Knowledge-Based Systems 120 (2017) 99–117

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Fat node leading tree for data stream clustering with density peaks

Ji Xu[a,c,d], Guoyin Wang[b,*], Tianrui Li[a], Weihui Deng[c], Guanglei Gou[a]

[a] School of Information Science & Technology, Southwest Jiaotong University, Chengdu 610031, China
[b] Chongqing Key Laboratory of Computational Intelligence, Chongqing University of Posts and Telecommunications, Chongqing 400065, China
[c] Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences,Chongqing 400714, China
[d] School of Information Engineering, Guizhou University of Engineering Science, Bijie 551700, China

ARTICLE INFO

Article history:
Received 23 June 2016
Revised 20 December 2016
Accepted 28 December 2016
Available online 29 December 2016

Keywords:
Data stream clustering
Density peaks
Fat node leading tree
Change point

ABSTRACT

Detecting clusters of arbitrary shape and constantly delivering the results for newly arrived items are two critical challenges in the study of data stream clustering. However, the existing clustering methods could not deal with these two problems simultaneously. In this paper, we employ the density peaks based clustering (DPClust) algorithm to construct a leading tree (LT) and further transform it into a fat node leading tree (FNLT) in a granular computing way. FNLT is a novel interpretable synopsis of the current state of data stream for clustering. New incoming data is blended into the evolving FNLT structure quickly, and thus the clustering result of the incoming data can be delivered on the fly. During the interval between the delivery of the clustering results and the arrival of new data, the FNLT with blended data is granulated as a new FNLT with a constant number of fat nodes. The FNLT of the current data stream is maintained in a real-time fashion by the Blending-Granulating-Fading mechanism. At the same time, the change points are detected using the partial order relation between each pair of the cluster centers and the martingale theory. Compared to several state-of-the-art clustering methods, the presented model shows promising accuracy and efficiency.

1. Introduction

Data streams have been generated everywhere nowadays because of the technological development in sensors, networks, smart phones and surveillance. Mining data streams in specific domains such as environmental monitoring, city traffic load monitoring [1], or online commercial activities [2], etc., has produced a lot of researches. Clustering data stream has become one of the important issues since a majority of the data streams come unlabeled in the age of Big Data, and turned to be critical in summarizing data or finding out outliers [3].

The major challenges in clustering data streams include: 1) Data streams continuously flow in, so it is usually unfeasible to store all the original data on disk. Therefore, it demands that the data be processed in one single pass. 2) The patterns may change occasionally or frequently as data points in the streams keep arriving [4]. To address these challenges, quite a few research works have been published, e.g. [5–12]. We will discuss these works in the coming section.

Current data stream clustering approaches fall into two categories: *K*-means-like and density- based. The former intends to minimize the distance summation of non-center data points to their corresponding centers, hence the incapability of detecting non-spherical clusters. The methods of latter category cluster items based on their density distribution in the space where the items are embedded, so they can detect right clusters in arbitrary shapes of datasets. However, some density-based data stream clustering methods (like D-Stream [8] and MR-Stream [7]) that find clusters with the concept of *dense grids* (determined with a preset threshold value), may fail to perform well when there coexist clusters of different density levels. Recently, Hahsler and Bolanos addressed this problem by proposing a micro-cluster-based data stream clustering method that leverages the density between micro-clusters through a shared density graph (this method is named as DBSTREAM) [11].

We present in this paper a novel data stream clustering (named as DP-Stream) with the underlying *leading tree* (LT, refer to Section 3.2 for a detailed explanation) structure in the density-peaks-based clustering method [13]. The initial buffered data points are firstly used to construct an LT. The LT can be used

* Corresponding author.
E-mail addresses: alanxuch@hotmail.com, AlanxuCh@hotmail.com (J. Xu), wanggy@ieee.org (G. Wang), trli@swjtu.edu.cn (T. Li), dengweihui@cigit.ac.cn (W. Deng).

http://dx.doi.org/10.1016/j.knosys.2016.12.025
0950-7051/

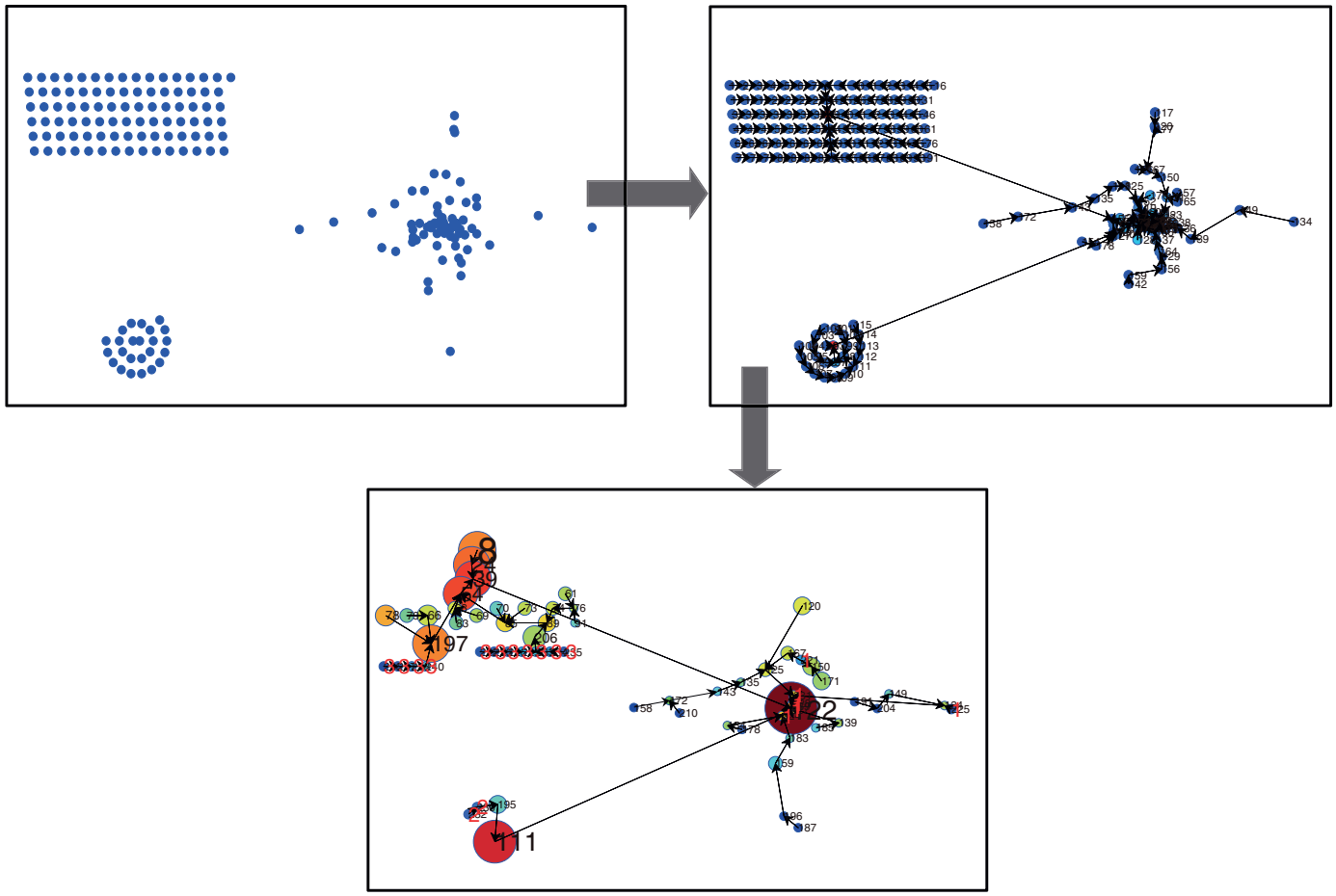


Fig. 1. A simplified illustration of DP-Stream. From top to bottom: buffered data before initialization → initial LT → Granulating LT into FNLT and incorporating the new items with cluster label immediately delivered. The color represents the local density and the radius represents the population (weight) of each node.

to deliver the clustering result for the initial buffered data given the cluster centers are selected automatically. Then the LT is granulated into a *fat node leading tree* (FNLT, refer to Section 4.1 for a detailed definition) by merging the closest points to their corresponding parents, so as to capture the essence of the finest-grained data items with a synopsis of data [14]. Heinz and Seeger proposed to use a Cluster Kernel to present a group of objects in the data stream [15]; they also addressed the issue of limiting the memory consumption in clustering streaming data, in which the cluster kernels may be regarded as information granules as the fat nodes in our DP-Stream. But the difference is obvious, since the fat nodes in our method are some closely located data points other than the resultant clusters. As the data items streaming in, their clustering assignation is quickly determined as soon as the local density of every node (including new items and the existing fat nodes) is incrementally updated. An example of FNLT is shown in Fig. 1.

At the last stage of a clustering-new-items round, the previous FNLT along with the incorporated new items is granulated again (to keep the population of the nodes stable), waiting the next batch of coming items in the stream. Like most of the data stream clustering, DP-Stream includes a fading out mechanism to focus on recent data points and a change point detection utility to deal with concept drift. However, the difference is that our method has very simple implementation of these two utilities due to the properties of an FNLT structure.

DP-Stream has the following salient features:

- It can detect clusters of arbitrary shapes and different density levels;
- The concept drift is accurately and efficiently detected in a simple way;
- It offers an intuitively interpretable visualization of the evolving synopsis of the data stream;
- The evolution of the FNLT is implemented with an efficient incremental update, thus DP-Stream permanently offers clustering result for streaming in items.

Most of the existing data stream clustering methods, such as DBSTREAM, MR-Stream, CluStream, fall in the online-offline category. However, those online-offline models do not adapt well to some applications (e.g. system monitoring), in which the clusters information is required to be always ready.

To the best of our knowledge, DP-Stream is the first model using a density-peaks-based LT structure to cluster data stream. And more importantly, it is the first data stream clustering method that simultaneously meets the two demands: detecting clusters of any shape and running without an offline component.

The remainder of this paper is organized as follows. After a brief discussion of the related works in Section 2, we present in Section 3 the automatic selection of centers and the leading tree structure with density-peaks-based clustering. In Section 4, we describe the DP-Stream method including the components of outliers' recognition, drift detection, and fading function. In Section 5, we discuss the computational complexity of maintaining the FNLT. Section 6 describes detailed experiments with synthetic and real datasets. A conclusion is given in Section 7.

2. Background

In this section, we review some related works on data stream clustering and the clustering method based on density peaks.

2.1. Clustering data streams

Generally, a clustering method for data stream is originated from a corresponding clustering method for batch data, e.g., CluStream [16] from *K*-means, DenStream [6] from DBSCAN [17], and STRAP [9] from AP clustering [18]. From the perspective of whether they can detect arbitrary shape in the data stream, the data stream clustering methods can be divided into two categories: *K-means-like* and *density-based*.

2.1.1. K-means-like

Aggarwal et al. developed CluStream for clustering large evolving data streams [16]. It divides the clustering process into an "online component" and an "offline component". The former periodically summarizes the statistics of raw data and the latter uses this summary statistics to perform clustering. Many later researches on data stream clustering follow this "offline component" paradigm. Subsequently, Aggarwal et al. expanded CluStream to HPStream to clustering high dimensional data streams [3].

Ackermann et al. proposed the StreamKM++ method by maintaining a small sketch of the input using the merge-and-reduce technique [19]. StreamKM++ produces good clustering results for ellipsoidal data sets but is not efficient enough.

Zhang et al. presented a data stream clustering method called STRAP [9], which extends the clustering algorithm of affinity propagation (AP) [18] to evolving data streams. In STRAP, weighted AP is firstly extended from AP to updating the model when a new item arrives and then Page-Hinkley test is used to detect change point of a drift. The users of STRAP need to specify the parameter of ϵ, which is the threshold for deciding whether an item should go to reservoir as an outlier or join in the existing model.

Lughofer and Sayed–Mouchaweh extended *evolving vector quantization* (eVQ) [20] to eVQ-A to propose a single-pass and sample-wise streaming data clustering method, which delivers the clustering result permanently without a retraining procedure [10]. It can detect clusters with convex (ellipsoidal) shapes in any directions and locations.

Although *k*-means-like data stream clustering methods are not able to detect nonspherical clusters, these methods may produce lower within cluster sum of squares (WSS) than those density-based methods when the dataset is actually spherically shaped, because *k*-means directly tries to minimize WSS. For example, CluStream produced twice a slightly lower WSS than DBSTREAM [11].

2.1.2. Density-based

Cao et al. proposed an approach for discovering clusters in an evolving data stream named DenStream [6]. Its online component constructs and maintains three sorts of micro-clusters, namely core micro-clusters, potential micro-clusters and outlier micro-clusters, to summarize the data streams. And a pruning strategy is used to remove some outlier micro-clusters for economical memory consumption. Its offline component employs a variant of DBSCAN on the set of potential micro-clusters to meet a clustering request. However, It has problems with detecting the rectangular low-density cluster.[1]

D-Stream [8] is a grid-density-based stream clustering algorithm organizing the grids in a red-black tree. It uses the concept of connecting neighboring grids (which is defined as two grids that differ only on one dimension and on this dimension their intervals are adjacent) to summarize the close data points. The relation of neighboring is used to transitively connect the grids to form a grid group. A set of grids $G = (g_1, \ldots, g_m)$ is a grid cluster if it is a grid group, every inside grid of G is a dense grid, and every outside grid of G is either a dense grid or a transitional grid. It will be difficult for D-Stream to find the density threshold to detect dense grids when there are several clusters of different density level in the data stream.

