Efficient Online Stream Clustering Based on Fast Peeling of Boundary Micro-Cluster

Jiarui Sun, Mingjing Du, *Member, IEEE*, Chen Sun, and Yongquan Dong

***Abstract*— A growing number of applications generate streaming data, making data stream mining a popular research topic. Classification-based streaming algorithms require pre-training on labeled data. Manually labeling a large number of samples in the data stream is impractical and cost-prohibitive. Stream clustering algorithms rely on unsupervised learning. They have been widely studied for their ability to effectively analyze high-speed data streams without prior knowledge. Stream clustering plays a key role in data stream mining. Currently, most data stream clustering algorithms adopt the online–offline framework. In the online stage, micro-clusters are maintained, and in the offline stage, they are clustered using an algorithm similar to density-based spatial clustering of applications with noise (DBSCAN). When data streams have clusters with varying densities and ambiguous boundaries, traditional data stream clustering algorithms may be less effective. To overcome the above limitations, this article proposes a fully online stream clustering algorithm called fast boundary peeling stream clustering (FBPStream). First, FBPStream defines a decay-based kernel density estimation (KDE). It can discover clusters with varying densities and identify the evolving trend of streams well. Then, FBPStream implements an efficient boundary micro-cluster peeling technique to identify the potential core micro-clusters. Finally, FBPStream employs a parallel clustering strategy to effectively cluster core and boundary micro-clusters. The proposed algorithm is compared with ten popular algorithms on 15 data streams. Experimental results show that FBPStream is competitive with the other ten popular algorithms.**

***Index Terms*— Boundary peeling, data stream, density-based clustering, online clustering.**

Nomenclature

Notation	Definition
$\boldsymbol{X} = \{x_1, x_2, \ldots, x_n\}$	Data stream.
n	Number of data points in $\boldsymbol{X}$.
d	Dimensionality of data points in $\boldsymbol{X}$.
s	Seed point of a micro-cluster, see Definition 2.
r	Radius of micro-clusters.
MF	Micro-cluster feature vector, see Definition 1.
λ	Decay factor.
β	Weight threshold factor, see (7).
k	Number of nearest neighbors.
ω	Level of peeling.
m	Number of intermediate macro-clusters.
K	Number of result clusters.
$W(x_i, t)$	Weight of x_i at t, see (1).
$W(\mathrm{mc}, t)$	Weight of mc at t, see (3)–(5).
$\mathrm{DKD}(\mathrm{mc}, t)$	Decay-based kernel density of mc at t, see Definition 3.
$\boldsymbol{V}_{\mathrm{core}}$	Set of core micro-clusters.
$\boldsymbol{V}_{\mathrm{border}}$	Set of boundary micro-clusters.
$\mathrm{Rep}(\mathrm{mc}, t)$	Representative of mc at t, see Definition 4.
$\boldsymbol{S_m} = \{\boldsymbol{C}_1, \boldsymbol{C}_2, \ldots, \boldsymbol{C}_m\}$	Set of intermediate macro-clusters.
$\boldsymbol{S_C} = \{\boldsymbol{C}_1, \boldsymbol{C}_2, \ldots, \boldsymbol{C}_K\}$	Set of result clusters.

Manuscript received 12 June 2023; revised 20 November 2023 and 3 February 2024; accepted 15 March 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62006104 and Grant 61872168 and in part by the Natural Science Foundation of Jiangsu Higher Education Institutions under Grant 20KJB520012. *(Corresponding author: Mingjing Du.)*

The authors are with the School of Computer Science and Technology, Jiangsu Normal University, Xuzhou 221116, China (e-mail: sunjr@jsnu.edu.cn; dumj@jsnu.edu.cn; 3020200309@jsnu.edu.cn; tomdyq@163.com).

This article has supplementary downloadable material available at https://doi.org/10.1109/TNNLS.2024.3382033, provided by the authors.

Digital Object Identifier 10.1109/TNNLS.2024.3382033

I. Introduction

OWING to the rapid development of the Internet and the Internet of Things (IoT), an immense volume of data, often termed as data streams, is continuously produced at near real-time pace. These data streams originate from a diverse range of sources such as social media, sensors, and financial transactions. Extracting valuable information from continuous data streams in real time is crucial for forecasting future events and making timely decisions [1], [2], [3]. Clustering is a well-suited approach for real-time data stream processing, because it requires less a priori information and no labeled instances [4], [5], [6].

Existing data stream clustering algorithms either adopt the fully online framework or the two-stage (online-offline) framework [7], [8], [9]. Most algorithms use the latter [10]. For a two-stage algorithm, the system receives the data and summarizes it into micro-clusters or grid cells in the online stage. In the offline stage, the micro-clusters (grid cells) are viewed as pseudo-points. They are reclustered or merged according to some traditional clustering algorithm (sometimes with slight modifications) to form the final clusters. Clustering algorithms commonly used in data stream scenarios include k-means, density-based spatial clustering of applications with noise (DBSCAN) [11], grid-based algorithms, etc. Although the two-stage framework is widely used, it has some inherent limitations: 1) the two-stage algorithms cannot accurately capture the drifting features [12] in the data stream because they do not implement real-time processing and 2) the two-stage algorithms cannot respond quickly to high-speed evolving data streams, etc. Considering the limitations of the above two-stage algorithms, some fully online stream clustering algorithms have been developed [2], [13], [14], [15], [16]. These algorithms use a tree or graph structure that is updated and searched in real time, enabling fast responses to clustering requests. The fully online algorithms improve clustering efficiency and quality to a certain extent.

In many real-world scenarios, different clusters in the data stream often have different densities and there may be ambiguous boundaries between clusters. While this is a well-recognized problem, most existing stream clustering algorithms either focus only on the identification of clusters with varying densities [17], [18] or on the detection of clusters with ambiguous boundaries [13], [19]. To our knowledge, none of the existing stream clustering algorithms tackle the problem in a unified way. To solve the above problem, this article proposes a fully online stream clustering algorithm called **F**ast **B**oundary **P**eeling **Stream** Clustering (FBPStream[1]). To utilize FBPStream, we introduce a decay-based KDE, which takes into account both the temporal and spatial distribution of each micro-cluster. Then, an efficient boundary micro-cluster peeling clustering strategy based on the decay-based kernel density (DKD) is used to reveal the potential core micro-clusters by peeling off the boundary micro-clusters, which improves the efficiency and accuracy of clustering. Experimental results on a wide range of synthetic and real-world data streams show that FBPStream can effectively tackle the above problem in a unified manner.

In summary, the main contributions of this article are as follows.

1) A fully online stream clustering algorithm FBPStream is proposed, which can simultaneously guarantee the clustering efficiency and quality.
2) A decay-based KDE is proposed. It can discover clusters with varying densities and identify the evolving trend of streams well.
3) An efficient boundary micro-cluster peeling clustering strategy is proposed to improve the clustering quality of clusters with ambiguous boundaries.

The rest of this article is organized as follows. Section II reviews studies related to our work. The relevant fundamental concepts and definitions are introduced in Section III. Section IV gives a high-level overview in terms of the general framework and process. The key techniques utilized in the proposed algorithm are described in detail in Section V. The experimental results on synthetic and real-world datasets are thoroughly analyzed in Section VI. Finally, the article is summarized in Section VII.

II. Related Work

As mentioned in Section I, most existing stream clustering algorithms follow the two-stage framework [20], [21], [22], [23]. These algorithms receive data in the online stage and summarize the original data into micro-clusters or grid cells. In the offline stage, micro-clusters (grid cells) are represented as pseudo-points and are reclustered or merged according to some traditional clustering algorithms (sometimes with slight modifications) to form the final clusters.

According to the traditional clustering methods used in the offline stage, these two-stage algorithms can be classified into partition-based clustering algorithms [10], [24], [25], hierarchy-based clustering algorithms [26], [27], [28], density-based clustering algorithms [19], [29], [30], [31], [32], and grid-based clustering algorithms [18], [33], [34], [35]. Among them, density-based and grid-based clustering algorithms have the advantages of identifying arbitrarily shaped clusters and detecting outliers. For example, Cao et al. [29] propose the DenStream algorithm, which defines the reachability between core micro-clusters based on the idea of DBSCAN. It connects all micro-clusters whose distance is smaller than the reachable distance threshold to form the final clusters. HDDStream [30] addresses the clustering problem for high-dimensional streaming data with density-based projection. The concept of the shared density map is introduced in the DBSTREAM [19], which improves the clustering quality by explicitly capturing the density of the original data between micro-clusters. DWDP-Stream [17] is a recently proposed algorithm based on density peak clustering (DPC) [36]. It introduces natural neighbors and an improved allocation process to enhance clustering performance. Chen and Tu [33], [34] proposed D-Stream, a density grid-based clustering algorithm, which clusters arbitrarily shaped data streams by merging adjacent dense grids and transition grids. Based on the idea of D-Stream, MR-Stream [18] employs a multiresolution grid technique to improve the clustering quality. MuDi-Stream [31] clusters stream data using micro-clusters and grids. The grid-based method is used as an outlier buffer to handle noise and multidensity data. However, these two-stage algorithms do not achieve real-time processing in the true sense of the word because they are based on timed processing of short batches. In response to high-speed evolving data streams, they may miss concept drifts. For instance, during network traffic monitoring, malicious network activities tend to evolve and change over time. If the algorithm does not detect concept drifts frequently enough, it may miss new concepts and patterns in malicious network traffic. As a result, novel forms of network attacks or anomalies may not be detected, which may threaten network security.

A portion of the stream clustering algorithms adopt the fully online framework. For example, CEDAS [13] employs a graph structure to improve the efficiency of clustering. In the first stage of the algorithm, new points are added to existing micro-clusters or are used to form new micro-clusters, and micro-cluster information is updated. Then, it searches for overlapping micro-clusters in the second stage and specifies that each macro-cluster consists of a graph of intersecting micro-clusters. EDMStream [14] is a fully online density-based stream clustering algorithm developed by Gong et al. It is an online version of the traditional DPC. It maintains a dependency tree (DP-Tree) consisting of all active micro-clusters incrementally in memory and cuts tree by adaptively adjusting a minimum connection threshold.

[1]Code available: https://github.com/Du-Team/FBPStream.

After the cut, each subtree forms a single cluster. In addition, Li et al. propose the ESA-Stream algorithm [16], which is a fully online grid-based stream clustering algorithm. The algorithm uses a parameter adaptive technique that automatically adjusts the parameters to improve the quality of the clustering results. It also uses an efficient dimensionality reduction technique based on the grid density centroid to reduce the dimension of high-dimensional data, which greatly improves the efficiency of clustering.

