal with COPOD. It turns out that the outlier removal only slightly alleviates the visual clutter (some of the spikes are removed, highlighted by the black boxes) and it introduces a side effect: some normal points are incorrectly identified as outliers and excluded from their corresponding clusters (highlighted by the red boxes). Fig.12 shows the boundaries generated by our method with three strategies mentioned in section 4.3. In contrast, our method achieves concise and readable boundaries. Fig.13 presents two local areas that show a strong contrast.

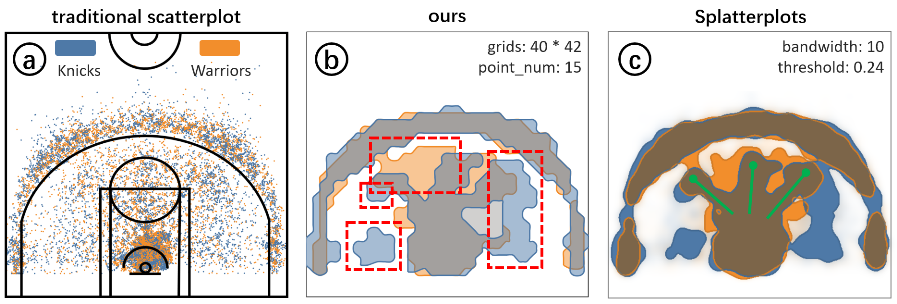

Fig. 15 ⓐ shows a total of 14,575 shots by the Warriors and the Knicks. The red boxes in ⓑ highlight the difference in preferred shooting locations between the two teams. The green lines in ⓒ indicate three popular shooting locations in basketball games.

Next, we show a difference between Splatterplots [34] and our method by a concrete instance. We select several research areas from Case2 in which the distribution of research areas (classes) are characterized by their papers (data points). As shown in Fig.14, the two methods give similar distributions of these areas, but the red box in (a) highlights a significant difference. The additional boundary in Splatterplots is the AI area. In Splatterplots, contours can only be generated according to the absolute data density of individual classes. The AI area has a density above the set threshold in the highlighted region, so it is retained. In addition to the absolute density (which is supported by using the *point_num* in Step1 of our boundary construction pipeline), our method also supports grid filtering based on a relative density by using the *proportion* in Step1. Although the AI area has a lot of data in the highlighted region, its proportion is less than 30%, so it is removed in (b). If the user's goal is to find regions that are relatively exclusive to each class, relative density is the appropriate choice. However, it is not supported by Splatterplots.

6.2 Case1: Compare Two Basketball Teams

In this case, we demonstrate that our boundary construction method is able to relieve overdraw probelm and extract valuable information from a jumble of highly overlapped scattered points. In the 2018-2019 NBA regular season, the Warriors had the best FG% and points/100 possessions, and the Knicks had the worst in both categories. We hope to provide some reasonable explanations by comparing their preferred shooting locations. In Fig.15, each dot represents a shot, and there are a total of 14,575 shots by the two teams. We notice that many shots are taken inside the restricted area and around the three-point line, but beyond that, it is difficult to obtain any other valid information due to the overdraw probelm. Then, we apply our proposed boundary construction method and SplatterPlots on this data. The two methods yield similar results, as shown in (b) and (c) (the red boxes in (b) highlight the differences). We find that the Warriors' shooting distribution is fairly even, suggesting that the