MR-Stream [7] provides a multiresolution cluster discovery capability for stream data, based on the idea of first putting records into treelike organized cells of multi-granularity and then clustering the cells considering their weights, volumes and densities. MR-Stream can detect the clusters of arbitrary shape with noise. Apart from those for fading function, MR-Stream involves quite a few additional parameters, namely ratio of distance threshold divide unit length of interval ϵ, minimum weight of a cluster β, and minimum size of a cluster μ. Like D-Stream, MR-Stream also has the difficulty to determine the density threshold to identify dense grids when the density levels of clusters are different.

Recently, Hahsler proposed DBSTREAM [11], whose micro-cluster-based online clustering component explicitly captures the density between micro-clusters via a shared density graph, addressed the problem with D-Stream and MR-Stream when the data points within each cell are not uniformly distributed and two dense cells are separated by a small area of low density.

All the density based data stream clustering methods above follow the "online-offline" style. By "online-offline", it means that the stream clusterings consist of two major components. The "online" part is responsible for maintaining the microclusters of the streaming objects, while the "offline" performs the resultant clustering based on the microclusters only when a user submits his/her query.

However, there are some researches combined the ideas of the mentioned "*k*-means-like" and "density-based", hence cannot be categorized as any one type of them. For example, Rehman et al. presented a successful hybrid of "loading objects into grids" of D-Stream (or MR-Stream) and "merging closest micro-clusters" of CluStream [21]. Besides, there are some hierarchical clustering models for data streams that do not need to specify a number of clusters and can return a more informative cluster hierarchical structure [22,23].

2.1.3. Change point detection

When clustering evolving data streams, the major cluster may change over time. We will get the clustering result wrongly evaluated if we do not tag the change point, because the cluster labeled as C_1 is actually another cluster rather than the former cluster C_1 once the drift occurs.

Zhang et al. used the Page-Hinkley test to detect the change points in STRAP [9]. Koshijima et al. proposed a nonparametric and computationally efficient change point detection method named bag-of-words [24]. Ho and Wechsler developed a general martingale framework for detecting changes in time-varying data streams, in which the exchangeability of the data is tested to determine whether a change has occurred [12]. This martingale approach is of special interest in our research, since we will show later the FNLT structure can be conveniently put in the martingale framework to detect a major concept drift.

2.2. Clustering with density peaks (DPClust)

The proposed method starts from DPClust [13] by Rodriguez and Laio, so we give a brief introduction to its idea and the algorithm here. They firstly made a sound intuitive assumption that no matter what the shape of clusters looks like, centers are always

[1] https://cran.r-project.org/web/packages/streamMOA/vignettes/streamMOA.pdf.

Table 1
Notations in DPClust and DP-Stream.

Symbol	Meaning
$X = (x_1, \ldots, x_N)$	The dataset with x_i as its i th data point
$D = \{d_{i,j}\}$	The distances of the pairs of data points in X, where $1 \le i < j \le N$
$\mathbf{I} = (1, \ldots, N)$	The set of the indices of data points in the dataset
$\boldsymbol{C} = (C_1, \ldots, C_K)$	K centers of the clustering result
$\boldsymbol{\rho} = (\rho_1, \ldots \rho_N)$	The local density of X
$\boldsymbol{\delta} = (\delta_1, \ldots, \delta_N)$	The distance to nearest points of higher local density
$\boldsymbol{Q} = (q_1, \ldots, q_N)$	The sorted indices of $\boldsymbol{\rho}$ in descending order i.e., $\rho_{q_1} \ge \rho_{q_2} \ge \ldots \ge \rho_{q_N}$
$\boldsymbol{Nn} = (Nn_1, \ldots, Nn_N)$	The indices of the nearest neighbor with larger ρ for data points series$(x_1, \ldots, x_N)$
$\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_N)$	The elementwise product of $\boldsymbol{\rho}$ and $\boldsymbol{\delta}$
$\boldsymbol{W} = (w_1, \ldots, w_N)$	The weight (population) of nodes in FNLT
$\boldsymbol{Cl} = (Cl_1, \ldots, Cl_N)$	The cluster label of nodes in FNLT

surrounded by non-center data points with lower density, and the distance between two centers are relatively long. Then two simple measures, namely local density (denoted as ρ) and minimal distance to data points with higher density (denoted as δ), are employed to accomplish the clustering job.

The notations used to describe the algorithm DPClust and our methods are listed in Table 1.

The algorithm of DPClust takes the distance matrix of a given dataset as input, and performing the following steps:

1) Compute $\{\rho_1, \rho_2, \ldots, \rho_N\}$ via cut-off kernel:

$$\rho_i = \sum_{j \in I \setminus \{i\}} \chi(d_{i,j} - d_c), \ where \ \chi(x) = \begin{cases} 1, & x < 0; \\ 0, & x \ge 0. \end{cases} \tag{1}$$

d_c is the cutoff distance; or via Gaussian kernel:

$$\rho_i = \sum_{j \in I \setminus \{i\}} e^{-\left(\frac{d_{i,j}}{d_c}\right)^2}. \tag{2}$$

Eq. (2) is used in the implementation of Rodriguez and Laio;

2) Sort $\{\rho_1, \rho_2, \ldots, \rho_N\}$ in a descending order to yield $(\rho_{q_1}, \rho_{q_2}, \ldots, \rho_{q_N})$;

3) Compute $\boldsymbol{\delta}$ via

$$\delta_{q_i} = \begin{cases} \min\limits_{j<i}\{d_{q_i,q_j}\}, & i \ge 2; \\ \max\limits_{j \ge 2}\{d_{q_i,q_j}\}, & i = 1. \end{cases} \tag{3}$$

and write the index of the nearest neighbor with larger ρ in vector $\boldsymbol{Nn}$:

$$Nn_i = \begin{cases} 0, \ if \ i = q_1 \\ j \ such \ that \ \delta_i = d_{i,j}, \ otherwise \end{cases} \tag{4}$$

4) Interactively choose the points with "anomalously large" ρ and δ as centers;

5) Assign each data point to the same cluster as its nearest neighbor with larger ρ. At first, the centers are assigned to their corresponding cluster label, then each noncenter object x_i is assigned to the same cluster as Nn_i. Formally, this is written as:

$$Cl_i = \begin{cases} k, & if \ i = C_k, k \in \{1, \ldots, K\} \\ Cl_{Nn_i}, & otherwise \end{cases} \tag{5}$$

DPClust uses a parameter named *bord_rho* to distinguish *core* and *hallo* data points of a cluster. The hallo data points around a cluster are somehow similar to but are not outliers in fact. And whether a point is a hallo point depends heavily on the choice of d_c parameter. So for simplicity, we omit the discussion of hallo and core in this study.

3. Extending DPClust

In original implementation of DPClust offered by the authors, the centers are chosen by the users interactively. This is good for running the program once per dataset, and it can involve human's insight to follow the statement – *the only points of high δ and relatively high ρ are the cluster centers* [13]. But when DPClust has to be called iteratively to cluster a data stream, the centers need to be automatically selected. We address this issue in Section 3.1. Although it is not explicitly pointed out in [13], DPClust constructs a tree structure (we define it as *Leading Tree* later) as its intermediate result. This tree structure is the key for us to design a data stream clustering algorithm which constantly delivers the clustering result online without an offline component.

3.1. Automatic selection of centers

There have existed some works on automatically selecting the cluster centers, e.g., Hinneburg and Keim used a hill climbing approach to identify the local maxima of the density function [25] (the corresponding clustering method is called DENCLUE), which are corresponding to the cluster centers in DPClust. Hinneburg and Garbriel further made a progress toward a faster DENCLUE (called DENCLUE 2.0) by reducing the hill climbing to a special case of an expectation maximization problem [26]. We use a linear fitting approach to automatically select the centers [27]. As pointed out in [13], the cluster centers are featured by their anomalously large γ values.

Let $[\boldsymbol{\gamma^s}, \boldsymbol{\gamma Ind}] = sortDescending(\boldsymbol{\gamma})$, For $i = N - l$ to 0 (from the end to beginning) we fit the γ points against their indices to a linear equation with length l:

$$\boldsymbol{\gamma}_i^s = a_i \boldsymbol{I}_i + b_i, \tag{6}$$

where $\boldsymbol{\gamma}_i = (\gamma_{i+1}, \gamma_{i+2}, \ldots, \gamma_{i+l})$, $\boldsymbol{I}_i = (I_{i+1}, I_{i+2}, \ldots, I_{i+l})$. The two variables a_i and b_i are solved to reach a minimal mean square error (MSE).

Then compute $\hat{\gamma}_i = a_i I_i + b_i$ and $\Delta\gamma = \gamma_i - \hat{\gamma}_i$, when the first i is found to satisfy

$$\Delta\gamma_i > LocalR * \mathrm{Max}(\boldsymbol{d\gamma}_i^s), \tag{7}$$

and

$$\Delta\gamma_i > GlobalR * \mathrm{Max}(\boldsymbol{d\gamma}^s), \tag{8}$$

the procedure of detecting centers terminates, where $\boldsymbol{d\gamma}_i^s$ and $\boldsymbol{d\gamma}^s$ are the difference-vector of $\boldsymbol{\gamma}_i^s$ and $\boldsymbol{\gamma}^s$ respectively. *LocalR* is designed to control how significant the γ value of the first center should grow larger than the previous data points (considering the direction of linear fitting movement), and *GlobalR* is used to control how large $\Delta\gamma$ should be with respect to the global greatest γ value. *GlobalR* is meant to avoid a situation, where the $\Delta\gamma_i$ is very large relative to $\boldsymbol{d\gamma}_i^s$ but very trivial to the maximum increment in γ. In this case, the point should not make a center.

When the linear fitting process terminates, the number of centers is i and the points with indices in $(\gamma Ind_1, \ldots, \gamma Ind_i)$ are the corresponding centers. This method is illustrated as Fig. 2.

3.2. Leading Tree structure in DPClust

By careful investigation, we find out that the intermediate result $\boldsymbol{Nn}$ in DPClust actually represents a tree. In this tree, each node except the root is led by its parent to join the same cluster. Thus we call this tree a *Leading Tree* (LT), and name the parent of an item as its *leading node*. With an LT, the process of assigning the noncenter data points turns into disconnecting the centers from their parents (except the root of the LT). And the resulted subtrees represent the clusters. Example 1 illustrates the idea.

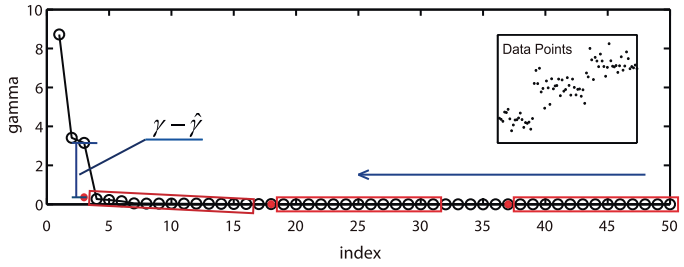


Fig. 2. Diagram to illustrate the idea of selecting centers with linear fitting. The black circles are the real γ values, and a red dot is the predicted γ value of the data point before the points currently being linearly fitted. If there is a significant "jump" from the predicted $\hat{\gamma}$ value to the real γ value at position i, then the points whose γ values are larger or equal to γ_i are chosen as cluster centers. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Example 1. We firstly generate 13 points (DS1) on the 2D plane, as shown in the upper left corner of Fig. 3, and then the intermediate results $\boldsymbol{\rho}$, $\boldsymbol{\delta}$, $\boldsymbol{\gamma}$, ***Nn*** and the final result ***Cl*** for DS1 are computed with DPClust (according to the definitions in Table 1 and the Equations in Section 2.2. That is, $\boldsymbol{\rho}$ is computed with Eq. (2), $\boldsymbol{\delta}$ with Eq. (3), $\gamma_i = \rho_i \times \delta_i$ for each i from 1–N, ***Nn*** is computed with Eq. (4), and ***Cl*** is computed with Eq. (5)), as shown in the bottom right corner of Fig. 3. Finally, the tree structure is indicated by vector ***Nn*** and the 3 points with largest γ value are chosen to be centers by the linear fitting approach. The LT of Example 1 is shown in the salient position of Fig. 3, and it can be split into 3 subtrees if the points x_6 and x_{11} are disconnected from their parents. Each subtree is a cluster, which is corresponding to the CL row in the table contained in Fig. 3.

The logical relationship among the centers of clusters in DP-Clust is not of "peer to peer" as in many other clustering methods (e.g. *K*-means, AP clustering), but of partial ordinal relation. To prepare the discussion of change detection in later sections, we give two definitions here.