The density-based and grid-based stream clustering algorithms mentioned above are capable of clustering arbitrarily shaped data streams. However, their clustering effectiveness is reduced when the data streams have clusters with varying densities and ambiguous boundaries. To solve similar problems in the traditional clustering domain, Averbuch-Elor et al. [37] propose the BP algorithm. They develop a technique to iteratively identify the border points and peel off them to reveal the potential cluster cores. Du et al. [38] propose the ROBP algorithm, which employs a density estimation based on the Cauchy kernel and a linkage criterion based on the shared neighborhood information. ROBP further improves the clustering accuracy and efficiency on the basis of BP. DCF [39] employs the mutual k-NN graph to optimize the DPC algorithm, allowing it to discover peaks of clusters with varying densities. Li et al. [40] propose the LGD algorithm, which introduces the concept of local gap density to handle high-dimensional data with varying densities. Although the above algorithms have the ability to detect clusters with varying densities or ambiguous boundaries, they cannot be applied in data streaming.

III. Fundamental Concepts

Many devices are constantly generating data streams. Processing such enormous amounts of data offline requires much storage space and computing power. Therefore, real-time data stream processing is a great solution [41], [42], [43]. A data stream is a massive, unbounded, ordered sequence of arriving data points. A data stream can be represented formally as $\boldsymbol{X} = \{x_1, x_2, \ldots, x_n\}$, where x_i denotes the ith arrived d-dimensional data point, $1 \leq i \leq n$, $1 \leq d \in \mathbb{Z}^+$, and $n \to \infty$.

Clustering for evolving data streams often requires using specific window models to capture the concept drifts of the data stream. There are three well-known window models: landmark window, sliding window, and damped window [5], [29]. Our algorithm is based on a damped window model. In this study, an exponential decay function widely used in temporal applications is chosen as the decay function.

Each data point x_i in the data stream is assigned a weight, which gradually decays over time. If x_i arrives at time t_c, the weight of x_i at time t ($t \geq t_c$) is denoted as $W(x_i, t)$, defined as follows:

$$W(x_i, t) = \lambda^{t-t_c} \tag{1}$$

where $\lambda \in (0, 1)$ is a constant known as the *decay factor*. The higher the λ value, the faster the data weights decay, i.e., the less influence the old data has on the clustering results.

Given the enormity of data streams, storing them in their entirety becomes impractical. Consequently, employing data compression techniques becomes necessary. Similar to DenStream [29] and DBSTREAM [19] algorithms, our algorithm uses the micro-cluster structure to summarize the original data points. A visualization of the micro-cluster structure is presented in the Supplementary Material. The micro-cluster structure is expressed as a feature vector MF with five attributes, which facilitates its dynamic maintenance.

Definition 1 (MF): At time t, for a micro-cluster mc, its feature vector is defined as

$$\text{MF}_{\text{mc}} = (s, r, W(\text{mc}, t), t_u, \text{status}) \tag{2}$$

where s is the seed point of mc (see Definition 2). r is the predefined radius of all micro-clusters. $W(\text{mc}, t)$ denotes the weight of mc at time t [see (3)]. t_u is the last update time of mc. status indicates whether mc is active [see (6)].

Our algorithm uses an actual data point, called a seed point, to create and represent a micro-cluster, instead of the micro-cluster center defined in DenStream [29]. The definition of a seed point is as follows.

Definition 2 (Seed Point, s): If a data point, denoted as s, arriving at time t, does not fall within a predefined radius of any existing micro-cluster, it is designated as a seed point for the construction of a new micro-cluster.

Each point is assigned to the micro-cluster of its nearest seed point, provided that the distance is within the radius threshold r.

For a micro-cluster mc, at a given time t_c, let $V(\text{mc}, t_c)$ be the set of data points that are summarized into the mc at or before t_c. Then, the weight $W(\text{mc}, t_c)$ of mc is defined as the sum of the weights of all data points in $V(\text{mc}, t_c)$, defined as follows:

$$W(\text{mc}, t_c) = \sum_{x_i \in V(\text{mc}, t_c)} W(x_i, t_c). \tag{3}$$

The weight of each micro-cluster is constantly changing due to the constant arrival of data and the concept drift [44], [45], [46] that occurs along with it. When a new data point x_o arrives at time t, if it is not within any existing micro-cluster, a new micro-cluster is established by x_o (see Definition 2). If the distance between an existing micro-cluster mc and the new data point x_o is minimal and smaller than a predefined radius, mc will absorb x_o and update weight as follows:

$$W(\text{mc}, t) = \lambda^{t-t_c} \sum_{x_i \in V(\text{mc}, t_c)} W(x_i, t_c) + W(x_o, t). \tag{4}$$

Specifically, recalculate the sum of the weights of all data points within the micro-cluster. By applying (1) and (3) to (4), we can derive

$$W(\text{mc}, t) = \lambda^{t-t_c} W(\text{mc}, t_c) + 1 \quad (t > t_c) \tag{5}$$

where $W(\text{mc}, t_c)$ is the weight of mc at the last moment t_c before t.

In order to improve the readability, Nomenclature lists the major symbols and notations used in the article.

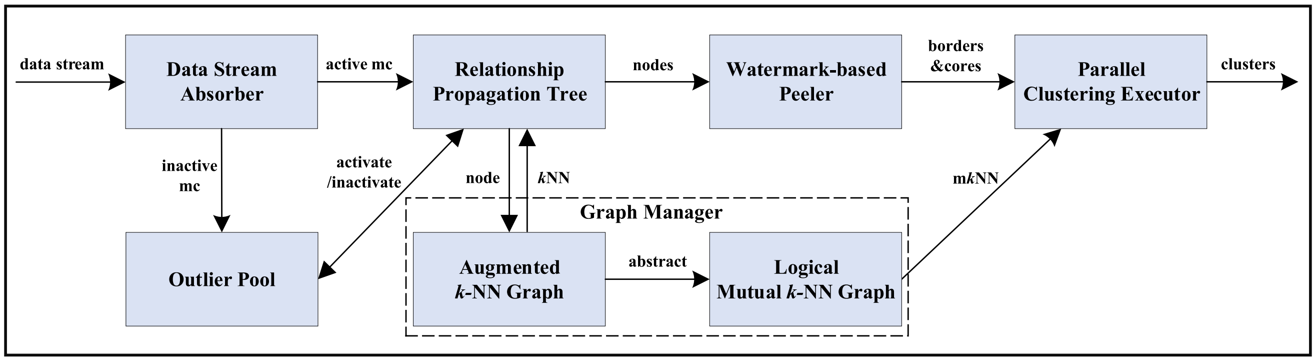


Fig. 1. Framework of FBPStream.

IV. Overview of FBPStream

A. Framework

The proposed FBPStream framework is outlined in Fig. 1. With the interplay of the six components, FBPStream can efficiently cluster data streams in real time. The descriptions of all the components are given below.

1) *Data Stream Absorber:* It receives each data point from the data stream and summarizes it into an existing micro-cluster (or creates a new one) (see Section III for details).
2) *Outlier Pool:* It caches inactive micro-clusters which cannot be removed directly from memory because they may be activated in the future (see Section V-A for details).
3) *Relationship Propagation Tree (RP-Tree):* It collects active micro-clusters and transforms them into nodes with k-nearest neighbor information for subsequent clustering.
4) *Graph Manager:* It contains an *augmented k-NN graph* and a *logical mutual k-NN graph* (see Section V-B for details). The *augmented k-NN graph* maintains at most αk-nearest neighbors incrementally for each active micro-cluster. It is used to accelerate the update of the k-NN graph. An abstraction of the *augmented k-NN graph* yields the *logical mutual k-NN graph*, which is the basis of the entire algorithm.
5) *Watermark-Based Peeler:* It rapidly peels off boundary micro-clusters from active micro-clusters and reveals potential core micro-clusters (see Section V-D for details).
6) *Parallel Clustering Executor:* It efficiently clusters the active micro-clusters in the *RP-Tree* based on the underlying logical structure (*logical mutual k-NN graph*) and outputs the clustering results (see Section V-E for details).

B. Workflow

To facilitate the understanding of the execution flow of FBPStream, a comprehensive flowchart of the proposed algorithm is provided in the Supplementary Material. For a more detailed elaboration of the proposed algorithm, please see the next section. Based on the above overview of FBPStream, the following unique features can be obtained, which are verified in the subsequent theoretical analyses and experimental results.

1) *Real-time:* It processes the data stream in real-time in a fully online fashion.
2) *Effective:* It peels off boundary micro-clusters based on the DKD to reveal potential core micro-clusters. This idea facilitates detecting clusters with varying densities and ambiguous boundaries.
3) *Efficient:* It uses a variety of acceleration mechanisms in the fully online state, including the fast peeling mechanism, updating the *augmented k-NN graph*, extracting the *logical mutual k-NN graph* and the parallel clustering mechanism, etc.

V. Proposed FBPStream

In this section, we first explain the key techniques and concepts of the important components of FBPStream. Then, we give the complete pseudo-code of the algorithm and analyze its complexity in detail.

A. Detection of Outliers

In FBPStream, we consider micro-clusters with lower weights as outliers [47], [48], [49]. In general, micro-clusters may become outliers due to two factors: they absorb few data points, or they may not absorb new data points for long periods, causing them to become obsolete. Instead of deleting these outliers immediately, they are temporarily stored in the *Outlier Pool* [14], [29] and considered inactive (i.e., inactive micro-clusters). When an outlier in the *Outlier Pool* has absorbed enough data points from the data stream, it is still possible to transform into a non-outlier (i.e., an active micro-cluster). Such activated micro-clusters are inserted into the *RP-Tree*. Clustering is performed only on the active micro-clusters in the *RP-Tree*.

Cao et al. [29] and Chen and Tu [33] prove that when $t \to \infty$, the sum of the weights of all data points in an unbounded data stream is a constant $1/(1-\lambda)$. Based on the above knowledge, we give the following definition to distinguish active and inactive micro-clusters:

$$\text{status} = \begin{cases} \text{active}, & \text{if } W(\text{mc}, t) \geq \beta \big/ \big(1-\lambda\big) \\ \text{inactive}, & \text{otherwise} \end{cases} \quad (6)$$

where status denotes whether mc is active at time t. β is a factor that controls the weight threshold. It is easily known

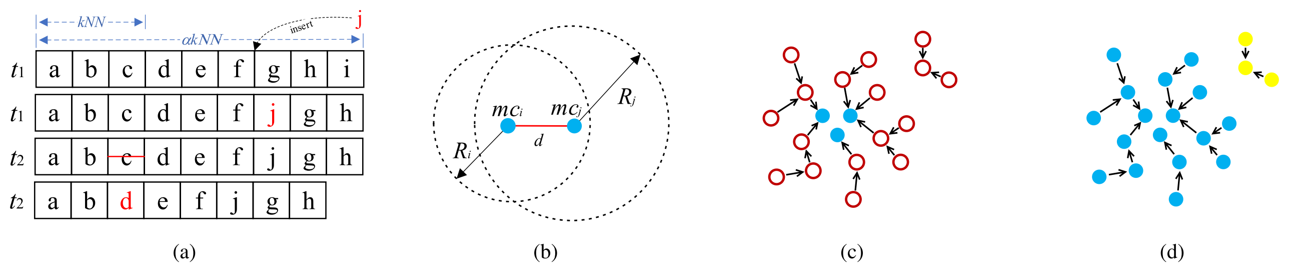


Fig. 2. Examples of some key techniques. (a) Update of the *augmented k-NN graph*. (b) Extraction of the *logical mutual k-NN graph*. (c) Linkage of boundary micro-clusters, where red circles are boundary micro-clusters, blue points are core micro-clusters, and arrows point to representatives. (d) Assignment of boundary micro-clusters and two clusters are generated (blue and yellow points, respectively).

that the larger the β value, the less the number of active micro-clusters.