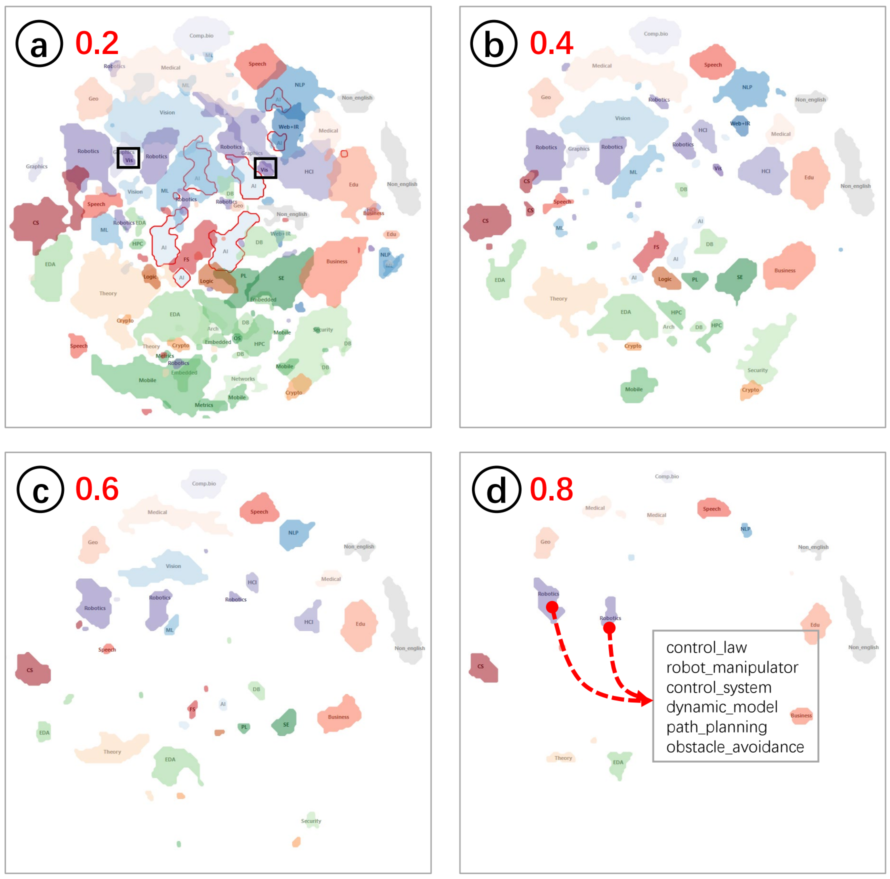

Fig. 16 As the parameter *proportion* is increased, the computer science map moves from having extensive overlaps between areas to preserving only the core of a subset of areas. The black boxes highlight the VIS area. It intersects with Computer Graphics and HCI.

team has no obvious weaknesses and is difficult to defend. The Knicks shoot farther away from the basket, which indicates that their penetration ability is inadequate. The Knicks also lack mid-range shots from either side of the end line. The team could consider bringing in an end line shooter to fill the vacancy. The overlaps between the two teams bear out a common sense in basketball, that the center line and the 45-degree diagonal on both sides are popular shooting locations (the green lines in (c)).

6.3 Case2: Make a Turbid Science Map Readable

Science map is a visualization method that uses map metaphor to reveal the structure and evolution of science [5]. Multi-class scatterplot is a common representation of science map in which classes represent local semantic structure, and similar documents (points) are distributed in close proximity. In this case, we demonstrate the effectiveness of our methods by making a large-scale turbid science map of computer science readable and comprehensible.

First, we build an initial science map of computer science. Using DBLP[2] records as an index, we collect textual data of 4.1 million computer science papers from Microsoft Academic Graph [44] and then generate a science map following the technology roadmap presented in Galex [27], i.e., vectorization $\rightarrow$ DR $\rightarrow$ coloring. We extend the classification scheme of CSRankings[3] and divide all papers into five categories (encode with color scheme) and 34 fine-grained areas (encode with color). For example, the AI category has Computer Vision, Machine Learning & Data Mining, Natural Language Processing, etc. Obviously, as shown in Fig.1 (a), due to the overdraw, we cannot perceive the distribution of each area, especially in the current case where related areas are in adjacent regions and have similar colors. Meanwhile, it is almost impossible to manually label each area.

We then convert the scatterplot into a map decorated with boundaries and text labels by applying the three steps presented in this paper. Step 1: Clustering. We set the ϵ to 0.05 and the minimum cluster size threshold to 10,000. In this way, we divide the 34 areas into 362 clusters and filter out 238,348 class-level outliers. Then we perform our stroke-based cluster refinement for each class, based on the locations and semantics of their clusters. In the end, we obtain 314 semantically distinct and independent clusters. Step 2: Boundary Construction. We divide the scatterplot space into 200*200 grids and set the parameters *iterations* as 2, *grid_num* as 20, *n_jump* as 5 and adopt the Strategy1 (overlaps are enclosed by two boundaries) to emphasize overlaps. Without changing other parameters, we set the parameter *proportion* to 0.2, 0.4, 0.6, and 0.8, respectively. Thus, we get four new science maps with different data distribution. Step 3: Label Placement. To reduce visual complexity, we will draw only the boundaries, not the scattered points. Therefore, we set the weight of the distance index to 1, which means that the density index is completely ignored in this case. It is worth noting that since labels are expected to stay away from overlaps, we use the boundaries obtained by applying the strategy3 (overlaps are not enclosed by any boundary) instead of the strategy1 as input for label placement.