Definition 1. η **operator**. For any non-root node x in an LT, there is a node p such that $\eta(x) = p$, where p is the nearest neighbor with higher local density to x. More formally, $\eta(x) = p$ iff $p = \arg\min_y \{d_{x,y} | \rho_x < \rho_y; x, y \in X\}$.

we denote $\underbrace{\eta(\eta(\ldots\eta(\bullet)))}_{n\ times} = \eta^n(\bullet)$, e.g., in Fig. 3, $\eta(x_6) = x_{13}$, $\eta^2(x_9) = x_{12}$.

Definition 2 partial order in LT. Suppose x_i, $x_j \in X$, we say $x_i \prec x_j$, $iff\ \exists m \in N^+$ such that $x_j = \eta^m(x_i)$.

Obviously, the root C_1 satisfies $C_i \prec C_1$, $\forall C_i \in C \backslash C_1$, where C_1 is always the center of a cluster no matter how many clusters there would be. With this starting point, we can define the *significant drift* in DP-Stream, which will be elaborated on in Section 4.4.

4. DP-Stream algorithm

This section aims at using FNLT (see Section 4.3 for a detailed definition) to cluster data streams, more specifically, achieving online clustering in the case of non-stationary data distributions. The resulting algorithm, called DP-Stream, involves the following steps (Algorithm 1 with a diagram in Fig. 4).

1) The initial buffered data is used to construct an LT after computing the vectors $\boldsymbol{\rho}$, $\boldsymbol{\delta}$, ***Nn***. The cluster centers are automatically selected with a linear fitting approach for parameter vector $\boldsymbol{\gamma}$. The cluster label for each data point is stored as ***Cl***.
2) Granulate the LT into an FNLT;
3) The evolving FNLT is updated in a batch fashion:
3a) With a new arrived item, the first step is to incrementally update $\boldsymbol{\rho}$ and $\boldsymbol{\delta}$ for x and all nodes in the FNLT. Compute the strangeness parameter θ to decide whether it is an outlier (or possibly a starting point of a drift).
3b) Find the leading node for the new item X_{new}, and assign the same cluster label as its leading nodes if X_{new} is not *strange*. If X_{new} is strange, then its clustering result will be postponed to the moment when the buffer is filled up.
3c) Decide whether a drift has occurred. If there is a drift, then tag the change point (for further use in the evaluation or validation); if the strange points are outliers, then store them on the hard disk or simply discard them.
3d) Fade the history nodes. The *weak nodes* are removed.
3e) The FNLT with newly incorporated items are further granulated into a new FNLT.

Algorithm 1: DP-Stream algorithm.

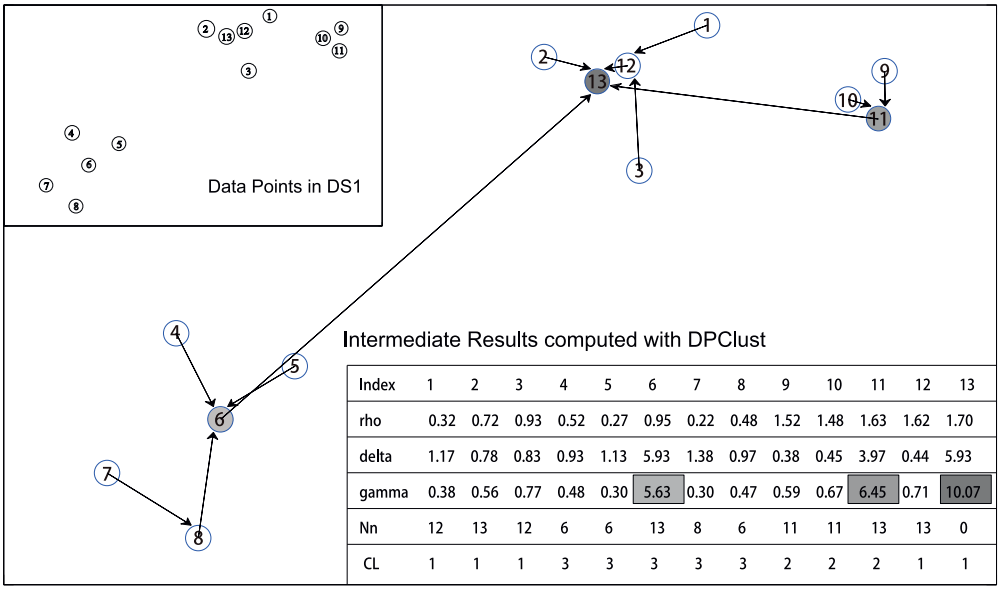
```
Input: Data stream X
Output: Cl, outliers, and change points
Construct the initial LT;
Granulate the LT (Algorithm 2);
while Data X_new streaming in do
    if X_new is not strange then
        Merge X_new into FNLT, output Cl_new (Algorithm 3);
    end
    Buffer BufferSize new data points ;
    for each data point X_new in the Buffer do
        if X_new is noise then
            Discard X_new or store it on hard disk;
        end
        else
            Merge X_new into FNLT, output Cl_new ;
        end
    end
    Detect drift ;
    Fade out, remove weak nodes ;
    Granulate and update FNLT ;
end
```

Steps from 3a) to 3d) are iteratively executed when there are items flowing in. The complexity of DP-Stream is analyzed in Section 5, and the performance of DP-Stream will be empirically assessed in Section 6.

4.1. Granulating the LT/FNLT

To prevent the size of an FNLT from ever growing and to reserve the essential information of the current data points in the stream, we employ two strategies. One is *granulating*, i.e., the closest nodes are merged to their corresponding leading nodes. The other is *fading – removing*. When an item x_m is merged to L satisfying $L = \eta(x_m)$, we perform $T'.\boldsymbol{\rho}_L \leftarrow T.\boldsymbol{\rho}_L + T.\boldsymbol{\rho}_{x_m}$ and $T'.\boldsymbol{W}_L \leftarrow T.\boldsymbol{W}_L + T.\boldsymbol{W}_{x_m}$. Also, for any node y satisfying $x_m = \eta(y)$, we set $L \leftarrow \eta(y)$. For more details of granulating the FNLT, the readers are referred to Algorithm 2.

A manually set constant *Sketch Index* (SI) is used to control how large a proportion of all the nodes are remained after merging the closet data points to their leading nodes. The first N_{merge} points $(x_m^{(1)}, \ldots, x_m^{(N_{merge})})$ in *SortDeltaInds* are merged into their leading



Index	1	2	3	4	5	6	7	8	9	10	11	12	13
rho	0.32	0.72	0.93	0.52	0.27	0.95	0.22	0.48	1.52	1.48	1.63	1.62	1.70
delta	1.17	0.78	0.83	0.93	1.13	5.93	1.38	0.97	0.38	0.45	3.97	0.44	5.93
gamma	0.38	0.56	0.77	0.48	0.30	5.63	0.30	0.47	0.59	0.67	6.45	0.71	10.07
Nn	12	13	12	6	6	13	8	6	11	11	13	13	0
CL	1	1	1	3	3	3	3	3	2	2	2	1	1

Fig. 3. The LT generated from the data points DS1 (the upper left corner), including the intermediate results (the bottom right corner), by which the leading relation and the centers are determined. The three largest gamma values are highlighted with gray background.

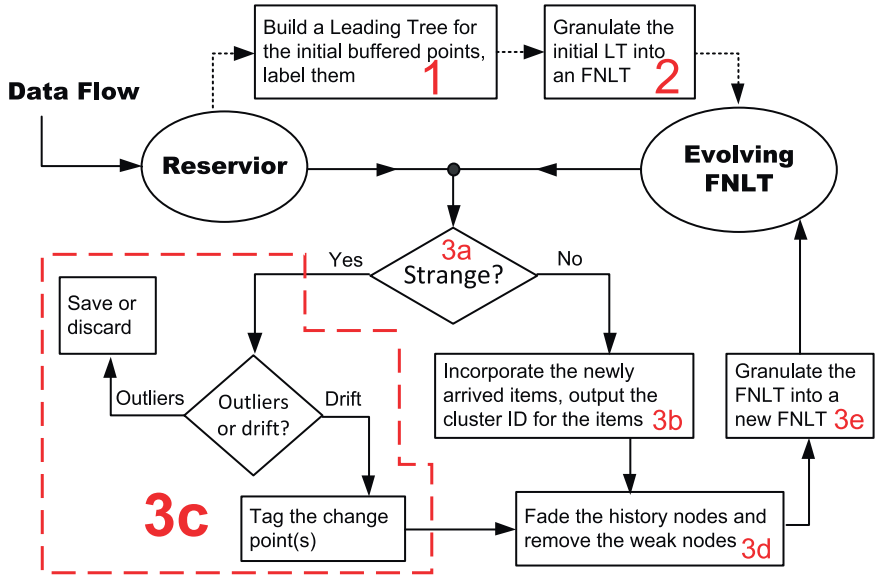


Fig. 4. Diagram of DP-Stream algorithm. The dashed arrows represent the steps executed only once during system initialization.

nodes respectively. If an $L_{(i)} = \eta(x_m^{(i)})$ has been merged to L' already, then $x_m^{(i)}$ is transitively merged to L' in both ***newNn*** and *RemainInds*. ***newNn*** holds the changed ***Nn*** and *RemainInds* is used to store the IDs of fat nodes after the granulation.

Example 2. We continue to demonstrate this granulating-FNLT algorithm using the LT constructed in Example 1. The remaining number of fat nodes would be 6 if we let $SI = 0.55$, since the seven individual points x_9, x_{10}, x_{12}, x_2, x_3, x_4, x_8 are merged into their parents, respectively. As shown in Fig. 5.

4.2. Outliers detection

Outlier detection has a long history since the data are collected or generated from a variety of sources. It finds out and removes the anomalous samples from datasets [28]. Recently, Milos et al. defined *antihubs*, based on reverse nearest neighbor and *hubs*, to detect outliers, especially in high-dimensional setting [29]. Huang et al. combined the ideas of nearest neighbor and reverse nearest neighbor to propose a concept of *natural neighbor*, with which *natural value* and *natural outlier factor* are computed to select the outliers [30].

With DPClust, outliers are characterized by large δ and small ρ [13], thus they can be detected by the ratio of δ to ρ. Similar to the mechanism of selecting centers, we define a parameter $\theta = \delta/\rho$ to indicate how likely a data point would be an potential outlier. This approach can be easily implemented in the context of DP-Stream because the parameters ρ and δ are readily available for every object, and the linear fitting approach to select the anomalously large value is ready for reusing (Section 3.1). However, detecting noises has one thing different from selecting centers, that is, the data points with very large θ might have an extraordinary large δ and a relatively large ρ. And a point with large ρ means that it is surrounded by a group of objects, hence it is by no means an outlier. So, after the first step of choosing the data points with anoma-

Algorithm 2: granulating the FNLT.

Input: the FNLT T, Sketch Index (SI)
Output: a newly granulated FNLT T'
Procedure:
$N_{merge} \leftarrow \lceil N(1-SI) \rceil$;
$\boldsymbol{newNn} \leftarrow T.\boldsymbol{Nn}$;
$[\boldsymbol{\delta^s}, \boldsymbol{SInd_\delta}] \leftarrow SortAscend(T.\boldsymbol{\delta})$;
$RemainInds \leftarrow \emptyset$;
$i \leftarrow 0$;
for *each* j *in the first* N_{merge} *elements of* $\boldsymbol{SInd_\delta}$ **do**
  $i \leftarrow i+1$;
  $RemainInds_i \leftarrow newNn_j$;
  if $newNn_m == j$ *for any* $m \in [1, N]$ **then**
    $newNn_m \leftarrow newNn_j$;
  end
  if $RemainInds_m == j$ *for any* $m \in [1, i-1]$ **then**
    $RemainInds_m \leftarrow newNn_j$;
  end
end
Append $N - N_{merge}$ elements in $\boldsymbol{SInd_\delta}$ to $RemainInds$;
$U \leftarrow \text{Unique}(RemainInds)$;
$T'.X \leftarrow T.X_U$;
$T'.\boldsymbol{w}_i \leftarrow \sum T.w_{M_i}$, where M_i is the nodes set in T merged into $T'.X_i$;
$T'.\boldsymbol{\rho}_i \leftarrow \sum T.\rho_{M_i}$, ;
Extract new $T'.D$ from $T.D$ via $T'.X$;
Update $T'.\boldsymbol{\delta}$ and $T'.\boldsymbol{Nn}$ with $T'.D$ and $T'.\boldsymbol{\rho}$;

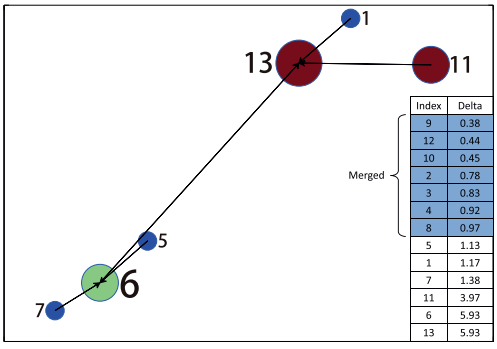


Fig. 5. Illustration of granulating an LT into FNLT. The size reflects the population (the larger size, the greater population), and the color reflects the local density for each node (the warmer color, the higher density).

lously large θ, a second step is needed to filter the points with relatively large ρ. After the two steps, the points left are chosen as outliers and should be simply removed or stored on the hard disk if necessary.