In addition, when a new micro-cluster is created by a seed point, the new micro-cluster should be inactive. Therefore, the range of β can be derived from the following equation:

$$1 < \beta/(1-\lambda) < 1/(1-\lambda)$$
$$\Rightarrow 1-\lambda < \beta < 1. \quad (7)$$

By using the outlier detection technique, we are able to control the number of active micro-clusters and ignore the inactive micro-clusters with lower weights. As a result of this technique, the algorithm is robust to outliers and noise in the data stream. A cleanup strategy consistent with EDM-Stream [14] is employed for the *Outlier Pool* to recycle memory space.

B. Graph Manager

As shown in Fig. 1, FBPStream runs on the *Graph Manager*. The main role of the *Graph Manager* is to ensure the efficient output of the mutual k-NN graph, which is a logical structure extracted from an augmented k-NN graph.

1) Augmented k-NN Graph: We generalize the k-NN graph to the data stream environment, where each vertex in the graph is an active micro-cluster. In a high-speed evolutionary data stream environment, the emergence and extinction of active micro-clusters will directly change the structure of the k-NN graph. To cope with the negative effects of frequent updates of the k-NN graph, we design the *augmented k-NN graph* to accelerate the update operation. It maintains at most αk-nearest neighbors (generally $\alpha = 3$) incrementally for each active micro-cluster.

We give an example in Fig. 2(a) to illustrate the principle of updating the *augmented k-NN graph* (including insertion and deletion). Before time t_1, **a**~**i** are the active micro-clusters in the *RP-Tree*. They are arranged in ascending order of distance to a given active micro-cluster mc. They form the list of αk-nearest neighbors of mc ($\alpha = 3, k = 3$), where the k-nearest neighbors of mc are {**a**, **b**, **c**}.

Assume that at time t_1, the inactive micro-cluster **j** grows into an active micro-cluster so that it will be inserted into the *RP-Tree*. Correspondingly, the *augmented k-NN graph* needs to be updated. As shown in Fig. 2(a), to ensure the orderliness of the list of αk-nearest neighbors of mc, the *Binary Insertion* method is used to speed up the insert operation of **j**. This method can quickly detect the position to be inserted. Thus, **j** is eventually inserted after **f**, and **i** is moved out of the list accordingly.

Suppose that the active micro-cluster **c** decays to an inactive micro-cluster at time t_2 as the data stream evolves. Then, it will be removed from the k-nearest neighbors of mc. In this case, the kth nearest neighbor of mc needs to be redetermined. The rudimentary approach is to recalculate the distance from mc to each active micro-cluster. However, the approach is inefficient when it occurs frequently. To overcome the above problem, we augment the k-nearest neighbors to αk-nearest neighbors. When a micro-cluster is removed from the first k-nearest neighbors due to decay, the nearest neighbor behind it (the original $(k+1)$st nearest neighbor) is directly filled into the missing position without additional computation. This strategy counteracts the negative impact of updating the k-NN graph on the efficiency of the clustering algorithm execution. As shown in Fig. 2(a), after **c** is removed, **d** fills the missing position and becomes the kth nearest neighbor of mc.

2) Logical Mutual k-NN Graph: We can extract the mutual k-NN graph from the *augmented k-NN graph*. As shown in Fig. 2(b), assume that mc_i and mc_j are two active micro-clusters in the *RP-Tree* at time t. We set $R = \|\mathrm{mc} - N_k(\mathrm{mc}, t)\|$, where $\|\cdot\|$ is the Euclidean distance between two micro-clusters (i.e., the distance between the seed points of two micro-clusters). $N_k(\mathrm{mc}, t)$ is the kth nearest micro-cluster to mc at time t.

If $\|\mathrm{mc}_i - \mathrm{mc}_j\| \leq \min(R_i, R_j)$, then mc_i and mc_j are mutual k-nearest neighbors. That is, $\mathrm{mc}_j \in mk\mathrm{NN}(\mathrm{mc}_i, t)$ or $\mathrm{mc}_i \in mk\mathrm{NN}(\mathrm{mc}_j, t)$.

Through the above update and extraction mechanisms, the latest mutual k-NN graph can be obtained in real-time in the rapidly evolving data stream. It supports each of the key techniques below.

C. Decay-Based Kernel Density

In the clustering phase, most data stream clustering algorithms only use a fixed distance threshold to cluster active micro-clusters that are close to each other. When clusters have varying densities, these algorithms may be less effective. In contrast to traditional data stream clustering algorithms, we develop a robust DKD estimator for the active micro-clusters. The estimator not only captures the time-varying weights of the micro-clusters but also takes into account the spatial distribution of the micro-clusters.

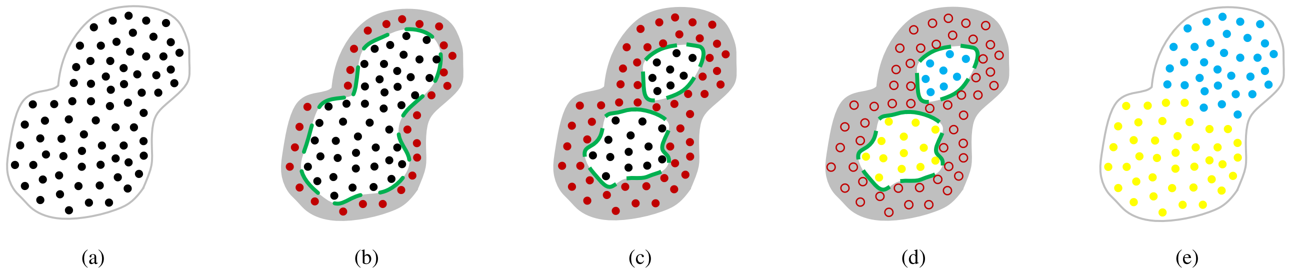


Fig. 3. Example of the boundary micro-cluster peeling clustering strategy ($\omega = 2$). (a) Active micro-clusters in the *RP-Tree*, where seed points (black points) are used to represent micro-clusters. (b) Boundary micro-clusters (red points) of the first layer are peeled off (i.e., $l = 1$). (c) Boundary micro-clusters of the second layer continue to be peeled off (i.e., $l = 2$). (d) Core micro-clusters are clustered, and two macro-clusters are obtained (blue and yellow points, respectively). (e) Boundary micro-clusters are assigned, and the clustering results are obtained (blue and yellow points, respectively).

Kernel density estimation (KDE) is a widely used density measure. Regarding the choice of the kernel function, we adopt a widely used smooth kernel function, the Gaussian kernel function. In addition, designing a suitable bandwidth is crucial for the density estimator. Assuming a fixed bandwidth is utilized, the following problems may be encountered: 1) it is challenging to choose a suitable global bandwidth manually and 2) a fixed bandwidth means that the densities of points in sparse regions are lower than in dense regions. It does not facilitate discovering clusters with varying densities.

To overcome the above problems, we adopt a general adaptive scheme similar to [50] and utilize mutual k-nearest neighbors (mk-NN) to capture the distribution information of the data. Our system works on micro-clusters, not original data points. Therefore, we define the micro-cluster-based Gaussian kernel density as

$$\mathrm{MKD}(\mathrm{mc}_i, t) = \sum_{\mathrm{mc}_j \in mk\mathrm{NN}(\mathrm{mc}_i, t)} \exp\left(\frac{-\|\mathrm{mc}_i - \mathrm{mc}_j\|^2}{R_j^2}\right) \tag{8}$$

where $mk\mathrm{NN}(\mathrm{mc}_i, t)$ is a set of mutual k-nearest neighbor micro-clusters of mc_i at time t. R_j is the distance from mc_j to its kth nearest neighbor at time t.

Definition 3 (DKD): Let mc_i be a micro-cluster at time t. After considering the weight and the MKD (micro-cluster-based Gaussian kernel density) of mc_i together, the DKD of mc_i is defined as

$$\mathrm{DKD}(\mathrm{mc}_i, t) = W(\mathrm{mc}_i, t) \cdot \mathrm{MKD}(\mathrm{mc}_i, t). \tag{9}$$

The DKD integrates information about the weights and spatial distribution of micro-clusters. A higher density indicates that the micro-cluster has a higher weight and a more dense distribution. Therefore, it can discover clusters with varying densities and identify the evolving trend of streams well.

D. Watermark-Based Peeler

Most density-based stream clustering algorithms use a variant of DBSCAN to perform simple clustering of micro-clusters. These algorithms may be less effective when clusters have varying densities and ambiguous boundaries. Our proposed FBPStream uses an efficient boundary micro-clusters peeling strategy to improve the clustering results. It can reveal potential core micro-clusters by fast identifying and peeling off boundary micro-clusters with lower density. Fig. 3 illustrates the process of peeling off boundary micro-clusters.

In order to adapt the peeling process to the distribution of different datasets, we introduce the concept of *watermark* (ω for short). It is an integer hyperparameter greater than or equal to 0 set by the user. More intuitively, it indicates the level of peeling. A higher *watermark* value means a higher level of peeling, meaning more active micro-clusters are peeled as boundary micro-clusters.

As shown in Algorithm 1, we give the procedure of the algorithm for fast peeling based on *watermark*. During the peeling process of each layer, it first calculates the average density (Line 4). Then, micro-clusters with densities less than the average are considered boundary micro-clusters. Finally, it fast peels the boundary micro-clusters off to reveal the potential core micro-clusters with higher density [Lines 5–10, corresponding to Fig. 3(b) and (c)]. In particular, it should be noted that when the user sets watermark = 0, the algorithm does not perform the peeling process (i.e., it skips Lines 3–11) and considers all active micro-clusters in the *RP-Tree* as core micro-clusters (Lines 1 and 2).

Algorithm 1 FastPeeling

```
Input:  $V = \{mc | mc \in RP\text{-}Tree\}$,
        level of peeling $\omega$;
Output: A set of boundary micro-clusters $V_{border}$,
        a set of core micro-clusters $V_{core}$;
1  $V_{border} \leftarrow \emptyset$;
2  $V_{core} \leftarrow V$;
3  for $l = 1; l \le \omega; l{+}{+}$ do
4      $\rho^{(l)} = \frac{\sum_{mc \in V_{core}} DKD(mc, t)}{|V_{core}|}$;
5      foreach $mc \in V_{core}$ do
6          if $DKD(mc, t) < \rho^{(l)}$ then
7              $V_{core} \leftarrow V_{core} \setminus \{mc\}$;
8              $V_{border} \leftarrow V_{border} \cup \{mc\}$;
9          end
10     end
11 end
12 return $V_{border}$ and $V_{core}$.
```

Compared to the processing idea of [37] and [38], our peeling strategy differs from it in three ways. First, FBPStream maintains the density of each active micro-cluster incrementally in memory. Thus, the density of each micro-cluster can be obtained directly without any computational cost during the peeling process. Second, FBPStream employs *watermark* to control the number of layers peeled strictly, and the density threshold for peeling is calculated directly in each layer.