As shown in Fig.16, the distribution of each area in new science maps is clearly visible. As the parameter *proportion* is increased, the computer science map moves from having extensive overlaps between areas to preserving only the core of a subset of areas. For example, Fig.16 (a) shows that the AI area is closely related to other areas under the AI category, while its coverage decreases dramatically in (b). This indicates that the AI area shares similar knowledge with several areas, but its unique topic is not conspicuous. HCI, VIS and Computer Graphics share similar characteristics to the AI area. They are all highly interdisciplinary areas. In contrast, the remaining areas in (d),

[2] https://dblp.org
[3] http://csrankings.org/

such as Education, Robotics, Medical, Geography, all have their own unique topics. We note that they are all top disciplines in science.

Exploring the new science maps allows us to understand inter-topics among areas and unique topics of an area. We use G^2 statistics [41] to extract the semantics of a selected local region. Fig.1(b) and (c) respectively illuminate the intersections and specific inter-topics between Rrobotics and HCI, Computer Vision, and Control System. For example, the intersection between Robotics and HCI lies in human-robot intersection, mobile robot, human operator, virtual environment, etc. Fig.16 (d) presents the unique topics of Robotics: control law, robot manipulator, dynamic model, path planning, obstacle avoidance, etc.

7 Discussion and Future Work

Cluster refinement interaction can also be performed at the grid level as with our boundary construction method. The connected component mentioned in the third step of the boundary construction pipeline is a valuable structure to this point. It enables a stroke-based grid-level cluster pickup by two steps. First, indentify the connected components of the filled grids crossed by the stroke; second, merge these connected components. The merged component is the picked cluster. We leave the integral design of the grid-level cluster refinement to the future.

In our current implementation, the boundary construction step sometimes fails to preserve the results of cluster refinement. For example, if a user has merged two separated groups of points in the cluster refinement step, the merged group may still be identified as two continents in the following boundary construction step if the two groups are far apart or lack of obvious connection. There are two simple strategies to overcome this problem. The first strategy is to increase the *n_jump* value in the fourth step of the boundary construction pipeline. The second strategy is to turn the blank grids crossed by strokes into filled grids and ensure that they are not be filtered out in subsequent steps. The merits and drawbacks of these two strategies should be further investigated.

There are several optimization strategies for the label placement step that are worth trying. For example, enlarging the size of labels that have a large distance from the corresponding boundary. Also, some boundaries are like strips, that is, if we use a smallest ellipse to closely enclose the boundary, the difference in length between the long and short axes of the ellipse will be large. In this case, placing labels along the long axis would produce a better visual effect, especially for long labels. We leave these ideas to future work

8 Conclusion

In large-scale multi-class scatterplots, the distribution of classes is usually not discernible due to overdraw problem and semantics are not explicitly expressed, making it difficult for the reader to grasp the rich information hidden in the data. In this paper, we propose a framework to create a visual abstraction for multi-class scatterplots that highlights the distribution and semantics of classes. The framework consists of three steps: clustering, boundary construction, and label placement. In the clustering step, we show that a MST-based clustering algorithm is suitable to remove class-level outliers that act as background noise and to indentify clusters that belong to the same class but are scattered in multiple regions. In particular, we integrate a stroke-based cluster refinement interaction on the algorithm, through which users can quickly materialize desired clusters. To reveal the distribution of data of each cluster, we design a grid-based boundary construction pipeline. It encapsulates multiple outlier detection methods, allowing users to gradually tighten the boundary to the core distribution region of data points by filtering out outliers with varying degrees. Compared with existing methods, our pipeline is more controlled and can general more compact, concise, and readable boundaries. As for the label placement, we note that the boundary of the label itself, as well as the point density of the target cluster, are also important factors affecting the ideal position. The current implementation faces some challenges, such as the slight and local inconsistency between the results of the last two steps. However, overall, the proposed framework helps the user to understand and gain insights into large-scale multi-class scatterplots by highlighting the class-level information.

References

1. V. Agafonkin. A new algorithm for finding a visual center of a polygon. `https://blog.mapbox.com`. (Accessed March 12, 2021).
2. M. Barrault. A methodology for placement and evaluation of area map labels. *Computers, Environment and Urban Systems*, 25(1):33–52, 2001.
3. K. Been, E. Daiches, and C. Yap. Dynamic map labeling. *IEEE Transactions on visualization and computer graphics*, 12(5):773–780, 2006.
4. J. Bernard, M. Hutter, M. Zeppelzauer, D. Fellner, and M. Sedlmair. Comparing visual-interactive labeling with active learning: An experimental study. *IEEE transactions on visualization and computer graphics*, 24(1):298–308, 2017.
5. K. Börner, C. Chen, and K. W. Boyack. Visualizing knowledge domains. *Annual review of information science and technology*, 37(1):179–255, 2003.
6. H. Byelas and A. Telea. Visualization of areas of interest in software architecture diagrams. In *Proceedings of the 2006 ACM symposium on Software visualization*, pp. 105–114, 2006.
7. 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.
8. H. Chen, S. Engle, A. Joshi, E. D. Ragan, B. F. Yuksel, and L. Harrison. Using animation to alleviate overdraw in multiclass scatterplot matrices. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pp. 1–12, 2018.
9. 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, 2019.
10. C. Collins, G. Penn, and S. Carpendale. Bubble sets: Revealing set relations with isocontours over existing visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1009–1016, 2009.