Example 3. The method of using parameter θ to identify strange data points and using buffering mechanism to differentiate outliers from a new pattern is illustrated in Fig. 6. The strange items or outliers are separated from the FNLT. This existing FNLT was constructed in Example 2.

4.3. Merging the new items into the FNLT

A fat node leading tree (FNLT) is defined as a tuple (X, $\boldsymbol{W}$, $\boldsymbol{D}$, $\boldsymbol{\rho}$, $\boldsymbol{\delta}$, $\boldsymbol{Nn}$, $\boldsymbol{Cl}$), in which the meaning of the elements is listed in Table 1. The initial LT can be regarded as an FNLT by assigning its $\boldsymbol{W} = (1, \ldots, 1)$.

For a fast delivery of the clustering result of the items arrived, we assign each item to its leading node immediately after the incremental updating of the local densities for all fat nodes and the newly arrived, leaving the updating of the FNLT (possibly includes the change of centers) after the granulation. The procedure of merging the new items into the FNLT is described as Algorithm 3 .

Algorithm 3: Incrementally updating the FNLT.

Input: the FNLT T, a new item x
Output: an updated FNLT
Procedure:
Step1: //Update $T.\boldsymbol{\rho}$ and compute ρ_x for x
for *each point* x_i *in* $T.X$ **do**
  $d_{i,new} \leftarrow \text{computeDistance}(x_i, x)$;
  $\text{IncreValue} \leftarrow \exp(-(d_{i,new}/dc)^2)$;
  $T.\rho_i \leftarrow T.\rho_i + \text{IncreValue}$;
  $\rho_x \leftarrow \rho_x + \text{IncreValue}^* \; T.W_i$;
end
Append ρ_x to $T.\boldsymbol{\rho}$;
Step2: // Expand $T.\boldsymbol{D}_{N\times N}$ to $T.\boldsymbol{D}_{(N+1)\times(N+1)}$
The bottom row of $T.\boldsymbol{D}_{(N+1)\times(N+1)} \leftarrow [\boldsymbol{d_{new}}, 0]$;
The last column of $T.\boldsymbol{D}_{(N+1)\times(N+1)} \leftarrow [\boldsymbol{d_{new}}, 0]^T$;
Step3://Compute δ_x and Nn_x for x
if ρ_x *is not the biggest* **then**
  $Nn_x \leftarrow \underset{i}{\text{argmin}} \left\{T.D_{i,N+1} | i \in [1, N], T.\rho_i > \rho_x\right\}$;
  $\delta_x \leftarrow \min\{\text{T.D}_{\text{i,N+1}} | \text{i} \in [1, \text{N}], \text{T.}\rho_\text{i} > \rho_x\}$;
end
else
  $Nn_x \leftarrow 0$;
  $\delta_x \leftarrow \max\{\text{T.D}_{\text{i,N+1}} | 1 \le \text{i} \le \text{N}\}$;
end
Step4: //Output the clustering result for x
if ρ_x *is not the biggest* **then**
  $Cl_x \leftarrow Cl_{Nn_x}$;
end
else
  $Cl_x \leftarrow Cl_s$, where $s = \arg\min_i \{D_{i,N+1}\}$;
end
Step5: //update $T.\boldsymbol{\delta}$ and $T.\boldsymbol{Nn}$
if x *does not change the order of* $T.\boldsymbol{\rho}$ **then**
  $\boldsymbol{SI} \leftarrow \{i | \rho_i < \rho_x, 1 \le i \le N\}$;
  for *each* si *in* $\boldsymbol{SI}$ **do**
    if $T.D_{si,N+1} < T.\delta_{si}$ **then**
      $T.\delta_{si} \leftarrow T.D_{si,N+1}$;
      $T.Nn_{si} \leftarrow N+1$;
    end
  end
end
else
  Recompute $T.\boldsymbol{\delta}$ and $T.\boldsymbol{Nn}$ according to the definitions;
end
Append δ_x, Nn_x, x, 1 to $T.\boldsymbol{\delta}$, $T.\boldsymbol{Nn}$, X, $\boldsymbol{W}$, respectively;

Algorithm 3 is the core of DP-Stream. It is an accurate/exact incremental updating method to extend DPClust for data streams.

Lemma 1. *The FNLT constructed with Algorithm 3 is the same as in a non-incremental approach.*

Proof. To show the correctness of Algorithm 3, we indeed show that the incremental approach obtains each element in FNLT tuple (X, $\boldsymbol{W}$, $\boldsymbol{D}$, $\boldsymbol{\rho}$, $\boldsymbol{\delta}$, $\boldsymbol{Nn}$, $\boldsymbol{Cl}$) the same as that obtained in the non-incremental approach.

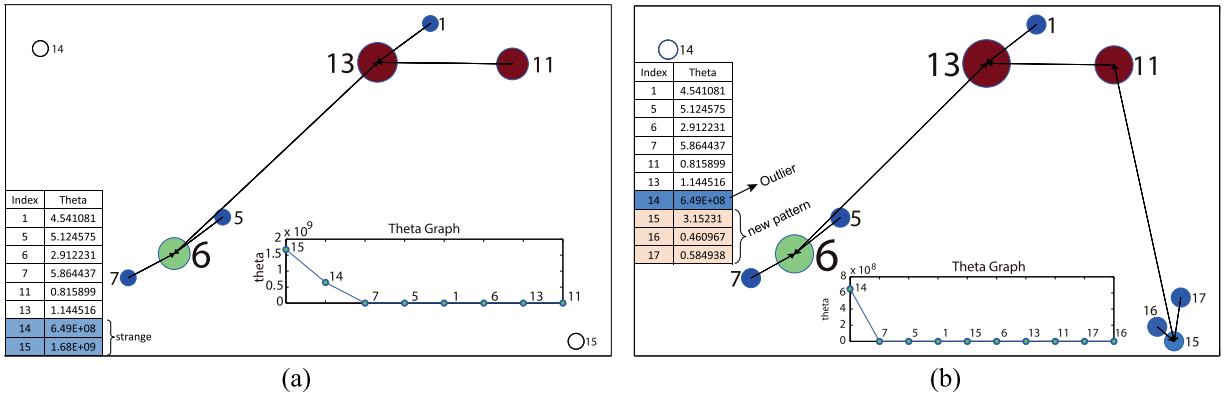


Fig. 6. Illustration of detecting outliers and a new pattern. (a) 2 data points x_{14} and x_{15} arrived and were detected strange with their extraordinary large θ value; (b) x_{14} is identified as an outlier and x_{15} is identified as one item in a new pattern after x_{16} and x_{17} appeared in the data stream.

a) $\boldsymbol{\rho}$. Let us denote $T.\boldsymbol{\rho}$ as $(\rho_1^T, \ldots, \rho_N^T)$, denote $\boldsymbol{\rho}$ computed after the arrival of a new item x via incremental approach as $\boldsymbol{\rho}^{Inc} = \{\rho_i^{Inc}\}_{i=1}^{N+1}$, and denote that via non incremental approach as $\boldsymbol{\rho}^{NonInc} = \{\rho_i^{NonInc}\}_{i=1}^{N+1}$. For the local density of the new item x,

$$\rho_{N+1}^{Inc} = \rho_{N+1}^{NonInc} = \sum_{i=1}^{N} w_i e^{-\left(\frac{d_{i,new}}{dc}\right)^2}. \tag{9}$$

For the density of the fat nodes in the FNLT, i.e., $1 \le i \le N$, we have

$$\rho_i^{Inc} = \rho_i^T + e^{-\left(\frac{d_{i,new}}{dc}\right)^2}, \tag{10}$$

$$\rho_i^{NonInc} = \sum_{1\le j\le N+1, j\ne i} w_j e^{-\left(\frac{d_{i,j}}{dc}\right)^2} = \sum_{1\le j\le N, j\ne i} w_j e^{-\left(\frac{d_{i,j}}{dc}\right)^2} + 1 \times e^{-\left(\frac{d_{i,new}}{dc}\right)^2}, \tag{11}$$

and since

$$\rho_i^T = \sum_{1\le j\le N, j\ne i} w_j e^{-\left(\frac{d_{i,j}}{dc}\right)^2}, \tag{12}$$

Substituting Eq. (12) into Eq. (10), we get

$$\rho_i^{Inc} = \rho_i^{NonInc},\ 1 \le i \le N. \tag{13}$$

Combining Eq. (13) and Eq. (9), $\boldsymbol{\rho}^{Inc} = \boldsymbol{\rho}^{NonInc}$ is obtained.

b) $\boldsymbol{D}$. The newly arrived item does not change the original distances between every pair of existing nodes. So what need to do is to add the distance $D_{N+1,\bullet}$ and $D_{\bullet,N+1}$ to the original $\boldsymbol{D}$, as described in Step 2.

c) $\boldsymbol{Nn}$. The incremental update of $\boldsymbol{Nn}$ consists of two parts: computing Nn_x and updating the existing $\boldsymbol{Nn}$.

(i) computing Nn_x. $Nn_x \leftarrow \underset{i}{\operatorname{argmin}}\{T.D_{i,N+1} | i \in [1, N], T.\rho_i > \rho_x\}$, if ρ_x is not the largest; $Nn_x \leftarrow 0$, otherwise (Step 3). This is the same as in a non-incremental approach.

(ii) updating the existing $T.\boldsymbol{Nn}$. Most frequently, a single new object would not change $T.\boldsymbol{Q}$. So Step 5 of the algorithm first finds such node y that $\rho_y < \rho_x$ (condition 1). Subsequently, if $d_{x,y} < Nn_y$ (condition 2), then $Nn_y \leftarrow x$ (Step 5). This procedure leaves unchanged the Nn of the nodes that do not satisfy the two conditions simultaneously, because of the fact that $T.\boldsymbol{Nn}$ is determined by $T.\boldsymbol{D}$ and $T.\boldsymbol{Q}$. However, if the new item x happened to make some change to $T.\boldsymbol{Q}$, we directly recompute $T.\boldsymbol{Nn}$ using $T.\boldsymbol{D}$ and $T.\boldsymbol{\rho}$.

d) $\boldsymbol{\delta}$. According to the definition of δ (see Eq. (3)), both δ_x and the existing $\boldsymbol{\delta}$ can be obtained once $\boldsymbol{Nn}$ and $\boldsymbol{D}$ are updated correctly.

e) $\boldsymbol{Cl}$. If ρ_x is the largest, then x is labeled as cluster 1 (x is the root of the whole FNLT); else Cl_x is assigned with Cl_{Nn_x}.

f) X and $\boldsymbol{W}$. The update of X and $\boldsymbol{W}$ is trivial, simply let $X = [X, x]$ and $\boldsymbol{W}$=[$\boldsymbol{W}$, 1] (last statement in Step 5).

From the steps a) to f), we can tell that each element in the tuple ($\boldsymbol{X, W, D, \rho, \delta, Nn, Cl}$) obtained via the incremental updating algorithm is the same as that computed non-incrementally. Hence the correctness of Algorithm 3 is proved. □

Example 4. An illustrative example based the FNLT in Example 2 is shown in Fig. 7 to demonstrate the basic idea of Algorithm 3. A new data point (x_{14}) arrives after the LT of 13 points is granulated into an FNLT of 6 fat nodes. Then the new FNLT is obtained after the elements in the definition of an FNLT are incrementally updated or expanded. We will discuss acceleration effect of Algorithm 3 in Section 5.

4.4. Drifts detection

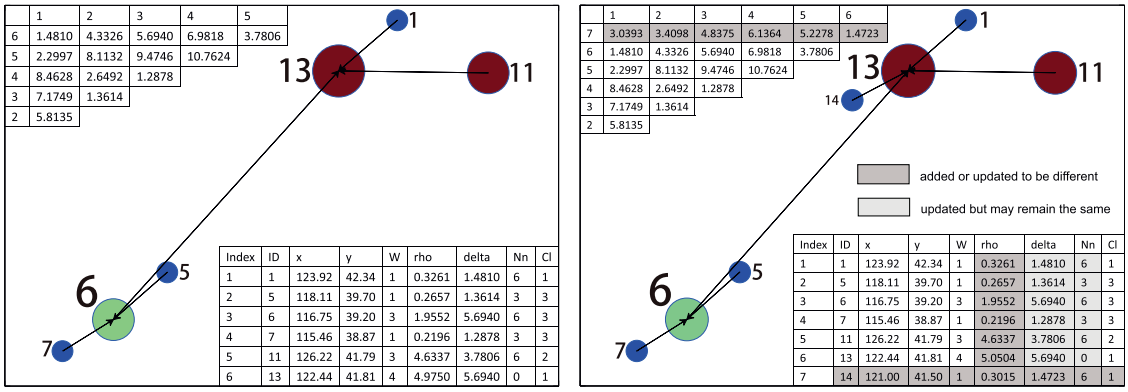
A key difficulty in data streaming is to detect a change in the generative process underlying the data stream, referred to as drift. Ho and Wechsler proposed a martingale framework to detect the change of data generating model [12]. We firstly review the core ideas in Ho's work, and then introduce them into our drift detection problem.