Third, FBPStream peels each layer directly based on the density thresholds instead of a percentage. The effectiveness of our boundary micro-cluster peeling strategy is demonstrated later in the experimental results.

E. Parallel Clustering Executor

After the peeling process, the active micro-clusters in the *RP-Tree* are divided into boundary micro-clusters and core micro-clusters. When cluster boundaries are ambiguous, we first cluster the core regions separated by boundary regions to prevent incorrect clustering. Clustering core micro-clusters yield intermediate macro-clusters. Then, we try to assign the boundary micro-clusters to the existing macro-clusters or form new clusters to obtain the final clustering results.

1) Clustering Core Micro-Clusters: Algorithm 2 shows the detailed process of clustering core micro-clusters.

Algorithm 2 CoreClustering

```
Input:  A set of core micro-clusters $\boldsymbol{V}_{core}$,
        the Logical Mutual k-NN Graph;
Output: A set of intermediate macro-clusters
        $\boldsymbol{S_m} = \{\boldsymbol{C}_1, \boldsymbol{C}_2, \ldots, \boldsymbol{C}_m\}$;
1  $\boldsymbol{S_m} \leftarrow \emptyset$;
2  if $|\boldsymbol{V}_{core}| = 1$ then
3      if $mc.cluster = null$ then                // $mc \in \boldsymbol{V}_{core}$
4          $\boldsymbol{C}_1 \leftarrow$ create a new empty macro-cluster;
5          $\boldsymbol{C}_1 \leftarrow \boldsymbol{C}_1 \cup \{mc\}$;
6          $\boldsymbol{S_m} \leftarrow \boldsymbol{S_m} \cup \{\boldsymbol{C}_1\}$;
7          return $\boldsymbol{S_m}$;
8      end
9  end
10 foreach $mc \in \boldsymbol{V}_{core}$ do
11     if $mc.visited = True$ then
12         continue;
13     end
14     $mc.visited \leftarrow True$;
15     if $mc.cluster = null$ then
16         $\boldsymbol{C}_t \leftarrow$ create a new empty macro-cluster;
17         $\boldsymbol{C}_t \leftarrow \boldsymbol{C}_t \cup \{mc\}$;
18         $\boldsymbol{S_m} \leftarrow \boldsymbol{S_m} \cup \{\boldsymbol{C}_t\}$;
19     end
20     $\boldsymbol{Q} \leftarrow$ create a new empty queue;
21     $\boldsymbol{Q} \leftarrow \boldsymbol{Q} \cup \{mc\}$;
22     while $|\boldsymbol{Q}| > 0$ do
23         $mc_i \leftarrow \boldsymbol{Q}.poll()$;
24         foreach $mc_j \in mkNN(mc_i, t)$ do
25             if $mc_j \in \boldsymbol{V}_{core}$ and $mc_j.visited = False$ then
26                 $mc_j.visited \leftarrow True$;
27                 $\boldsymbol{Q} \leftarrow \boldsymbol{Q} \cup \{mc_j\}$;
28                 Add $mc_j$ to $mc_i.cluster$;
29             end
30         end
31     end
32 end
33 return $\boldsymbol{S_m}$.
```

When only one core micro-cluster exists, only one macro-cluster contains the core micro-cluster (Lines 2–9). If there are multiple core micro-clusters, then we perform a breadth-first traversal of the *logical mutual k-NN graph* with the help of a queue. The core micro-clusters that are mutual k-nearest neighbors are merged into the same macro-cluster (Lines 10–32). Eventually, the intermediate macro-clusters $\boldsymbol{S_m} = \{\boldsymbol{C}_1, \boldsymbol{C}_2, \ldots, \boldsymbol{C}_m\}$ are generated. Fig. 3(d) shows an example of clustering core micro-clusters.

2) Assigning Boundary Micro-Clusters: Next, we try to assign the peeled boundary micro-clusters to the macro-clusters $\boldsymbol{C}_1, \boldsymbol{C}_2, \ldots, \boldsymbol{C}_m$ generated by core micro-clusters according to a specific rule. The rule is to construct a path (called a chain) for each boundary micro-cluster that connects to a core micro-cluster. However, these paths may not eventually connect to core micro-clusters. For this case, we will explain in detail below.

In order to generate such a chain for each boundary micro-cluster, we first need to explore an appropriate local relationship for each boundary micro-cluster, which we call a representative. It is defined as follows.

Definition 4 (Representative, Rep): At time t, there are two active micro-clusters mc_i and mc_j in the *RP-Tree*. If $\mathrm{mc}_j \in mk\mathrm{NN}(\mathrm{mc}_i, t)$ and mc_j is the nearest micro-cluster to mc_i with higher density, then mc_j is the representative of mc_i, denoted as $\mathrm{Rep}(\mathrm{mc}_i, t) = \mathrm{mc}_j$.

Property 1: There are no loops in these chains [51], i.e., each chain is a directed acyclic graph. A proof of acyclicity is given in the Supplementary Material. This property of acyclicity ensures the feasibility of assigning boundary micro-clusters.

As shown in Algorithm 3, we give the detailed procedure for finding a representative for each boundary micro-cluster at time t. Since we do not require that the representatives be core micro-clusters, they may be boundary micro-clusters as well. In this way, the emergence of new clusters can be captured effectively.

Algorithm 3 BorderLinkage

```
Input:  A set of boundary micro-clusters $\boldsymbol{V}_{border}$,
        the Logical Mutual k-NN Graph;
Output: A set of boundary micro-clusters $\boldsymbol{V}_{border}$;
                                    // with representatives
1 foreach $mc_i \in \boldsymbol{V}_{border}$ do
2     foreach $mc_j \in mkNN(mc_i, t)$ do
3         if $DKD(mc_j, t) > DKD(mc_i, t)$ then
4             $Rep(mc_i, t) \leftarrow mc_j$;
5             break;
6         end
7     end
8 end
9 return $\boldsymbol{V}_{border}$.
```

In the example depicted in Fig. 2(c), a chain is generated for each boundary micro-cluster after the representatives of all boundary micro-clusters are determined. The start of a chain is a boundary micro-cluster (with the lowest density), and the end of a chain may be a core micro-cluster or a boundary micro-cluster (with the highest density).

In a chain, the micro-cluster with the highest density determines the category labels of all boundary micro-clusters. Therefore, we define the following relationship propagation rule.

Definition 5 (Relationship Propagation Rule, RPR): At time t, there is a chain consisting of different micro-clusters $\mathrm{mc}_1, \mathrm{mc}_2, \ldots, \mathrm{mc}_n$. Among these micro-clusters, there are the following relationships: $\mathrm{Rep}(\mathrm{mc}_1, t) = \mathrm{mc}_2, \mathrm{Rep}(\mathrm{mc}_2, t) = \mathrm{mc}_3, \ldots, \mathrm{Rep}(\mathrm{mc}_{n-1}, t) = \mathrm{mc}_n$, where $\mathrm{mc}_1, \mathrm{mc}_2, \ldots, \mathrm{mc}_{n-1}$ are boundary micro-clusters. If mc_n is a core micro-cluster, micro-clusters $\mathrm{mc}_1, \mathrm{mc}_2, \ldots, \mathrm{mc}_{n-1}$ should be merged into the macro-cluster where mc_n is located. Otherwise, micro-clusters $\mathrm{mc}_1, \mathrm{mc}_2, \ldots, \mathrm{mc}_n$ should be merged together to form a new cluster.

According to Definition 5, the assignment rule of boundary micro-clusters is illustrated with an example in Fig. 2(d). As can be seen from the figure, the cluster formed by the blue points is expanded from an existing macro-cluster. The cluster formed by the yellow points is newly generated.

Finally, we give the detailed procedure for assigning the boundary micro-clusters in Algorithm 4. We devise a recursive function to recursively assign the boundary micro-clusters on the same chain (Lines 8–23). The exit of the recursion is the end of a chain (Lines 9–17). A new cluster is generated if the end of a chain is a boundary micro-cluster that has not yet been assigned (Lines 10–14). Otherwise, the category label of the end of the chain (core micro-cluster) is returned directly. Then all the remaining boundary micro-clusters in the chain are assigned to the cluster where the end of the chain is located (Lines 19–21). After assigning all the boundary micro-clusters, the clustering results are obtained (Line 7).

Algorithm 4 BorderAssignment

```
Input:  A set of boundary micro-clusters $V_{border}$,
        a set of intermediate macro-clusters $S_m$;
Output: A set of result clusters
        $S_C = \{C_1, C_2, \ldots, C_K\}$;
1  foreach $mc_i \in V_{border}$ do
2      if $mc_i.visited = True$ then
3          continue;
4      end
5      Call function assign($mc_i$, $S_m$);
6  end
7  return $S_C \leftarrow S_m$.
   // assign recursively
8  function assign($mc$, $S_m$)
9      if $Rep(mc, t) = null$ then              // mc is the end
10         if $mc.cluster = null$ then            // $mc \in V_{border}$
11             $C_t \leftarrow$ create a new empty cluster;
12             $C_t \leftarrow C_t \cup \{mc\}$;
13             $S_m \leftarrow S_m \cup \{C_t\}$;
14         end
15         $mc.visited \leftarrow True$;
16         return;
17     end
18     Call function assign($mc$, $S_m$);
19     if $mc.cluster \neq Rep(mc, t).cluster$ then
20         Add $mc$ to $Rep(mc, t).cluster$;
21     end
22     $mc.visited \leftarrow True$;
23 end
```

F. FBPStream and Complexity Analysis

1) Algorithm Details: The pseudo-code for our FBPStream algorithm is shown in Algorithm 5.