11. S. Elhami, A. Saalfeld, and H. Kang. Using shape analyses for placement of polygon labels. In *Esri International User Conference, San Diego, CA*, 2001.
12. E. R. Gansner, Y. Hu, and S. Kobourov. Gmap: Visualizing graphs and clusters as maps. In *2010 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 201–208. IEEE, 2010.
13. D. Garcia-Castellanos and U. Lombardo. Poles of inaccessibility: A calculation algorithm for the remotest places on earth. *Scottish Geographical Journal*, 123(3):227–233, 2007.
14. A. G. Ghuneim. Moore-neighbor tracing. `http://www.imageprocessingplace.com/downloads\_V3/root_downloads/tutorials/contour\_tracing\_Abeer\_George\_Ghuneim/moore.html`. (Accessed March 12, 2021).
15. M. Goldstein and A. Dengel. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. *KI-2012: Poster and Demo Track*, pp. 59–63, 2012.
16. F. Heimerl, C.-C. Chang, A. Sarikaya, and M. Gleicher. Visual designs for binned aggregation of multi-class scatterplots. *arXiv preprint arXiv:1810.02445*, 2018.
17. F. Heimerl, M. John, Q. Han, S. Koch, and T. Ertl. Docucompass: Effective exploration of document landscapes. In *2016 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 11–20. IEEE, 2016.
18. 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.
19. D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
20. S. G. Kobourov, S. Pupyrev, and P. Simonetto. Visualizing graphs as maps with contiguous regions. In *EuroVis (Short Papers)*, 2014.
21. D. Kohli, A. Cloninger, and G. Mishne. Ldle: Low distortion local eigenmaps. *arXiv preprint arXiv:2101.11055*, 2021.
22. D. Kouřil, L. Čmolík, B. Kozlíková, H.-Y. Wu, G. Johnson, D. S. Goodsell, A. Olson, M. E. Gröller, and I. Viola. Labels on levels: labeling of multi-scale multi-instance and crowded 3d biological environments. *IEEE transactions on visualization and computer graphics*, 25(1):977–986, 2018.
23. F. Krumpe and T. Mendel. Computing curved area labels in near-real time. *arXiv preprint arXiv:2001.02938*, 2020.
24. S. Lespinats, M. Aupetit, and A. Meyer-Baese. Classimap: A new dimension reduction technique for exploratory data analysis of labeled data. *International Journal of Pattern Recognition and Artificial Intelligence*, 29(06):1551008, 2015.
25. S. Lespinats, M. Verleysen, A. Giron, and B. Fertil. Dd-hds: A method for visualization and exploration of high-dimensional data. *IEEE transactions on Neural Networks*, 18(5):1265–1279, 2007.
26. Y. Li, M. Sakamoto, T. Shinohara, and T. Satoh. Automatic label placement of area-features using deep learning. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 43:117–122, 2020.
27. Z. Li, C. Zhang, S. Jia, and J. Zhang. Galex: Exploring the evolution and intersection of disciplines. *IEEE transactions on visualization and computer graphics*, 26(1):1182–1192, 2019.
28. Z. Li, Y. Zhao, N. Botta, C. Ionescu, and X. Hu. Copod: copula-based outlier detection. *arXiv preprint arXiv:2009.09463*, 2020.
29. Y. Liu, E. Jun, Q. Li, and J. Heer. Latent space cartography: Visual analysis of vector space embeddings. In *Computer Graphics Forum*, vol. 38, pp. 67–78. Wiley Online Library, 2019.
30. K. Lu, M. Feng, X. Chen, M. Sedlmair, O. Deussen, D. Lischinski, Z. Cheng, and Y. Wang. Palettailor: Discriminable colorization for categorical data. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):475–484, 2020.
31. M. Lu, S. Wang, J. Lanir, N. Fish, Y. Yue, D. Cohen-Or, and H. Huang. Winglets: Visualizing association with uncertainty in multi-class scatterplots. *IEEE transactions on visualization and computer graphics*, 26(1):770–779, 2019.
32. W. B. March, P. Ram, and A. G. Gray. Fast euclidean minimum spanning tree: algorithm, analysis, and applications. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 603–612, 2010.