Definition 3. Exchangeable [12]. Let $\{Z_i: 1 \le i < \infty\}$ be a sequence of random variables. A sequence of random variables $Z_1, \ldots, Z_n, \ldots$ is *exchangeable* if for every finite subset of the random variable sequence(containing n random variables), the joint distribution $p(Z_1, \ldots, Z_n)$ is invariant under any permutation of the indices of the random variables, i.e, $p(Z_1, Z_2, \ldots, Z_n) = p(Z_{(1)}, Z_{(2)}, \ldots, Z_{(n)})$, for all permutations π defined on the set $\{1, \ldots, n\}$.

A change in general in a data streaming setting is defined by a change in the parameter θ from θ_0 to θ_1 at time t_0 [31]. We adopt this claim in DP-Stream.

Definition 4. Martingale [12]. A sequence of random variables $\{M_i: 0 \le i < \infty\}$ is a martingale with respect to the sequence of random variables $\{Z_i: 0 \le i < \infty\}$ if, for all $i \ge 0$, the following conditions hold:

- M_i is a measurable function of $Z_0, Z_1 \ldots, Z_i$,
- $E(|M_i|) < \infty$, and



(a) (b)

Fig. 7. Illustration of Algorithm 3. (a) The FNLT and its intermediate results before the new item arriving. (b) The updated FNLT after the new item has been incorporated.

- $E(M_{n+1}|Z_0, \ldots, Z_n) = M_n$.

Let Z_i be the *prime center* (the root) C^T of the whole FNLT at time i, and M_i be the distance between Z_i and Z_{i-1}. Intuitively, M_i would remain relatively small with some fluctuations as long as the drift does not occur.

Before applying Martingale theory to DP-Stream, we make an assumption: a drift occurs at time i only if there are at least two clusters in the reservoir at time $i-1$. The assumption makes sense because the reasonable choice of reservoir size (batch of items we used to rebuild the FNLT) is much smaller than total weight (population) of the evolving FNLT nodes. Therefore, the drift must occur gradually with the old pattern fading out and a new pattern emerging simultaneously.

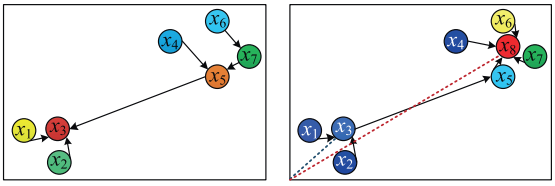
In DP-Stream, we say a drift occurs if and only if the location of the highest cluster center C_i^T has changed so significantly that the distance from the current C_i^T to previous C_{i-1}^T is farther than that to a previous non-top center. With the definitions given in Section 3.2, it can be put formally:

Lemma 2. *The time i is a change point, if $\exists C_{i-1}^S \prec C_{i-1}^T$, such that* $T.D_{C_i^T, C_{i-1}^S} < T.D_{C_i^T, C_{i-1}^T}$.

Proof. In DPClust, ***Cl*** is assigned in accordance with the ρ value of their corresponding centers, i.e., $Cl_{\rho Ind_1}$=1, $Cl_{\rho Ind_2}$=2, and so forth, where ρInd_i is computed within $[\boldsymbol{\rho^s}, \boldsymbol{\rho Ind}] = sortDesc(\boldsymbol{\rho})$. If $T.D_{C_i^T, C_{i-1}^S} < T.D_{C_i^T, C_{i-1}^T}$, which means the root of the FNLT is changed to another center, and let θ be the coordinate of C^T in the space in which X is embedded, then θ must change from θ_0 to a much different value θ_1. Because the definition of cluster center in DPClust implies that a center C^j is surrounded by the items with lower local density but *relatively far* from the points with higher local density (including C^k satisfying $C^j \prec C^k$). □

An example to illustrate Theorem 2 is shown in Fig. 8. The positions, colors and the leading relations of the each point in Fig. 8 are based on real data and computing results. In Fig. 8a, x_3 is C^T, for the three data points x_1, x_2, x_3 are closer to each other than the other four points, which leads to the highest ρ of x_3. In the next time (Fig. 8b), a new item x_8 arrives and locates among $x_4, \ldots, x_7$, hence x_8 is of the greatest local density and selected as the new C^T. If we denote the coordinate of x_3 as $\theta_0 = (r_0, \varphi_0)$, and the x_8 as $\theta_1 = (r_1, \varphi_1)$, obviously we have $\theta_0 \neq \theta_1$.

A change point indicates a significant drift. Therefore, the clustering result should be evaluated independently since the clusters with same label have actually changed into different distributions.



(a) (b)

Fig. 8. Illustration of major concept drift. (a) x_3 is the root of the whole LT before x_8's arrival, hence the cluster label for $\{x_1, x_2, x_3\}$ is 1. (b) x_8 is the new root of the whole LT after x_8 being incorporated, hence the cluster label for $\{x_1, x_2, x_3\}$ is changed into 2.

4.5. Fading out and removing weak nodes

To emphasize on the recent data and gradually forget the old ones, we use a fading-removing mechanism on the historical data. A parameter named *half-life* is firstly defined as in [3]. We choose an exponential form for the fading function because of its widespread application in temporal systems, where the importance of past data needs to be gradually decreased.

Definition 5. Half-life. In DP-Stream, the half-life t_0 of a point x_i is defined as the time at which $w_i \cdot f(t_0) = w_i \cdot (1/2)$, where $f(t) = e^{-\lambda t}, \lambda > 0$.

Because $e^{-\lambda t} = 1/2 \Rightarrow \lambda = \ln 2/t$, we can easily determine the parameter λ if t_0 is given. In the experiments, we perform $\boldsymbol{W} = \boldsymbol{W} \cdot e^{\lambda}$ and $\boldsymbol{\rho} = \boldsymbol{\rho} \cdot e^{\lambda}$ after reconstructing the FNLT every time. After many rounds of fading, some points will have very small weight (e.g. $<$ *RemovalThre*) if no new points are merged into them. In this case, they are removed from the FNLT.

5. Computational complexity

Space complexity of DP-Stream depends on the *Sketch Index*(*SI*) and the size of reservoir. Since the FNLT is defined as (*X*, ***W, D, ρ, δ, Nn, Cl***), size memory space used to hold an FNLT of $n_f (= StartBuf\,ferSize * SI)$ nodes is $n_f \times d + n_f^2 + 5 \times n_f$, where $n_f \times d$, n_f^2, and $5 \times n_f$ is the size of X, D and the 5 vectors {***W, ρ, δ, Nn, Cl***)} respectively. Although some components in FNLT can be computed from others, we store them to trade memory for running time. Let n_r be the size of reservoir that means the number of items used to rebuild the FNLT, then the space needed in merging and granulating procedure (apart from the FNLT) is $n_r \times d + (n_r^2 + 2 \times n_f \times n_r) + 4 \times n_r$, where $n_r \times d$ is the number of storage size for new items, $n_r^2 + 2 \times n_f \times n_r$ for the expanded dis-

Table 2
Datasets used to empirically evaluate DP-Stream.

Datasets	# Attributes	# Clusters	# Objects
ExclaStar	2	2–3	755
MRDS	2	2–3	42,470
ChameleonDS3	2	2–5	10,000
RBF10a	10	5	1,000,000
RBFDrift	15	5	1,000,000
CoverType	54	3–7	581,012
KDD'99	34	1–4	494,020

tance matrix before granulating, and $4 \times n_r$ for the 4 vectors {***ρ, δ, Nn, Cl***}. Therefore, the total space needed to update the FNLT with newly arrived items and granulate the FNLT is

$$SC = (n_f + n_r) \times d + (n_f + n_r)^2 + 5 \times n_f + 4 \times n_r. \quad (14)$$

We remove weak nodes from the FNLT. Therefore, the clustering method uses the space never exceeding *SC*.

To analyze the algorithm's time complexity, we need to consider Algorithm 1, Algorithm 2 and Algorithm 3. From Algorithm 1, it is clear that astrange item will require more steps to process than a non-strange item, although this does not affect the big *O* notation of DP-Stream. The highest complexity part of the DP-Clust to construct the initial LT is the distance matrix computation, which is $O(n_f^2)$. In Algorithm 2 for granulating the FNLT, the closest data points indicated by the δ parameter are merged to their leading point respectively. The most time consuming part in Algorithm 2 is the transitive updating of ***Nn*** values in ***newNn*** and *RemainInds*, whose complexity is $O(N_{merg} \times (N_{merge} + n_f + n_r))$, where N_{merge} is the number of nodes to be granulated into other leading nodes. Then, for the delivery of the clustering result of a non-strange new item, the complexity in each step is $O(n_f)$ in Algorithm 3. That is, DP-Stream can find the cluster assignation of a non-strange new item in $O(n_f)$ time complexity. This is a highlight of our method. The time complexity in outlier detection, as well as in fading out function, is linear.

In summary, the time complexity of building the initial LT is $O(N^2)$, and the complexity in updating and granulating the FNLT is $O(n_f)$ and $O(N_{merge} \times (N_{merge} + n_f + n_r))$. The overall complexity of DP-Stream is $O(N^2)$. DP-Stream has the virtue of assigning the cluster label to a new item X_{new} in $O(n_f)$ time complexity if X_{new} is not strange, which is to our best of knowledge the fastest delivery of clustering result in data stream clustering.

6. Experiments

The experiments are conducted on a personal computer with Intel i5-2430M CPU, 8G RAM, Windows 7 64bit OS, and Matlab 2014 programming environment. We test DP-Stream on 7 datasets: five of them are synthetic and the others are real world datasets from UCI Machine Learning Repository. Among the three 2D synthetic data streams, ChameleonDS3 is downloaded from Karypis Lab[2]; ExclaStar is generated originally in this paper, and MRDS is reproduced with the description in [7] with the help of Engauge Digitizer.[3] The other two synthetic datasets RBF10a and RBFDrift are generated by *generators.RandomRBFGenerator* and *generators.RandomRBFGeneratorDrift* methods of the open source software MOA (Massive Online Analysis)[4] respectively. The details of the seven datasets are listed in Table 2.

We compare the accuracy performance of DP-Stream with the classic CluStream [16] and DenStream [6] (implemented in MOA), as well as the state-of-the-art method STRAP [9]. The STRAP source code is generously offered by Zhang from the Internet.[5]

Table 3
Parameters in DP-Stream.

Parameter	Purpose	Interval
percent	to decide *dc* in DPClust	[0.05, 20]
LocalR	for selecting centers	[1.6, 4.8]
GlobalR	for selecting centers	[0.01,0.3]
InitialBuffSize	the number of items to construct the initial LT	[300, 2000]
SI	the rate of compressing the initial LT into an FNLT	[0.75, 0. 95]
bufferSize	the number of items to rebuild the FNLT	[20,1000]
half-life	the speed of fading out	[2,10]
RemovalThre	how weak a fat node should be removed from the FNLT	[0.1,0.5]

Table 4
Configurations of the three parameters.

Dataset	*percent*	*LocalR*	*GlobalR*
ExclaStar	5	4.8	0.1
MRDS	2	4.8	0.3
ChameleonDS3	0.5	4.8	0.2
RBFa10	0.2	2.6	0.1
RBFDrift	0.2	2.6	0.1
CoverType	20	4.8	0.3
KDD'99	2	1.6	0.05

The metric Purity to evaluate the stream clustering accuracy is defined as in [3]:

$$Purity = \frac{\sum_{i=1}^{K} \frac{|C_i^d|}{|C_i|}}{K} \times 100\%, \quad (15)$$

where *K* is the number of real clusters. The symbol $|C_i^d|$ denotes the number of points with dominant class label in cluster *i*, and $|C_i|$ is the number of points in real cluster *i* [7]. Because Purity has the limitation of favoriting smaller number of clusters, the metric Adjusted Rand Index (ARI) [32,33] is also employed in our evaluations. ARI and Purity can be used only with the presence of the external class label (ground truth), hence they are called external validation measures [4]. However, if there is no class label in the data stream, some internal validation measures will be chosen. For example, in the case of ChameleonDS3, we choose Silhouette Index [34] to evaluate the clustering quality achieved by DP-Stream and the comparative methods. Besides, internal validation measures have the ability to reflect the compactness and separability of clustering results, so we additionally employe Silhouette Index to evaluate all clustering results in our experiments.

DP-Stream involves 8 parameters in total, whose purposes and suggested value intervals are listed in Table 3. However, we find out that the clustering results are only sensitive to the first 3 parameters. In each experiment with the corresponding dataset, we give our parameters configuration in a row of Table 4. For example, the 3rd row shows the parameters setting in MRDS experiment. How these parameters are set will be discussed in Section 6.4.