Before processing the data stream $\boldsymbol{X}$, FBPStream creates a thread pool containing two threads (Line 2). The threads in the thread pool will be used later in parallel clustering. First, an attempt is made to absorb x_i into the nearest active micro-cluster mc_i and update the weight of mc_i (Lines 4–8). Then, the weight of each micro-cluster in the *RP-Tree* is decayed in turn, and the micro-clusters are checked to see whether

Algorithm 5 FBPStream

```
Input:  Data stream $X$, micro-cluster's radius $r$,
        decay factor $\lambda$, weight threshold factor $\beta$,
        the number of neighbor micro-clusters $k$,
        level of peeling $\omega$;
Output: A set of result clusters $S_C$;
1  $t \leftarrow 0$;
2  Create a thread pool $\{Thread_1, Thread_2\}$;
3  foreach $x_i \in X$ do
4      Find the micro-cluster $mc_i$ nearest to $x_i$ from RP-Tree;
5      if $d(x_i, mc_i) \leq r$ then
6          Summarize $x_i$ into $mc_i$;
7          Update weight of $mc_i$ by Eq. (4);
8      end
9      foreach $mc \in$ RP-Tree do
10         Decay the weight of $mc$;
11         if $W(mc, t) < \beta/(1-\lambda)$ then
12             Remove $mc$ from RP-Tree;
13             Insert $mc$ into Outlier Pool;
14             Remove $mc$ from Augmented k-NN Graph;
15         end
16     end
17     if $x_i$ is not summarized into $mc_i$ then
18         Find the micro-cluster $mc_j$ nearest to $x_i$ from
19         the Outlier Pool;
20         if $d(x_i, mc_j) \leq r$ then
21             Summarize $x_i$ into $mc_j$;
22             Update weight of $mc_j$ by Eq. (4);
23             if $W(mc_j, t) \geq \beta/(1-\lambda)$ then
24                 Remove $mc_j$ from Outlier Pool;
25                 Insert $mc_j$ into RP-Tree;
26                 Insert $mc_j$ into Augmented k-NN Graph;
27             end
28         end
29     end
30     if $x_i$ is not summarized into $mc_i$ and $mc_j$ then
31         $mc_t \leftarrow$ Create a new micro-cluster by $x_i$;
32         Insert $mc_t$ into Outlier Pool;
33     end
34     if RP-Tree has been updated then
35         $V_{border}, V_{core} \leftarrow$ call function FastPeeling;
36         $S_m, V_{border} \leftarrow$
37           $Thread_1$ calls function CoreClustering,
38           $Thread_2$ calls function BorderLinkage;
39         $S_C \leftarrow$ call function BorderAssignment;
40         Output $S_C$;
41     end
42     $t \leftarrow t + 1$;
43 end
```

they decay to an inactive status (Lines 10 and 11). If yes, the micro-clusters are removed from the *RP-Tree* and inserted into the *Outlier Pool* (Lines 12 and 13). Also, the micro-clusters are removed from the *augmented k-NN graph* (Line 14). If x_i is not absorbed into an active micro-cluster, an attempt is made to absorb it into the nearest inactive micro-cluster mc_j, and the weight of mc_j is updated (Lines 21 and 22). After mc_j has absorbed x_i, we need to check whether mc_j grows into an active micro-cluster (Line 23). If yes, mc_j is removed from the *Outlier Pool* and inserted into the *RP-Tree* (Lines 24 and 25). Also, mc_j is inserted into the augmented k-NN graph (Line 26). If x_i is not absorbed into any existing micro-cluster, then we create a new micro-cluster mc_t with x_i as the seed point and insert it into the *Outlier Pool* (Lines 30–33).

Next, by determining whether an update has taken place in the *RP-Tree*, we determine whether clustering should be performed (Lines 34–41). The possible update scenarios are as follows: 1) a micro-cluster in the *RP-Tree* absorbs a newly

TABLE I
DATASETS USED IN EXPERIMENTS

Data Streams		#Instances	#Features	#Clusters
Synthetic	Stream$_1$	220,000	2	8
	Stream$_2$	110,000	2	2
	Stream$_3$	110,000	2	4
	Stream$_4$	110,000	2	4
	Hyperplane	100,000	10	5
Real-world	NOAAweather	18,159	8	2
	Powersupply	29,928	2	24
	Adult	32,561	6	2
	Electricity	45,312	6	2
	Insects	57,018	33	6
	Rialto	82,250	27	10
	Airlines	539,383	7	2
	Poker	829,201	10	10
	Covertype	581,012	54	7
	Sensor	2,219,803	5	54

arrived data point x_i; 2) a micro-cluster is removed from the *RP-Tree*; and 3) a new micro-cluster is inserted into the *RP-Tree*.

The clustering process starts with a fast peeling of the boundary micro-clusters in the *RP-Tree* to obtain the set of boundary micro-clusters $\boldsymbol{V}_{\text{border}}$ and the set of core micro-clusters $\boldsymbol{V}_{\text{core}}$ (Line 35). Then, the clustering of core micro-clusters and the linkage of boundary micro-clusters are performed in parallel using double threads to obtain the intermediate macro-clusters $\boldsymbol{S_m}$ and the set of boundary micro-clusters $\boldsymbol{V}_{\text{border}}$ with representatives (Lines 36–38). After this, an attempt is made to assign the boundary micro-clusters to the existing macro-clusters to obtain the final clustering result $\boldsymbol{S_C}$ (Line 39). Finally, the clustering result is output (Line 40).

2) Time and Space Complexity Analysis: Assuming that the number of micro-clusters in memory at time t is M_t ($M_t \ll n$, $M_t \in \mathbb{R}^d$), where n is the number of data points in the data stream $\boldsymbol{X}$, d is the dimensionality of the data stream $\boldsymbol{X}$, k is the number of nearest neighbors, and αk is the number of augmented nearest neighbors. In the worst case scenario, FBPStream has a time complexity of $O((d+\omega+\alpha k)nM_t)$ and a space complexity of $O(\alpha kdM_t)$. A detailed description of each step of the complexity analysis is provided in the Supplementary Material.

VI. EXPERIMENTAL RESULTS

A. Preparations

To evaluate the performance of FBPStream, we conduct experiments on 15 different datasets. Table I provides the details about five synthetic datasets and ten real-world datasets. All experiments are conducted on a Lenovo Erazer Y40-70 (Intel i5-4210U, 4 cores, 1 thread/core, 2.40 GHz and 12 GB RAM) with Win10. The FBPStream algorithm and ten comparison algorithms are implemented in Java 8.

1) Datasets: Due to page limitations, descriptions of 5 synthetic datasets and 10 real-world datasets are provided in the Supplementary Material. Data distributions of the four 2-D

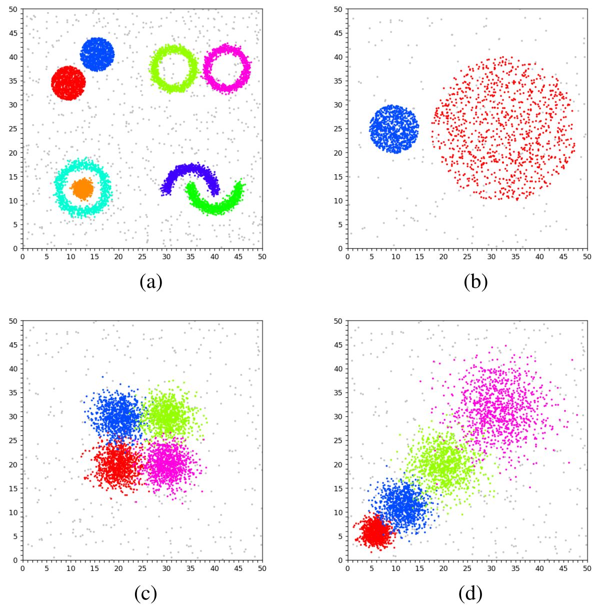


Fig. 4. Data distributions of synthetic datasets. (a) Stream$_1$. (b) Stream$_2$. (c) Stream$_3$. (d) Stream$_4$.

synthetic datasets are illustrated in Fig. 4. In our experiments, the above datasets are normalized.

2) Comparison Algorithms: Based on the above datasets, we compare the performance of FBPStream with ten popular data stream clustering algorithms.[2] ESA-Stream [16], EDMStream [14], and CEDAS [13] are density-based online clustering algorithms. Essentially, EDMStream is an online clustering algorithm using DPC [36], while CEDAS is an online version of DBSCAN. ESA-Stream is optimized for clustering high-dimensional data streams using a grid-based online clustering approach. DenStream [29], HDDStream [30], DBSTREAM [19], DWDP-Stream [17], D-Stream [33], MR-Stream [18], and MuDi-Stream [31] follow the online-offline framework. Among them, DenStream, HDDStream, DBSTREAM, and DWDP-Stream use micro-clusters to summarize data. DenStream is a streaming version of DBSCAN. To cluster high-dimensional data streams, HDDStream uses density-based projection methods. DBSTREAM is an optimized clustering algorithm based on shared density between micro-clusters. DWDP-Stream is an improved DPC algorithm using natural neighbors. D-Stream, MR-Stream, and MuDi-Stream are density- and grid-based clustering algorithms. D-Stream uses fixed-resolution grids, while MR-Stream uses hierarchical multiresolution grids. MuDi-Stream clusters multiresolution data using grids and micro-clusters.

3) Evaluation Metrics: For all algorithms, we use two metrics to measure their performance on each dataset. These two metrics are purity [52] and *cluster mapping measure* (CMM) [53]. We present a more detailed explanation of evaluation metrics in the Supplementary Material.

[2]The proposed years for the baseline models: DWDP-Stream (2022), ESA-Stream (2022), EDMStream (2017), CEDAS (2017), DBSTREAM (2016), MuDi-Stream (2016), HDDStream (2012), MR-Stream (2009), D-Stream (2007), and DenStream (2006).