33. D. Mashima, S. Kobourov, and Y. Hu. Visualizing dynamic data with maps. *IEEE Transactions on Visualization and Computer Graphics*, 18(9):1424–1437, 2011.
34. A. Mayorga and M. Gleicher. Splatterplots: Overcoming overdraw in scatter plots. *IEEE transactions on visualization and computer graphics*, 19(9):1526–1538, 2013.
35. Y. Meng, H. Zhang, M. Liu, and S. Liu. Clutter-aware label layout. In *2015 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 207–214. IEEE, 2015.
36. A. Moreira and M. Y. Santos. Concave hull: A k-nearest neighbours approach for the computation of the region occupied by a set of points. In *Proceedings of the Second International Conference on Computer Graphics Theory and Applications (GRAPP)*, pp. 61-68. SciTePress, 2007.
37. K. Mote. Fast point-feature label placement for dynamic visualizations. *Information Visualization*, 6(4):249–260, 2007.
38. H. Mumtaz, M. van Garderen, F. Beck, and D. Weiskopf. Label placement for outliers in scatterplots. In *EuroVis (Short Papers)*, pp. 1–5, 2019.
39. T. Pevnỳ. Loda: Lightweight on-line detector of anomalies. *Machine Learning*, 102(2):275–304, 2016.
40. K. Pokonieczny and S. Borkowska. Using artificial neural network for labelling polygon features in topographic maps. *Geoscape*, 13(2):125–131, 2019.
41. P. Rayson and R. Garside. Comparing corpora using frequency profiling. In *The workshop on comparing corpora*, pp. 1–6, 2000.
42. A. Sarikaya and M. Gleicher. Scatterplots: Tasks, data, and designs. *IEEE transactions on visualization and computer graphics*, 24(1):402–412, 2017.
43. S. Sen, A. B. Swoap, Q. Li, B. Boatman, I. Dippenaar, R. Gold, M. Ngo, S. Pujol, B. Jackson, and B. Hecht. Cartograph: Unlocking spatial visualization through semantic enhancement. In *Proceedings of the 22nd international conference on intelligent user interfaces*, pp. 179–190, 2017.
44. A. Sinha, Z. Shen, Y. Song, H. Ma, D. Eide, B.-J. Hsu, and K. Wang. An overview of microsoft academic service (mas) and applications. In *Proceedings of the 24th international conference on world wide web*, pp. 243–246, 2015.
45. J. Stahnke, M. Dörk, B. Müller, and A. Thom. Probing projections: Interaction techniques for interpreting arrangements and errors of dimensionality reductions. *IEEE transactions on visualization and computer graphics*, 22(1):629–638, 2015.
46. J. VanderPlas. mst_clustering: Clustering via euclidean minimum spanning trees. *Journal of Open Source Software*, 1(1):12, 2016. doi: 10.21105/joss.00012
47. Y. Wang, X. Chen, T. Ge, C. Bao, M. Sedlmair, C.-W. Fu, O. Deussen, and B. Chen. Optimizing color assignment for perception of class separability in multiclass scatterplots. *IEEE transactions on visualization and computer graphics*, 25(1):820–829, 2018.
48. C. Wu, Y. Ding, X. Zhou, and G. Lu. A grid algorithm suitable for line and area feature label placement. *Environmental Earth Sciences*, 75(20):1–11, 2016.
49. 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, 2020.
50. Z. Zhou, X. Zhang, Z. Yang, Y. Chen, Y. Liu, J. Wen, B. Chen, Y. Zhao, and W. Chen. Visual abstraction of geographical point data with spatial autocorrelations. In *2020 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 60–71. IEEE, 2020.
51. Y. Zhao, X. Luo, X. Lin, H. Wang, X. Kui, F. Zhou, J. Wang, Y. Chen, and W. Chen. Visual analytics for electromagnetic situation awareness in radio monitoring and management. *IEEE transactions on visualization and computer graphics*, 26(1):590–600, 2019.
52. Y. Zhao, H. Jiang, Y. Qin, H. Xie, Y. Wu, S. Liu, Z. Zhou, J. Xia, F. Zhou, et al. Preserving minority structures in graph sampling. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1698–1708, 2020.