6.1. 2D synthetic datasets

The three 2D synthetic datasets (as in Fig. 9) are chosen for different purposes. ExclaStar is designed to test the ability of a stream clustering model to find clusters of different density level and non-spherical shapes, and to track the mergence of clusters; MRDS is chosen to determine whether a clustering can filter the noises and

[2] http://glaros.dtc.umn.edu/gkhome/cluto/cluto/download.
[3] http://markummitchell.github.io/engauge-digitizer/.
[4] http://moa.cs.waikato.ac.nz/.
[5] http://mine.kaust.edu.sa/Pages/Software.aspx

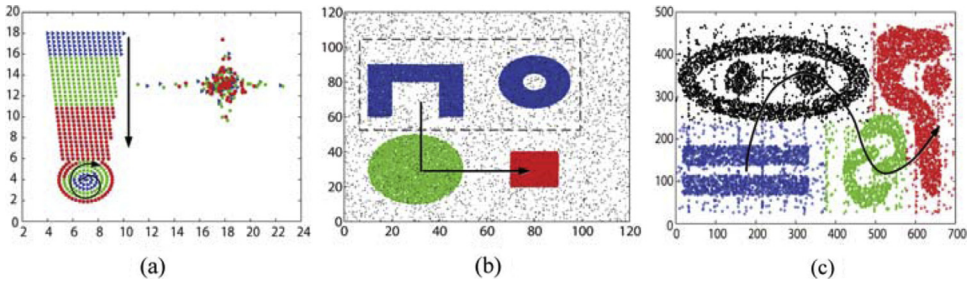


Fig. 9. The three 2D data streams. The points appear in the color sequence of "blue → green → red". (a) ExclaStar. It consists of an exclaim symbol and a star. (b) MRDS [7], and (c) ChameleonDS3. Its file name in the downloaded zip is "t7.10k.dat", and the points are without class label. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

identify the concept drifts; ChameleonDS3 is borrowed to demonstrate one limitation of the proposed method: when a cluster contains several internal high density spot, DP-Stream usually fails to detect the whole big cluster. This is because the assumption of DP-Clust (a cluster consists of a center of high density and many surrounding points of relatively lower density) does not hold in this circumstance.

6.1.1. ExclaStar dataset

ExclaStar dataset consists of 240 points for the star, 391 for the bar and 124 for the pie. Apart from the reasons mentioned above, we additionally visualize the whole clustering procedure on this dataset and gain insight on how the algorithm DP-Stream works because it has relatively fewer points.

As expected, it is shown that the FNLT has 3 clusters of points at the beginning stage (Fig. 10a) and gradually 2 clusters on the left side merge into one (Fig. 10b). It is also demonstrated that the cluster labels are assigned to newly arrived items as soon as they are incorporated into the FNLT.

From the point No. 708, the cluster 2 and cluster 3 should be merged into one. Therefore, the ground truth is modified accordingly. DP-Stream finally obtains an average purity of 99.77% with the parameters setting as that in the first row of Table 4. Because the basis of DP-Stream (DPClust) has the power of detecting the non-spherically shaped clusters such as a star or a bar, DP-Stream obtains a smooth high Purity and ARI score on the dataset ExclaStar. However, the measure Silhouette Index is fluctuating because it favors the well-separated compact spherical clusters, which is quite different from the case of ExclaStar. STRAP achieves a relatively lower ARI and Purity because its basis – AP clustering is not able to detect the right clusters in ExclaStar. The results are shown in Fig. 11.

6.1.2. MRDS dataset

MRDS is a synthetic data set of 38.7K records, containing 2 nonconvex-shaped clusters and 2 convex-shaped clusters with 10% noise. The ranges of both the x and y dimensions are set as [0, 120] [7]. It would greatly improve the accuracy of DP-Stream to use the parameter θ (defined as $\theta = \delta/\rho$ in Section 4.2) to detect outliers in stream. Because with the existence of outliers, the structure of the FNLT will be influenced if the outliers are regarded as normal items. In the test with MRDS dataset, the majority of outliers are successfully selected, as shown in Fig. 12.

Although the clustering result for each new item is permanently delivered, the granulation of FNLT is performed only when the reservoir for buffering the new items is fully occupied. Therefore, the number of times that FNLT granulating points N_{gr} is computed as:

$$N_{gr} = \lceil (N - InitialBuffSize)/bufferSize \rceil + 1, \quad (16)$$

e.g., $\lceil (38700 - 1000)/500 \rceil + 1 = 77$ in this experiment. Three change points are detected at the positions of 48, 54, and 74 out of 77 FNLT granulating points, respectively. Finally, DP-Stream achieves competitive ARI and Purity against other comparative methods, as shown in Fig. 13.

6.1.3. ChameleonDS3 dataset

As shown in Fig. 9c, the clusters in ChameleonDS3 are not "smoothly" distributed. Each cluster contains a few internal density peaks, which violates the assumption of the cluster characteristics defined in [13] saying: *cluster centers are surrounded by neighbors with lower local density ...* . Therefore, DP-Stream achieves a lower Silhouette Index than other methods on this dataset since it tends to detect one cluster per density peak, and consequently results in more clusters than needed.

The source code implementing STRAP needs the class labels in the function named *build_cluster*, so the clustering result of STRAP on ChameleonDS3 is not available. The results produced by CluStream, DenStream and DP-Stream are depicted in Fig. 14.

6.2. RBF10a And RBFDrift datasets

These two datasets are generated based on Radius Basis Function. RBF10a is a stationary data stream with its centers staying almost still all the time, while RBFDrift is an evolving data stream whose centers keep moving as the new items flow in.

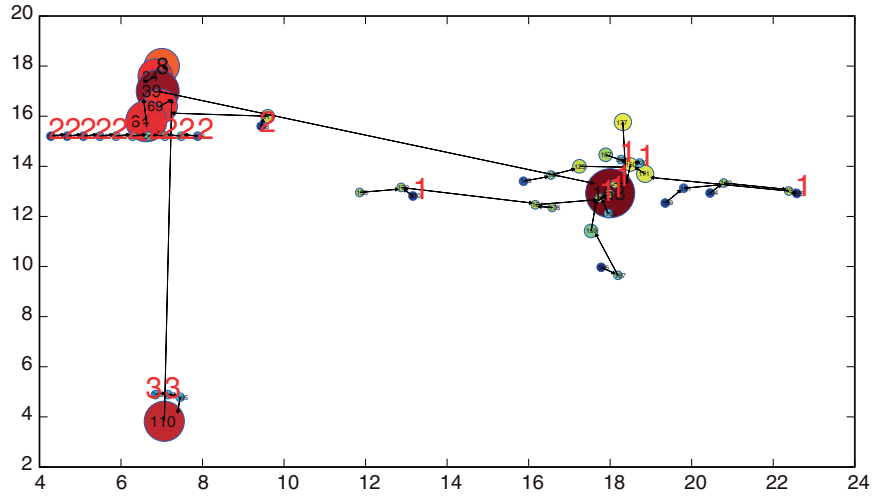
6.2.1. RBF10a

The initial buffered data points of RBF10a are visualized using 2D multidimensional nonclassical scaling in Fig. 15, from which we can see that the dataset has cluster No.1 and cluster No.2 located near to each other and they are of different density levels. In this situation, DPClust tends to identify the two clusters as one. Because DPClust is unable to determine the right number of clusters, the accuracy of DP-Stream on the dataset RBF10a is thus affected. However, DBSCAN can find the correct clusters in the initial dataset, which leads to a better performance of DenStream than that of DP-Stream. As for STRAP, it clusters the stream as only one class in most time, which results in a rather high purity but low ARI. The overall comparison is shown in Fig. 16.

6.2.2. RBFDrift

RBFDrift is generated with the command *generators.RandomRBFGeneratorDrift -s 0.002 -k 5 -i 5 -c 5 -a 15 -n 5*. DP-Stream detects a total of 1746 changes in the dataset and achieves a high Purity of 96.26% with the average number of clusters 5.0423. The performance of the comparative methods on RBFDrift is shown in Fig. 17.

With the data stream RBFDrift, we find out that CluStream and DenStream both obtained a little better Purity and Silhouette Index than DP-Stream. This is because the stream generated by



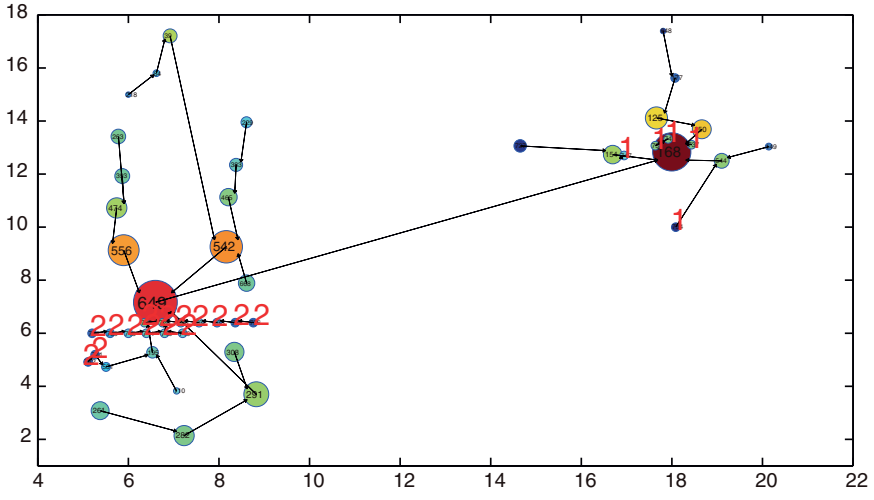


Fig. 10. Exemplar stages of Clustering in ExclaStar Dataset: (a) in the earlier stages there are three clusters, (b) near the end of the stream the two clusters on the left side merge into one.

radial basis function is actually spherically shaped. However, because this stream keeps constantly changing its positions of the clusters, CluStream and DenStream tend to cluster it as more clusters, hence the relatively lower ARI. Contrary to DenStream, STRAP clusters RBFDrift as smaller number of clusters, so STRAP achieves a lower ARI but a higher Purity. Regarding to the three validation measure comprehensively, DP-Stream achieves a very promising result on RBFDrift.

6.3. Real datasets

DP-Stream is evaluated also on two real world datasets: the well known KDD'99 and another UCI dataset named Forest CoverType dataset (referred to as CoverType in this paper).

6.3.1. CoverType dataset

The CoverType data is about predicting forest cover type from cartographic attributes. Although it is not a typical data stream set, we use it in a stream fashion, as in [11]. We normalize the attributes by min/max method and adopt Euclidian distance metric to compute the parameters ρ. The ground truth groups the instances into 7 different forest cover types, but the averaged number of classes over each evaluation window is 3.06.

Before the steam clustering algorithm functions, we begin with constructing an initial LT with 500 records. The initial LT detects a number of clusters as 4, which is in accordance with the ground truth. In the process of clustering the whole CoverType dataset, the averaged number of clusters given by DP-Stream is 2.5. The proposed clustering finally results in an average purity of 95.91% and averaged ARI of 0.3049, which outperforms the comparative methods (see Fig. 18). Note that the measures Purity and Silhouette Index for CluStream are incorrect for a software issue, so they are not included in the figure.

DP-stream performs better than STRAP because the data points in each cluster typically form a non-spherical shape (we validated this by visualizing the points on 2D plane). The result of DP-Stream is also slightly better than that of DenStream. The possible reason is that DenStream did not find a proper value for threshold parameter ϵ and μ that are used to determine neighbors and core micro-clusters, respectively. Meanwhile, DP-Stream is less sensitive on parameter variation to achieve the best performance (see more details in Section 6.4).

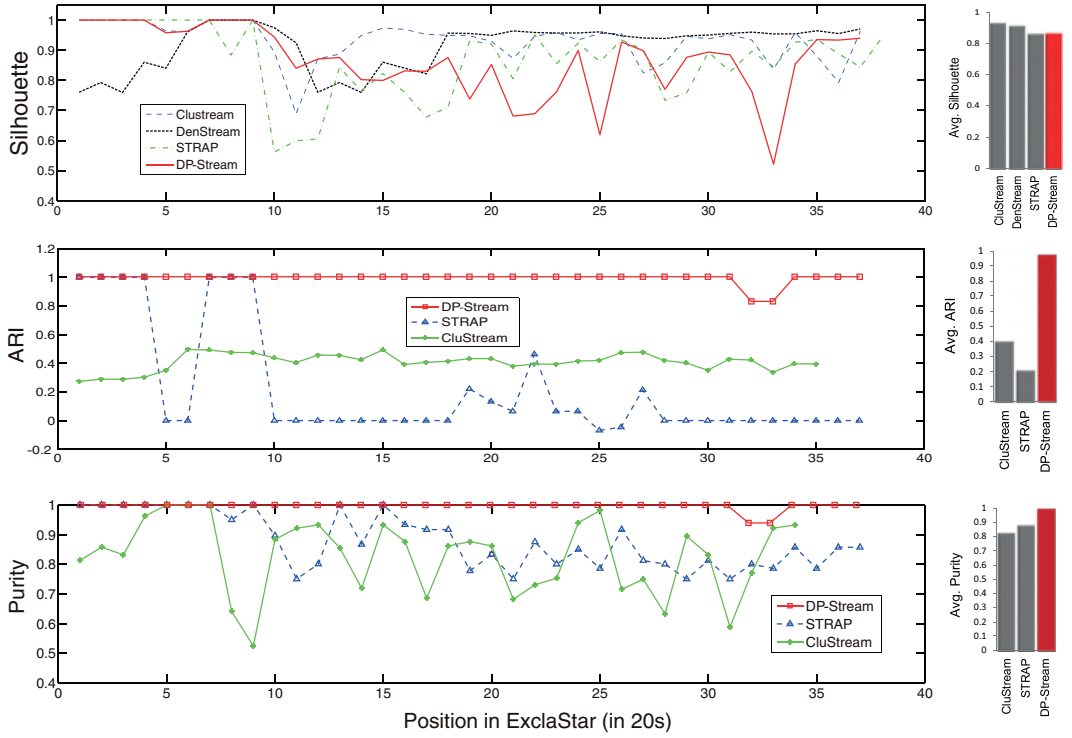


Fig. 11. ExclaStar Dataset. Because there is a bug when running DenStream for this small dataset, we do not include the result of DenStream in the sub-figures of ARI and Purity.