TABLE II
PERFORMANCE COMPARISON OF ALGORITHMS ON SYNTHETIC AND REAL-WORLD DATASETS

Algorithm	Purity	CMM	Time(s)	Purity	CMM	Time(s)	Purity	CMM	Time(s)	Purity	CMM	Time(s)	Purity	CMM	Time(s)
	Stream$_1$			Stream$_2$			Stream$_3$			Stream$_4$			Hyperplane		
FBPStream	**0.985**	**0.843**	**2.109**	**0.993**	**0.847**	**2.312**	**0.983**	**0.862**	**2.597**	**0.977**	**0.854**	**2.484**	0.671	0.726	**2.766**
EDMStream	0.910	0.751	2.261	0.893	0.839	2.451	0.697	0.789	2.653	0.837	0.706	2.491	0.601	0.697	2.891
CEDAS	0.895	0.803	3.754	0.891	0.815	3.617	0.371	0.486	5.742	0.625	0.553	3.696	0.546	0.702	5.332
DBSTREAM	0.795	0.734	2.659	0.792	0.740	2.662	0.867	0.829	3.964	0.852	0.759	2.882	0.648	0.665	3.210
D-Stream	0.498	0.592	23.914	0.915	0.772	15.238	0.256	0.351	11.704	0.387	0.303	11.872	0.399	0.462	14.992
DenStream	0.811	0.748	4.477	0.882	0.803	4.343	0.421	0.645	5.995	0.782	0.721	5.010	0.542	0.649	6.748
HDDStream	0.708	0.640	13.256	0.831	0.758	10.948	0.339	0.695	14.292	0.741	0.660	14.021	0.553	0.481	13.234
DWDP-Stream	0.786	0.705	5.865	0.977	0.831	6.499	0.305	0.365	6.071	0.504	0.603	4.625	0.476	0.535	4.372
MR-Stream	0.830	0.795	5.223	0.901	0.820	5.030	0.799	0.588	6.802	0.766	0.626	7.976	0.643	0.730	6.363
MuDi-Stream	0.755	0.614	6.547	0.829	0.784	6.228	0.601	0.733	7.688	0.901	0.830	6.290	0.498	0.505	7.796
ESA-Stream	0.949	0.815	4.380	0.986	0.822	3.484	0.934	**0.862**	4.770	0.724	0.683	3.409	**0.738**	**0.759**	7.800
	NOAAweather			Powersupply			Adult			Electricity			Insects		
FBPStream	**0.790**	**0.921**	**2.050**	**0.438**	**0.854**	**1.050**	**0.888**	**0.737**	**1.405**	**0.812**	**0.895**	**2.431**	**0.674**	**0.848**	**2.109**
EDMStream	0.747	0.899	2.317	0.411	0.755	1.129	0.878	0.721	1.590	0.737	0.828	2.976	0.597	0.843	2.220
CEDAS	0.684	0.713	4.323	0.369	0.841	3.863	0.865	0.705	5.688	0.767	0.804	5.434	0.528	0.756	10.406
DBSTREAM	0.721	0.862	2.980	0.402	0.826	1.581	0.887	0.719	1.662	0.725	0.785	2.958	0.633	0.804	3.458
D-Stream	0.582	0.660	45.324	0.299	0.673	6.229	0.841	0.594	4.945	0.526	0.628	6.593	0.409	0.688	245.436
DenStream	0.642	0.753	4.342	0.382	0.739	2.946	0.821	0.645	4.092	0.697	0.738	3.497	0.586	0.746	4.875
HDDStream	0.668	0.829	12.550	0.367	0.702	6.878	0.826	0.587	10.498	0.731	0.802	11.455	0.535	0.728	15.297
DWDP-Stream	0.674	0.686	4.156	0.304	0.652	4.550	0.843	0.688	5.353	0.623	0.728	5.721	0.403	0.654	3.896
MR-Stream	0.701	0.874	4.432	0.421	0.799	3.986	0.874	0.737	6.559	0.784	0.884	5.735	0.629	0.779	6.397
MuDi-Stream	0.663	0.780	6.543	0.377	0.686	4.382	0.801	0.684	6.968	0.683	0.695	6.340	0.407	0.663	7.264
ESA-Stream	0.725	0.852	7.493	0.384	0.710	3.664	0.806	0.637	7.750	0.692	0.801	5.590	0.561	0.785	10.744
	Rialto			Airlines			Poker			Covertype			Sensor		
FBPStream	**0.428**	0.850	**4.986**	0.661	**0.905**	**9.572**	**0.704**	**0.933**	**27.240**	**0.935**	**0.893**	**21.514**	**0.670**	**0.832**	**29.620**
EDMStream	0.272	0.729	5.361	0.626	0.798	10.379	0.654	0.857	28.920	0.766	0.831	22.731	0.447	0.672	30.563
CEDAS	0.408	0.796	11.391	0.642	0.835	24.418	0.673	0.861	40.872	0.894	0.859	33.051	0.511	0.736	36.975
DBSTREAM	0.303	0.697	5.922	0.654	0.867	11.973	0.654	0.895	30.350	0.902	0.885	24.879	0.388	0.587	32.455
D-Stream	0.350	0.658	47.329	0.627	0.705	294.151	0.636	0.748	349.800	0.781	0.760	242.359	0.261	0.470	3561.896
DenStream	0.375	0.754	10.980	0.641	0.818	13.967	0.661	0.799	37.852	0.831	0.868	30.453	0.494	0.696	39.423
HDDStream	0.399	0.768	13.876	0.601	0.754	17.098	0.657	0.735	45.575	0.852	0.823	34.754	0.520	0.696	45.101
DWDP-Stream	0.394	0.759	6.767	0.621	0.773	21.575	0.643	0.721	47.720	0.738	0.704	24.765	0.505	0.713	56.801
MR-Stream	0.401	**0.855**	10.446	0.644	0.847	15.874	0.601	0.687	40.497	0.891	0.799	31.303	0.478	0.696	41.867
MuDi-Stream	0.367	0.755	12.749	0.598	0.680	16.665	0.599	0.660	39.875	0.736	0.731	36.100	0.501	0.683	36.373
ESA-Stream	0.401	0.802	8.459	**0.687**	0.887	18.790	0.683	0.865	47.800	0.885	0.876	38.535	0.450	0.700	39.765

B. Clustering Results

We set up the relevant experiments on synthetic and real-world datasets, respectively. For all algorithms involving exponential decay functions, we carefully set the decay factors to keep them at the same rate. For the same dataset, all algorithms request clustering results at the same frequency (fixed time interval, i.e., gap). Each algorithm is run with a range of parameter values (see Table S1 in the Supplementary Material), and the best clustering results are presented for each dataset (from the optimal parameter value). In the Supplementary Material, Figs. S2 and S3 illustrate the clustering results of all algorithms on 2-D synthetic datasets (i.e., Stream$_1$, Stream$_2$, Stream$_3$, and Stream$_4$). Table II presents the performance of all algorithms on each dataset in Table I, where bolded letters indicate the best results in each metric.

Initially, we test the ability of all algorithms to filter outliers, recognize arbitrarily shaped clusters, and cope with concept drifts on Stream$_1$. For this purpose, we generate eight clusters with arbitrary shapes. These clusters emerge and dissipate successively over time, thereby simulating concept drifts. Fig. S2 demonstrates the clustering results of all algorithms at different gap times ($t1 < t2 < t3 < t4$). According to the results of the experiment, only FBPStream is able to accurately capture the evolution of the dynamic data stream among all the algorithms. Moreover, it proves that FBPStream is robust to outliers and noise.

After that, we test the ability of all algorithms to identify clusters with varying densities and ambiguous boundaries. Thus, we generate two clusters with significantly different densities in the Stream$_2$ dataset. Because the circular cluster on the right is very sparse, each micro-cluster can only absorb a few data points. As a result, only a few micro-clusters can grow into active micro-clusters, which can only be maintained in memory for a short period. The first row of Fig. S3 visually illustrates that FBPStream can effectively identify clusters with varying densities, while nine comparison algorithms (except DWDP-Stream) fail to do so. They divide the low-density cluster on the right side into many sub-clusters. Besides, we generate four clusters of Gaussian distributions with the same standard deviation close to each other in the Stream$_3$ dataset. In Fig. S3, the second row of results shows that FBPStream can handle clusters with ambiguous boundaries as well. It is due to the fact that FBPStream's boundary-peeling strategy can accurately detect the boundaries of clusters and identify their skeletons. DBSTREAM over-segments the clusters. Since DBSTREAM determines the connectivity between micro-clusters by a fixed shared density threshold, it reduces the clustering accuracy when the data distribution is complex. DWDP-Stream, EDMStream, CEDAS, and D-Stream misidentify the four clusters as one. DenStream, HDDStream, MR-Stream, MuDi-Stream, and ESA-Stream merge different clusters incorrectly. Combining the scenarios of Stream_2 and Stream_3, we generate four clusters with varying densities and ambiguous boundaries in the Stream_4 dataset. The third row of Fig. S3 illustrates that FBPStream can still obtain high-quality clustering results when the clusters in the data stream have varying densities and ambiguous boundaries. All comparison algorithms fail to produce correct clustering results. On the Hyperplane dataset, although FBPStream only obtains the second- and third-highest scores in terms of purity and CMM metrics, respectively, it demonstrates notable advancements compared to the two fully online algorithms (i.e., EDMStream and CEDAS). Specifically, it improves purity by 34% and CMM by 7.6%.

Finally, we evaluate the clustering quality of all algorithms on ten real-world datasets. As illustrated in Table II, FBPStream outperforms all comparison algorithms on most real-world datasets. On the NOAAweather dataset, FBPStream's purity improves by an average of 16% over all comparison algorithms, while its CMM improves by an average of 16.5%. Compared to all comparison algorithms, FBPStream improves the purity by an average of 17.8% and CMM by an average of 15.6% on the Powersupply dataset. On the Adult dataset, FBPStream's purity improves by an average of 5.2% over all comparison algorithms. Compared with D-Stream and HDDStream, FBPStream has a significantly better CMM. In comparison with all other algorithms, FBPStream improves purity by an average of 16.2% on the Electricity dataset, and CMM by an average of 16.3%. On the Insects dataset, FBPStream's purity improves by an average of 28.6% over all comparison algorithms, while its CMM improves by an average of 13.8%. Compared to all comparison algorithms, FBPStream improves the purity by an average of 14.2% and CMM by an average of 12.2% on the Rialto dataset. On the Airline dataset, FBPStream obtains the highest CMM while maintaining the next highest purity. On the Poker dataset, FBPStream's purity improves by an average of 8.2% over all comparison algorithms. FBPStream's CMM improves by an average of 19.2% over competitors. Furthermore, its CMM is far superior to MR-Stream and MuDi-Stream. For the high-dimensional dataset Covertype, FBPStream still achieves better clustering results while ensuring clustering efficiency. Specifically, the purity and CMM of FBPStream improve by an average of 10.6% and 9.7% over competitors, respectively. Clustering the Sensor dataset with millions of data, FBPStream takes only 29.620 s. It improves the purity and CMM by an average of 45.9% and 23.4% over all comparison algorithms, respectively. *In summary, the clustering purity and CMM of FBPStream show an average improvement of 16.7% and 15%, respectively, compared to all the comparison algorithms.*

Overall, our results prove that FBPStream is capable of capturing concept drifts, filtering outliers, and detecting clusters with arbitrary shapes. Moreover, FBPStream can effectively cluster data streams with varying densities and ambiguous boundaries. Two factors contribute to FBPStream's ability to reconstruct high-quality clusters from data streams: 1) the decay-based KDE method and 2) the boundary micro-cluster peeling clustering strategy. The decay-based KDE integrates information about the weights and spatial distribution of micro-clusters. It can discover clusters with varying densities and identify the evolving trend of streams well. The boundary micro-cluster peeling clustering strategy first clusters the revealed core micro-clusters to obtain initial clusters with clear boundaries. Then, it assigns the remaining micro-clusters to the initial clusters to avoid erroneous merging of boundary-ambiguous clusters. In contrast, other stream clustering algorithms lack a mechanism to handle both multidensity clusters and boundary-ambiguous clusters, which reduces their performance.

Specifically, the weaknesses of the competitors are analyzed as follows. EDMStream is a variant of DPC designed for data streams, and thus, it inherits the drawbacks of DPC. Like DPC, EDMStream employs a single density threshold to ascertain cluster membership, which poses challenges in detecting clusters with varying densities. Furthermore, the simple data allocation strategy employed by EDMStream may struggle to accurately capture the true relationships within complex-shaped clusters. As DenStream, HDDStream, and CEDAS use fixed connection thresholds, it is difficult to cluster streams that have clusters with varying densities or ambiguous boundaries effectively. Although DBSTREAM establishes connections between micro-clusters based on shared density, it still uses the global density threshold. It means that DBSTREAM cannot cluster streams with varying densities well. In DWDP-Stream, multidensity clusters can be detected, but overlapping clusters cannot be identified. D-Stream uses fixed-resolution grids, which presents a challenge in effectively clustering complex distributed data. MR-Stream, utilizing multiresolution grids, is capable of identifying clusters with varying densities; however, it struggles to effectively cluster streams that contain clusters with ambiguous boundaries. Like MR-Stream, MuDi-Stream cannot identify clusters with ambiguous boundaries. ESA-Stream focuses on the efficient processing of high-dimensional data streams. It does not provide mechanisms to handle clusters with varying densities and ambiguous boundaries.