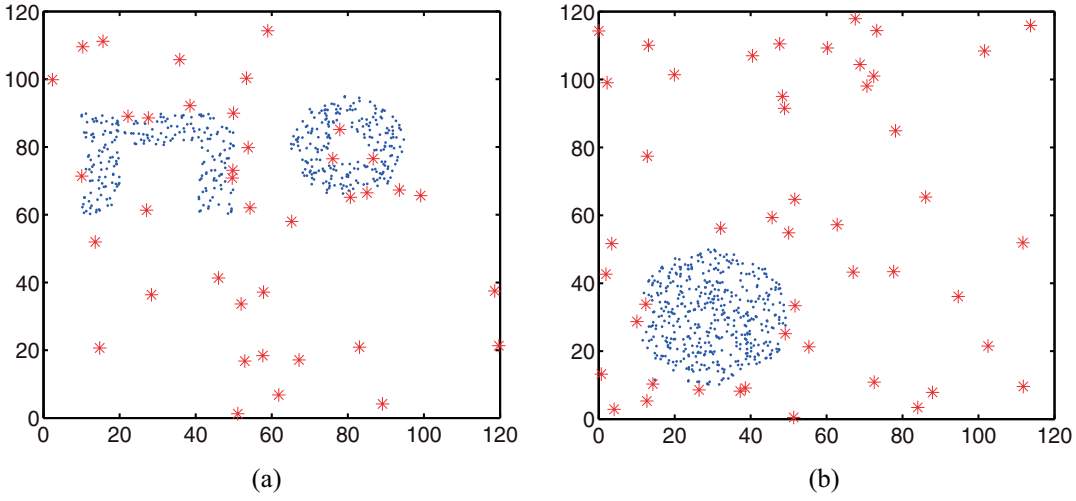


Fig. 12. Test the outliers detection by θ values in two typical stages in data stream MRDS. The selected outliers are displayed as red asterisks. (a) position = 2 of 84 windows; (6) position = 52 of 84 windows. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

6.3.2. KDD'99 dataset

Zhang et al. used STRAP program to build a clustering model on 1 percent of KDD'99 dataset [9], but we enlarge the proportion up to 10 percent. The dataset is preprocessed firstly by choosing its 34 numerical attributes and then normalizing the attributes by their semantics as suggested in [9].

By visualizing the initially buffered data with 2D nonclassical multi-dimensional scaling, one can tell that the KDD'99 is typically non-convex shaped. Therefore, DP-Stream has sound reason to perform better than the other three methods in most of the time. However, KDD'99 has a unique feature different from the previous six datasets: the number of clusters varies dramatically. As an intrusion detection dataset, KDD'99 usually has only one cluster labeled as "normal". But when several types of intrusions occur within a reservoir, the number of clusters may increase to a maximum of 6. Because DP-Stream currently adopts a static d_c policy, it is challenging for this model to adjust well to so wide range of cluster numbers. Thus DP-Stream suffers in the middle phase of KDD'99, which causes a lower averaged ARI than the other three methods. The performances of DP-Stream and the other three methods are depicted in Fig. 19.

6.4. Configuration and sensitivity of parameters

DP-Stream involves 8 parameters in clustering data streams, whose purposes and recommended value intervals are summarized in Table 3. For simplicity without sacrificing much accuracy, we set *half-life*=5 and *RemovalThre*=0.5 through

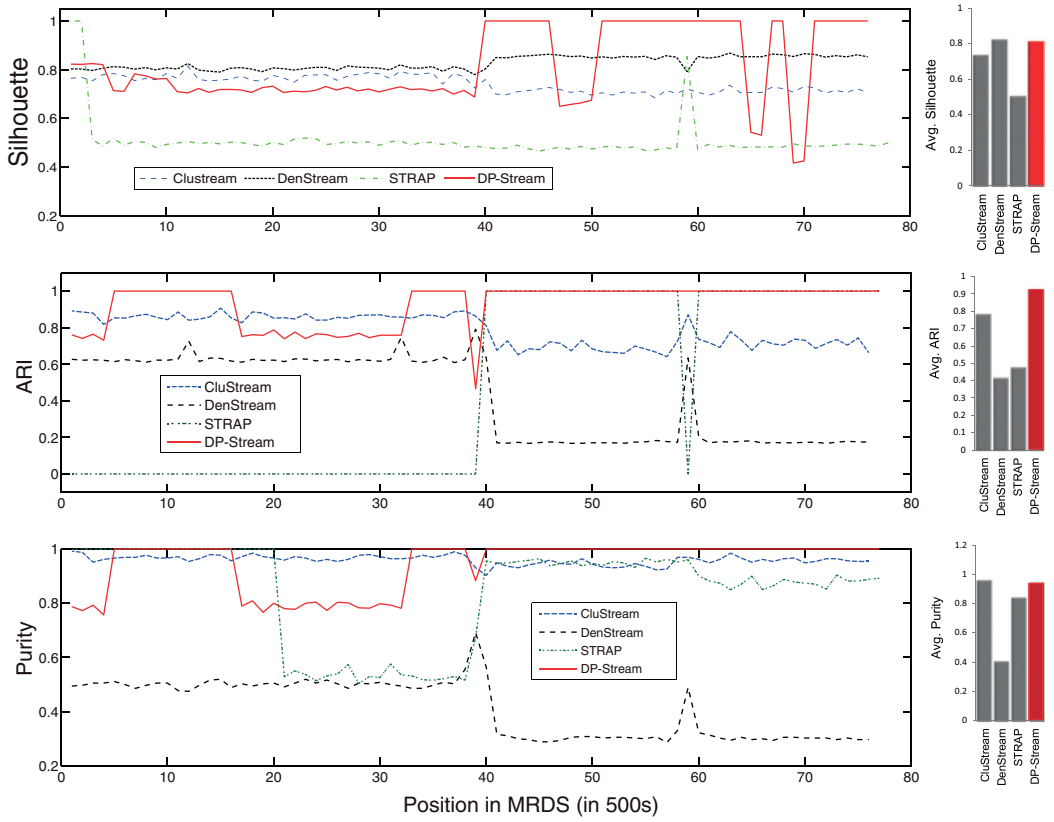


Fig. 13. Clustering results in data stream MRDS.

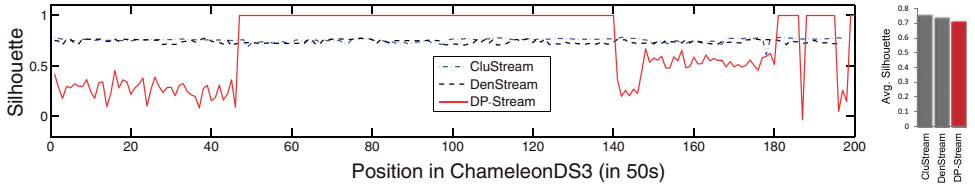


Fig. 14. Clustering results on ChameleonDS3 [35].

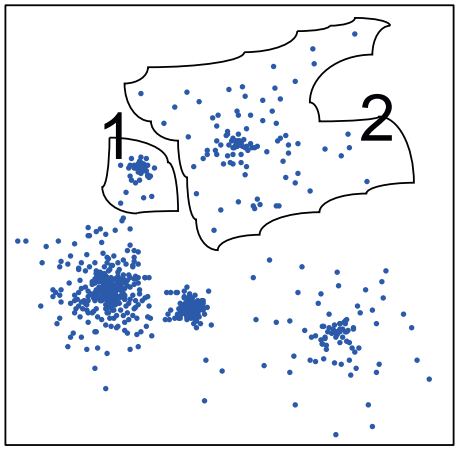


Fig. 15. 2D visualization of the initial buffered data points in RBF10a.

out the experiments. Except ExclaStar, for which we set $\{InitialBuffSize = 180,\ \ buffSize = 20,\ \ SI = 0.75\}$, the configuration $\{InitialBuffSize = 1000,\ \ buffSize = 50,\ \ SI = 0.90\}$ is used for the other six datasets. Therefore there are only 3 relatively more sensitive parameters (i.e., *percent*, *LocalR*, and *GlobalR*) left to be tuned.

The clustering result may be rather poor if the parameters are set without considering the distribution of the data points. However, reasonable values assigned to these parameters would achieve good performance (both in accuracy and efficiency), and small fluctuation around the well-chosen values makes little changes in the clustering result. Therefore, we apply a grid search technique [36] on the combination of the typical values of *percent*, *LocalR*, and *GlobalR* when the initial LT is constructed. With this policy, we first let *percent* $\in$ {0.2, 0.5, 2, 5, 20}, *LocalR* $\in$ {1.6, 2.6, 4.8} and *GlobalR* $\in$ {0.05, 0.1, 0.2, 0.3}, then we evaluate the total $5 \times 3 \times 4 = 60$ clustering results induced by the different parameter configurations under the internal evaluation metric *Silhouette Index* [34]. The configuration achieving highest Silhouette Index is selected for the later clustering process. With this grid search strategy, we find the best configuration of the 3 parameters for the seven datasets as tabulated in Table. 4.

The sensitivity of parameters {*Percent, LocalR, GlobalR*} is tested on the dataset CoverType by fixing any two of them and varying the third parameter. In addition to the values we search in the

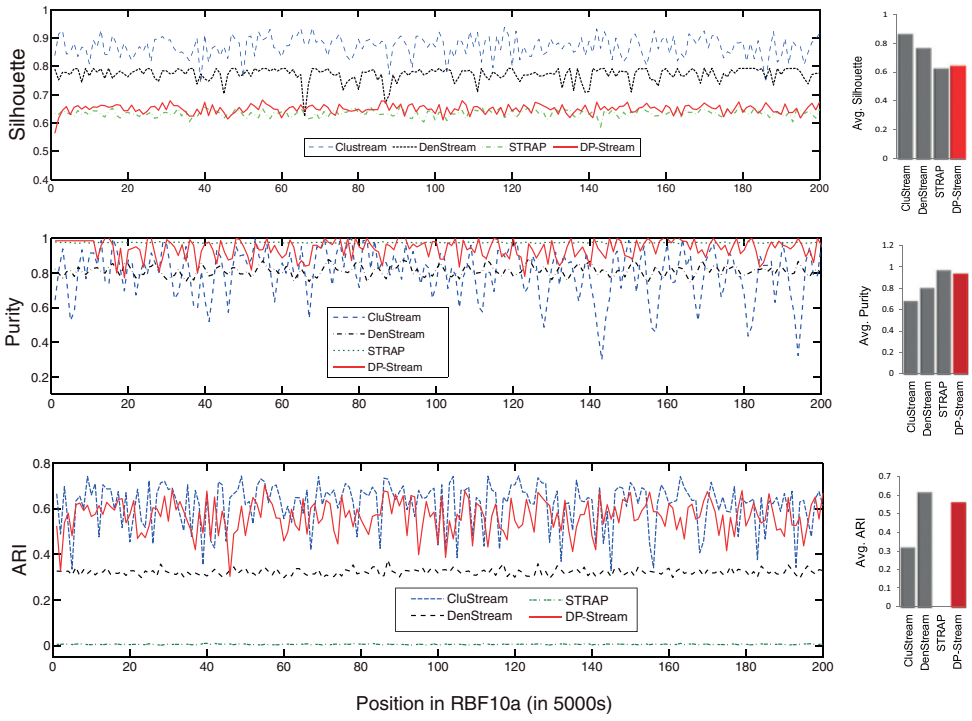


Fig. 16. Clustering results in data stream RBF10a.