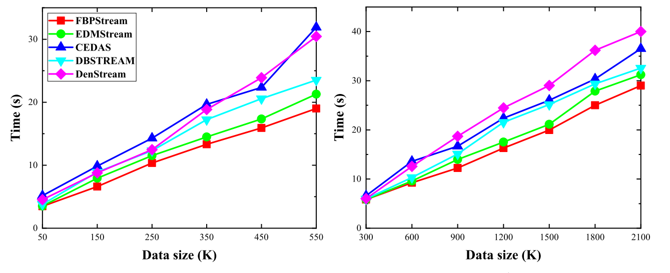
To compare the performance of all algorithms more visually and scientifically, the statistical tests are provided in the Supplementary Material.

C. Ablation Experiments

In this subsection, we perform a series of ablation experiments to validate the effectiveness of two key contributions in our algorithm: the decay-based KDE method and the boundary micro-cluster peeling clustering strategy. The specifics of the ablation experiments can be found in the Supplementary Material.

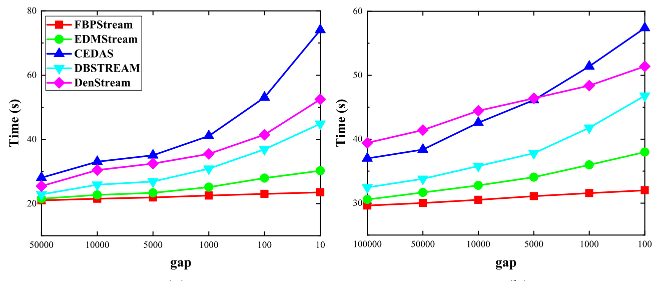
D. Scalability Analysis

In order to verify the scalability of FBPStream, we conduct scalability tests. The experiments utilize two representative datasets: Covertype, which has higher dimensionality, and Sensor, which contains a larger amount of data. In this experiment, we select four micro-cluster-based algorithms,



(a) (b)

Fig. 5. Running time of algorithms for varying data sizes on (a) Covertype and (b) Sensor.



(a) (b)

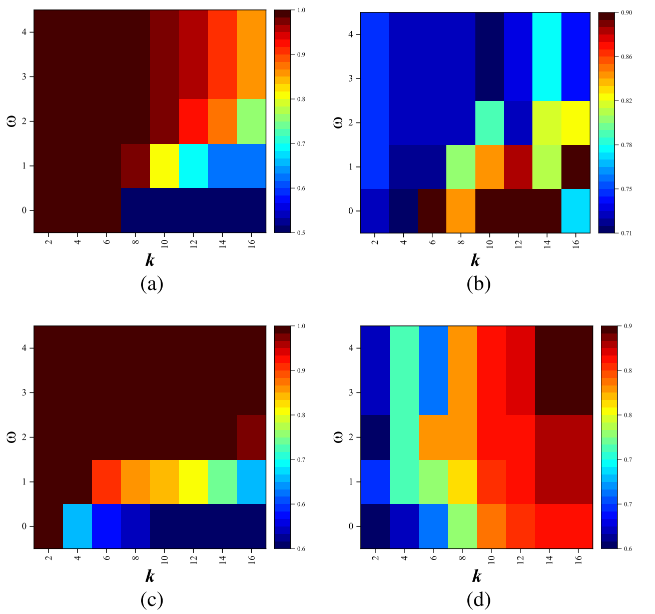
Fig. 6. Running time of algorithms for varying gap values on (a) Covertype and (b) Sensor.

namely EDMStream, CEDAS, DBSTREAM, and DenStream, for comparison with FBPStream.

First, we test the clustering speed of five different algorithms using datasets of varying sizes. We fix gap = 10 000 for all algorithms while varying the data size for clustering. The results are shown in Fig. 5. FBPStream consistently maintains a superior clustering speed as the size of the data increases. In order to maximize the efficiency of clustering, FBPStream employs a parallel clustering mechanism that enables simultaneous boundary micro-cluster assignment and core micro-cluster partitioning. Moreover, it is noteworthy that the *augmented k-NN graph* and the *logical mutual k-NN graph* are highly efficient. Therefore, FBPStream is more scalable for large-scale datasets than the other four algorithms.

The algorithm used for high-speed data streams should be capable of rapidly providing feedback on clustering results. Next, we compare the time taken by different algorithms to request clustering results at different frequencies. Hence, the clustering process is repeated several times using the entire dataset, altering only the time intervals between requests (referred to as “gap”). A smaller gap value indicates shorter time intervals, leading to increased request frequency. The results are shown in Fig. 6.

We can see that the running time of FBPStream grows most smoothly as the gap decreases. FBPStream initiates the clustering process of active micro-clusters by automatically capturing concept drifts within the stream, rather than relying on user requests. When the number of requests keeps increasing, the clustering frequency of FBPStream does not increase. While the clustering frequency of other algorithms continues to increase, clustering times also increase. As a result, FBPStream exhibits enhanced scalability for high-frequency user requests within high-speed data streams.



(a) (b) (c) (d)

Fig. 7. Performance changes of FBPStream with varying parameter values on Stream$_1$ and Stream$_4$. (a) Purity on Stream$_1$. (b) CMM on Stream$_1$. (c) Purity on Stream$_4$. (d) CMM on Stream$_4$.

Finally, we test the effect of using different strategies (single/multithreaded) in FBPStream on the efficiency of the algorithm on real-world datasets. Details are provided in the Supplementary Material.

E. Parameter Analysis

We investigate the influence of two critical parameters of FBPStream on the algorithm’s performance. The critical parameters are the number of nearest neighbors, k, and the peeling level, ω. The experiments are conducted on two representative synthetic datasets. Stream$_1$ is a stream with concept drifts. In Stream$_1$, four graph groups appear sequentially, simulating concept drifts. Cluster densities are uniform and no clusters are overlapping. Stream$_4$ is a stream characterized by clusters that have varying densities and ambiguous boundaries. Fig. 7 shows the values of the purity and CMM metrics obtained by FBPStream on the above two datasets with different parameter values.

First, we analyze how the number of nearest neighbors, k, affects the performance of the proposed algorithm. On the Stream$_1$ dataset and Stream$_4$ dataset, when ω is certain, purity decreases monotonically as k increases. In contrast, the CMM shows a gradual increase. We conjecture that the high purity at the beginning is due to the over-partitioning of the clusters. A small k-value results in a lack of local information, which is the cause of over-partitioning clusters. When the value of k is large, micro-clusters from different clusters have a higher probability of becoming mutual k-nearest neighbors. The purity of the proposed algorithm is reduced due to wrong mk-NN connections. When k falls within the range of [4, 14], FBPStream is generally guaranteed to obtain high purity and CMM on the two datasets.

Then, we observe the effect of peeling level ω on the performance of the proposed algorithm. We fix the value of k.

As the value of ω increases, the performance of the algorithm decreases and levels off on the Stream$_1$ dataset. On the Stream$_4$ dataset, the performance of the algorithm gradually increases and levels off. Stream$_4$ has overlapping clusters, so ω needs to be greater than 0, whereas Stream$_1$ needs to be 0. Evidently, the algorithm's performance exhibits negligible variation when ω exceeds 3. Consequently, setting ω within the range $\{0, 1, 2, 3\}$ is recommended for the proposed algorithm to achieve high purity and CMM.

In practical applications, the recommended range for the number of nearest neighbors, k, is [4, 14]. The recommended range for the peeling level, ω, is $\{0, 1, 2, 3\}$. The more complex and overlapping the data distribution in a stream, the higher the peeling level needs to be set (i.e., the larger the value of ω). The analysis of the remaining parameters of the proposed algorithm is presented in the Supplementary Material.

VII. Conclusion

In this article, we develop an efficient stream clustering algorithm, FBPStream. We define a decay-based KDE, which takes into account the temporal and spatial distribution of each micro-cluster. It can discover clusters with varying densities and identify the evolving trend of streams well. It enhances the algorithm's ability to capture the evolution of data streams with varying densities. Furthermore, FBPStream uses an efficient boundary micro-cluster peeling strategy to reveal the latent clusters' cores, making it easier to identify clusters with ambiguous boundaries. Finally, a parallel clustering strategy is employed to cluster core and boundary micro-clusters effectively. Experiments show that FBPStream is highly competitive in terms of efficiency and quality of results in clustering compared to other algorithms.

Although FBPStream proves to be effective for clustering some real-time data streams, further research is required to adjust the peeling level to more complex data distributions automatically. As a compromise, a performance metric can be monitored during the execution of the algorithm. The peeling level can be adjusted as needed if performance drops are perceived. By adjusting the peeling level dynamically, over-segmentation and under-segmentation of clusters can be avoided. In addition, FBPStream may suffer from low clustering quality when the stream's dimensionality is high. Therefore, clustering data streams with higher dimensionality is also a worth-exploring issue.

References

[1] C. C. Aggarwal, "The multi-set stream clustering problem," in *Proc. SIAM Int. Conf. Data Mining*, Anaheim, CA, USA, Apr. 2012, pp. 59–69.

[2] D. Puschmann, P. Barnaghi, and R. Tafazolli, "Adaptive clustering for dynamic IoT data streams," *IEEE Internet Things J.*, vol. 4, no. 1, pp. 64–74, Feb. 2017.

[3] S. Wang, L. L. Minku, N. Chawla, and X. Yao, "Learning from data streams and class imbalance," *Connection Sci.*, vol. 31, no. 2, pp. 103–104, Apr. 2019.

[4] D. Cheng, S. Zhang, and J. Huang, "Dense members of local cores-based density peaks clustering algorithm," *Knowledge-Based Syst.*, vol. 193, Apr. 2020, Art. no. 105454.

[5] A. Zubaroğlu and V. Atalay, "Data stream clustering: A review," *Artif. Intell. Rev.*, vol. 54, no. 2, pp. 1201–1236, Feb. 2021.

[6] X. Liu, "Incomplete multiple kernel alignment maximization for clustering," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 3, pp. 1412–1424, Jul. 2024.

[7] A. Bifet, S. Maniu, J. Qian, G. Tian, C. He, and W. Fan, "StreamDM: Advanced data mining in spark streaming," in *Proc. IEEE Int. Conf. Data Mining Workshop*, 2015, pp. 1608–1611.

[8] O. Backhoff and E. Ntoutsi, "Scalable online-offline stream clustering in apache spark," in *Proc. IEEE 16th Int. Conf. Data Mining Workshops (ICDMW)*, Dec. 2016, pp. 37–44.

[9] P. G. L. Cândido, J. A. Silva, E. R. Faria, and M. C. Naldi, "Optimization algorithms for scalable stream batch clustering with K estimation," *Appl. Sci.*, vol. 12, no. 13, p. 6464, 2022.

[10] C. C. Aggarwal, S. Y. Philip, J. Han, and J. Wang, "A framework for clustering evolving data streams," in *Proc. 29th Int. Conf. Very Large Data Bases*, Berlin, Germany, 2003, pp. 81–92.

[11] M. Ester et al., "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. 2nd Int. Conf. Knowl. Discovery Data Mining*, Portland, OR, USA, 1996, pp. 226–231.

[12] E. Lughofer and P. Angelov, "Handling drifts and shifts in on-line data streams with evolving fuzzy systems," *Appl. Soft Comput.*, vol. 11, no. 2, pp. 2057–2068, Mar. 2011.

[13] R. Hyde, P. Angelov, and A. R. MacKenzie, "Fully online clustering of evolving data streams into arbitrarily shaped clusters," *Inf. Sci.*, vols. 382–383, pp. 96–114, Mar. 2017.

[14] S. Gong, Y. Zhang, and G. Yu, "Clustering stream data by exploring the evolution of density mountain," *Proc. VLDB Endowment*, vol. 11, no. 4, pp. 393–405, Dec. 2017.

[15] C. Fahy, S. Yang, and M. Gongora, "Ant colony stream clustering: A fast density clustering algorithm for dynamic data streams," *IEEE Trans. Cybern.*, vol. 49, no. 6, pp. 2215–2228, Jun. 2019.

[16] Y. Li, H. Li, Z. Wang, B. Liu, J. Cui, and H. Fei, "ESA-Stream: Efficient self-adaptive online data stream clustering," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 2, pp. 617–630, Feb. 2022.

[17] D. Chen, T. Du, J. Zhou, Y. Wu, and X. Wang, "DWDP-stream: A dynamic weight and density peaks clustering algorithm for data stream," *Int. J. Comput. Intell. Syst.*, vol. 15, no. 1, p. 96, Nov. 2022.

[18] L. Wan, W. K. Ng, X. H. Dang, P. S. Yu, and K. Zhang, "Density-based clustering of data streams at multiple resolutions," *ACM Trans. Knowl. Discovery Data*, vol. 3, no. 3, pp. 1–28, Jul. 2009.

[19] M. Hahsler and M. Bola nos, "Clustering data streams based on shared density between micro-clusters," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 6, pp. 1449–1461, Jun. 2016.

[20] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. D. Carvalho, and J. Gama, "Data stream clustering: A survey," *ACM Comput. Surveys*, vol. 46, no. 1, pp. 1–31, 2013.

[21] A. Amini, T. Y. Wah, and H. Saboohi, "On density-based data streams clustering algorithms: A survey," *J. Comput. Sci. Technol.*, vol. 29, no. 1, pp. 116–141, Jan. 2014.

[22] D. Xu and Y. Tian, "A comprehensive survey of clustering algorithms," *Ann. Data Sci.*, vol. 2, no. 2, pp. 165–193, Jun. 2015.

[23] H.-L. Nguyen, Y.-K. Woon, and W.-K. Ng, "A survey on data stream clustering and classification," *Knowl. Inf. Syst.*, vol. 45, no. 3, pp. 535–569, Dec. 2015.

[24] M. Carnein and H. Trautmann, "EvoStream–evolutionary stream clustering utilizing idle times," *Big Data Res.*, vol. 14, pp. 101–111, Dec. 2018.

[25] P. L. Cândido, M. C. Naldi, J. A. Silva, and E. R. Faria, "Scalable data stream clustering with k estimation," in *Proc. Brazilian Conf. Intell. Syst.*, Uberlândia, Brazil, 2017, pp. 336–341.

[26] M. Hassani, P. Spaus, and T. Seidl, "Adaptive multiple-resolution stream clustering," in *Proc. 10th Int. Conf. Mach. Learn. Data Mining Pattern Recognit.*, St. Petersburg, Russia, 2014, pp. 134–148.

[27] M. Hassani, P. Spaus, A. Cuzzocrea, and T. Seidl, "Adaptive stream clustering using incremental graph maintenance," in *Proc. 4th Int. Workshop Big Data, Streams Heterogeneous Source Mining, Algorithms, Syst., Program. Models Appl.*, Sydney, NSW, Australia, 2015, pp. 49–64.

[28] J. Shao, Y. Tan, L. Gao, Q. Yang, C. Plant, and I. Assent, "Synchronization-based clustering on evolving data stream," *Inf. Sci.*, vol. 501, pp. 573–587, Oct. 2019.

[29] F. Cao, M. Estert, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *Proc. SIAM Int. Conf. Data Mining*, Bethesda, MD, USA, 2006, pp. 328–339.

[30] I. Ntoutsi, A. Zimek, T. Palpanas, P. Kröger, and H.-P. Kriegel, "Density-based projected clustering over high dimensional data streams," in *Proc. SIAM Int. Conf. Data Mining*, Anaheim, CA, USA, 2012, pp. 987–998.

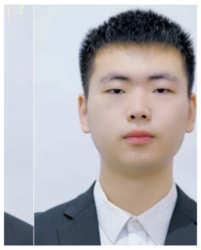
[31] A. Amini, H. Saboohi, T. Herawan, and T. Y. Wah, "MuDi-stream: A multi density clustering algorithm for evolving data stream," *J. Netw. Comput. Appl.*, vol. 59, pp. 370–385, Jan. 2016.
[32] M. Mousavi, H. Khotanlou, A. A. Bakar, and M. Vakilian, "Varying density method for data stream clustering," *Appl. Soft Comput.*, vol. 97, Dec. 2020, Art. no. 106797.
[33] Y. Chen and L. Tu, "Density-based clustering for real-time stream data," in *Proc. 13th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, San Jose, CA, USA, Aug. 2007, pp. 133–142.
[34] L. Tu and Y. Chen, "Stream data clustering based on grid density and attraction," *ACM Trans. Knowl. Discovery Data*, vol. 3, no. 3, pp. 1–27, Jul. 2009.
[35] Q. Qian, C.-J. Xiao, and R. Zhang, "Grid-based data stream clustering for intrusion detection," *Int. J. Netw. Secur.*, vol. 15, no. 1, pp. 1–8, 2013.
[36] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *Science*, vol. 344, no. 6191, pp. 1492–1496, Jun. 2014.
[37] H. Averbuch-Elor, N. Bar, and D. Cohen-Or, "Border-peeling clustering," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 7, pp. 1791–1797, Jul. 2020.
[38] M. Du, R. Wang, R. Ji, X. Wang, and Y. Dong, "ROBP a robust border-peeling clustering using Cauchy kernel," *Inf. Sci.*, vol. 571, pp. 375–400, Sep. 2021.
[39] J. Tobin and M. Zhang, "DCF: An efficient and robust density-based clustering method," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Auckland, New Zealand, Dec. 2021, pp. 629–638.
[40] R. Li, X. Yang, X. Qin, and W. Zhu, "Local gap density for clustering high-dimensional data with varying densities," *Knowledge-Based Syst.*, vol. 184, Nov. 2019, Art. no. 104905.
[41] C. N. Mavridis and J. S. Baras, "Online deterministic annealing for classification and clustering," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 10, pp. 7125–7134, Oct. 2023.
[42] Y. Liu, X. Fan, W. Li, and Y. Gao, "Online passive-aggressive active learning for trapezoidal data streams," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 10, pp. 6725–6739, Aug. 2023.
[43] I. Škrjanc, G. Andonovski, J. A. Iglesias, M. P. Sesmero, and A. Sanchis, "Evolving Gaussian on-line clustering in social network analysis," *Exp. Syst. Appl.*, vol. 207, Nov. 2022, Art. no. 117881.
[44] L. L. Minku and X. Yao, "DDD: A new ensemble approach for dealing with concept drift," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 4, pp. 619–633, Apr. 2012.
[45] H. Yu, W. Liu, J. Lu, Y. Wen, X. Luo, and G. Zhang, "Detecting group concept drift from multiple data streams," *Pattern Recognit.*, vol. 134, Feb. 2023, Art. no. 109113.
[46] F. Fedeli, A. M. Metelli, F. Trovò, and M. Restelli, "IWDA: Importance weighting for drift adaptation in streaming supervised learning problems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 10, pp. 6813–6823, Jul. 2023.
[47] G. Andonovski and I. Škrjanc, "Evolving clustering algorithm based on average cluster distance–eCAD," in *Proc. IEEE Int. Conf. Evolving Adapt. Intell. Syst. (EAIS)*, May 2022, pp. 1–5.
[48] I. Škrjanc, "Cluster-volume-based merging approach for incrementally evolving fuzzy Gaussian clustering—eGAUSS+," *IEEE Trans. Fuzzy Syst.*, vol. 28, no. 9, pp. 2222–2231, Sep. 2020.
[49] I. Škrjanc, J. A. Iglesias, A. Sanchis, D. Leite, E. Lughofer, and F. Gomide, "Evolving fuzzy and neuro-fuzzy approaches in clustering, regression, identification, and classification: A survey," *Inf. Sci.*, vol. 490, pp. 344–368, Jul. 2019.
[50] L. Breiman, W. Meisel, and E. Purcell, "Variable kernel estimates of multivariate densities," *Technometrics*, vol. 19, no. 2, p. 135, May 1977.
[51] D. Cheng, Q. Zhu, J. Huang, Q. Wu, and L. Yang, "Clustering with local density peaks-based minimum spanning tree," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 2, pp. 374–387, Feb. 2021.
[52] Y. Zhao and G. Karypis, "Empirical and theoretical comparisons of selected criterion functions for document clustering," *Mach. Learn.*, vol. 55, no. 3, pp. 311–331, Jun. 2004.
[53] H. Kremer et al., "An effective evaluation measure for clustering on evolving data streams," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2011, pp. 868–876.

Jiarui Sun received the B.S. degree from Jiangsu Normal University, Xuzhou, China, in 2019, where he is currently pursuing the master's degree with the School of Computer Science and Technology.

His research interests include big data analysis and data stream clustering.

Mingjing Du (Member, IEEE) received the Ph.D. degree in computer science from China University of Mining and Technology, Xuzhou, China, in 2018.

He is currently an Associate Professor with the School of Computer Science and Technology, Jiangsu Normal University, Xuzhou. His research interests include cluster analysis and three-way decisions. For more information, see https://dumingjing.github.io/.



Chen Sun is currently pursuing the bachelor's degree with the School of Computer Science and Technology, Jiangsu Normal University, Xuzhou, China.

His research interests include three-way clustering and artificial intelligence in education.

Yongquan Dong received the Ph.D. degree in computer science from Shandong University, Jinan, China, in 2010.

He is currently a Professor with the School of Computer Science and Technology, Jiangsu Normal University, Xuzhou, China. His research interests include web information integration and web data management.