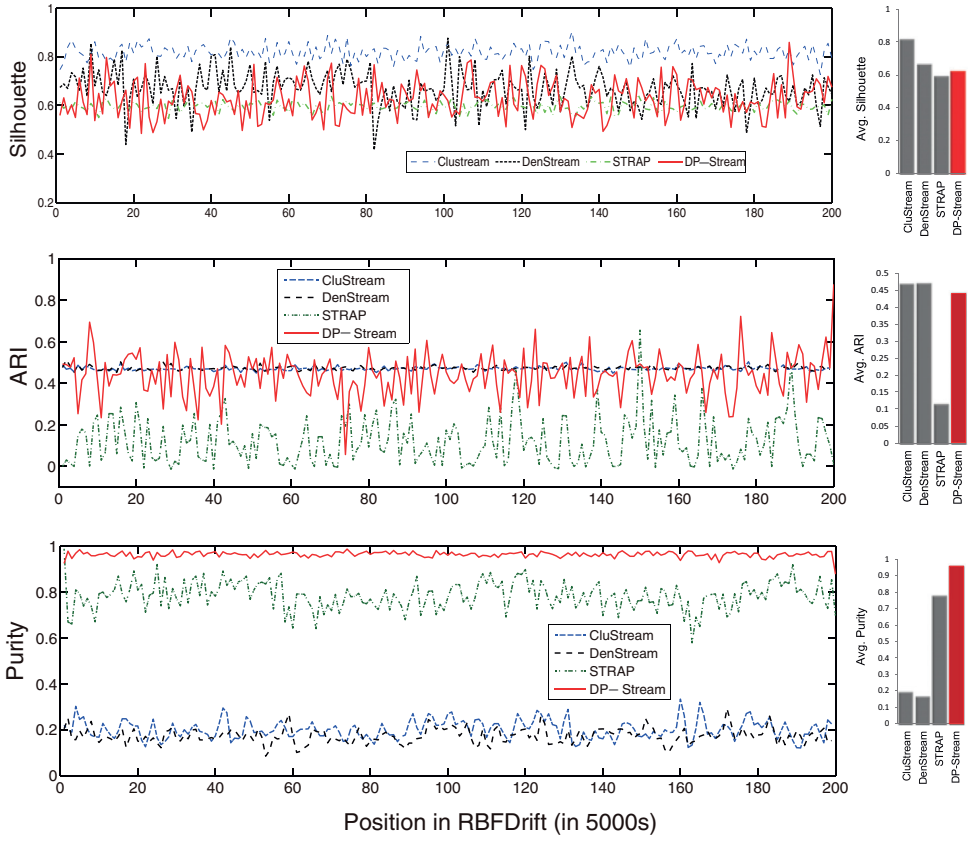


Fig. 17. Clustering results on data stream RBFDrift.

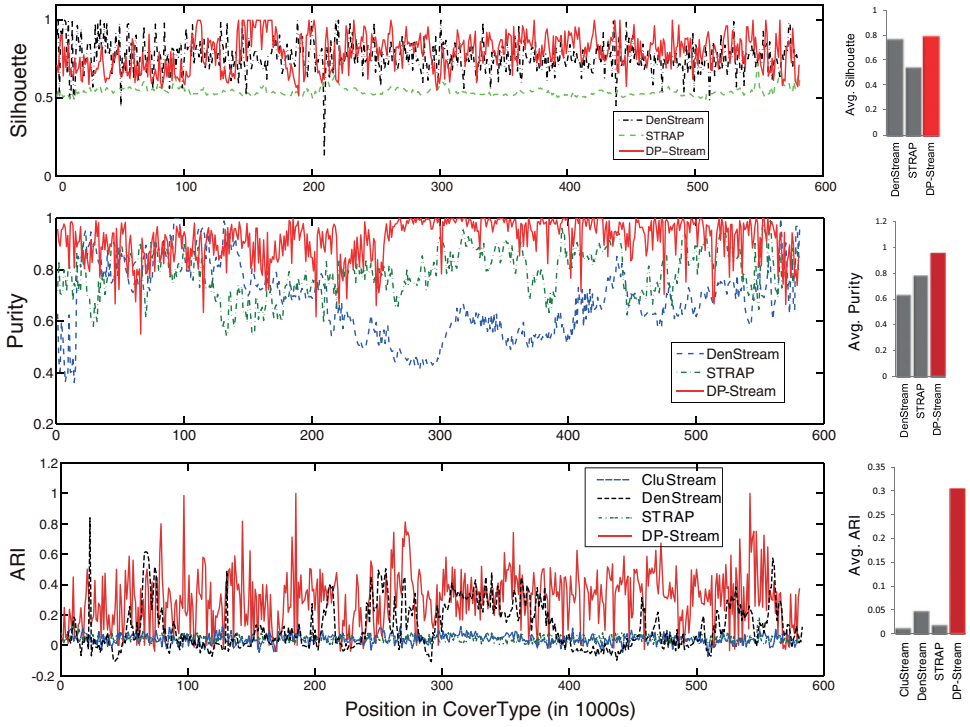


Fig. 18. Clustering results on CoverType Dataset.

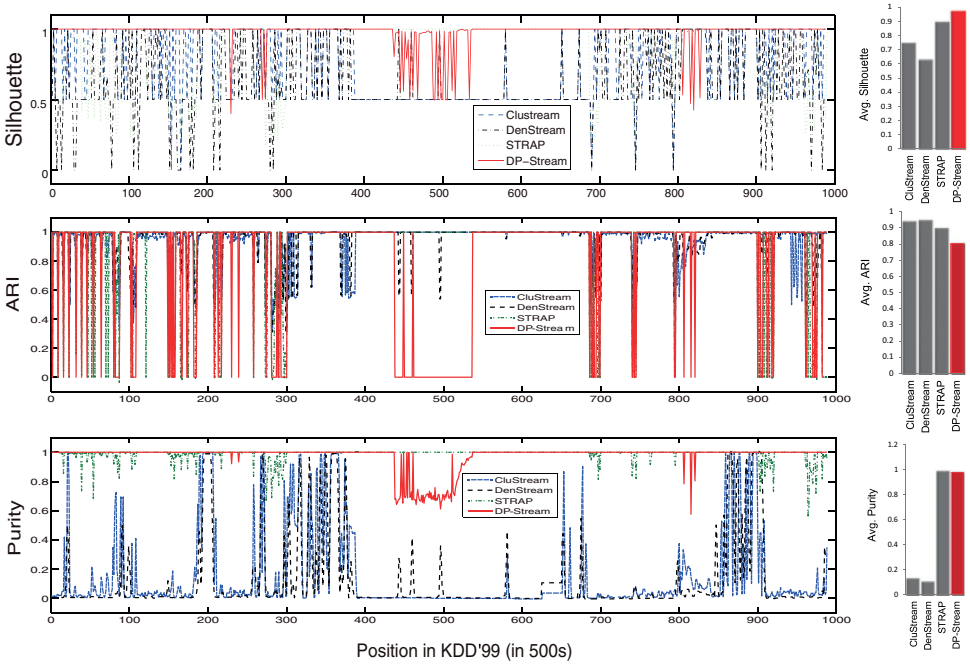


Fig. 19. Clustering results on data stream KDD'99.

grid, we add a smaller and a greater value out of the recommended range, e.g., 0.01 and 80 for the parameter *Percent*. The sensitivity is shown in Fig. 20, from which one reads that if two of the parameters are well chosen, then variation of the third does not change the accuracy significantly. However, if all of the three parameters are chosen badly, the accuracy would decrease considerably. For example, the values $\{Percent = 0.01, LocalR = 1.2, GlobalR = 0.001\}$ produce a much lower Purity and ARI of 85.88% and 0.2582 respectively.

6.5. Overall performance comparison and discussion

6.5.1. Accuracy comparison

An overall performance comparison between DP-Stream and the other 3 competing methods on the 7 datasets is summarized in Fig. 21. From the figure, we find that the majority of the spots lies in the upper left region, and if there are some spots located on the bottom right region, they are near the diagonal. This implies that DP-Stream is a robust and promising data stream clustering method.

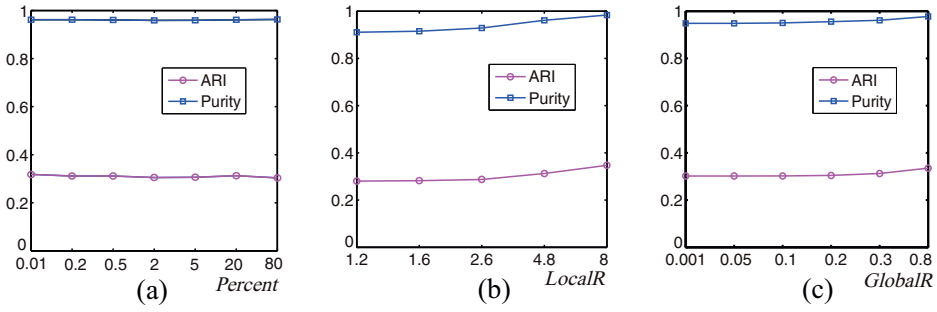
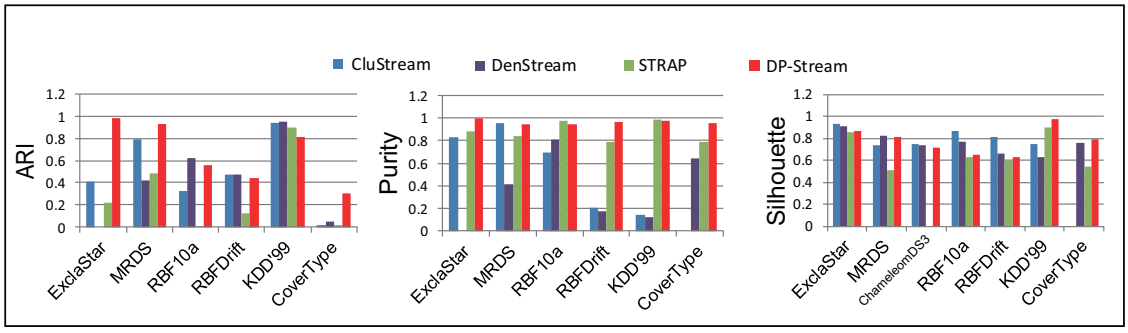
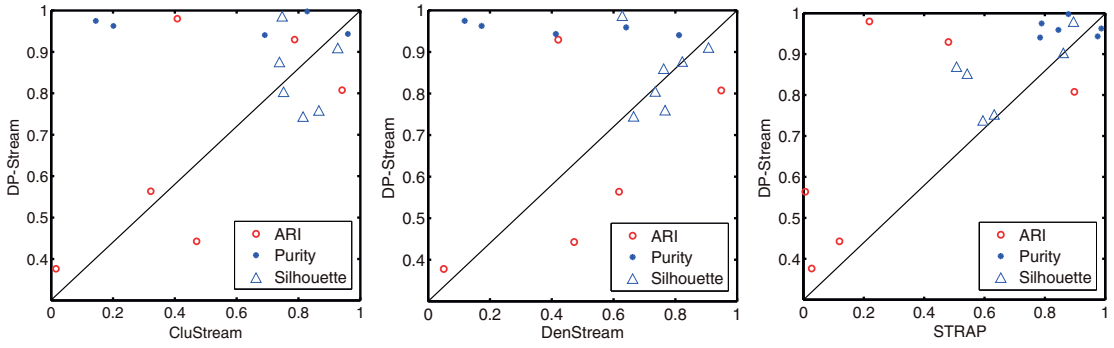


Fig. 20. Parameter sensitivity analysis on CoverType. (a) Fix parameters *LocalR* = 4.8 and *GlobalR* = 0.3, then vary *Percent*. (b) Fix parameters *Percent* = 20 and *GlobalR* = 0.3, then vary *LocalR*. (c) Fix parameters *Percent*=20 and *LocalR* = 4.8, then vary *GlobalR*.



(a) Accuracy comparisons over different datasets



(b) Win-Loss diagram when compared to the three competing models

Fig. 21. Overall accuracy comparison. (a) ARI, Purity and Silhouette Index of the four comparative methods on different datasets. (b) The win-lose scores of DP-Stream vs. CluStream, DenStream, and STRAP are 10:7, 9:7, and 14:3, respectively.

DP-Stream borrows quite a few ideas from the existing stream clusterings including microclusters, fading out, sliding window, and change detection. But it also has its own unique features. The fat nodes of the FNLT is approximately equivalent to the microclusters in many previous methods such as CluStream, DenStream, and DBSTREAM. However, one major contribution of DP-Stream is that the partial order relation (see Definition 2) between any pairs of the fat nodes essentially enables the fast delivery of the clustering result for the newly arrived objects. Also, efficient update of this partial order relation plays an important role in getting rid of the "online-offline" paradigm and in directly maintaining full clustering on the fly.

Another salient feature of DP-Stream different from the existing works is the reservoir. It may be compared to the "window" in CluStream, STRAP, etc., but usually it has much smaller size. The underlying reason is that CluStream and STRAP need a relatively larger number of objects in the window to find the clusters of this moment, while DP-Stream has an easy-to-update FNLT to incorporate and label the new objects. The reservoir of DP-Stream is used to decide when the FNLT with new individual objects needs to be granulated into another FNLT, and when the history objects need to decay in a batch fashion. That is, when the number of new objects grows up to the size of the reservoir, the operations of granulating and fading out are triggered. In the experiments, we find that appropriately reducing the size of reservoir would accelerate the processing with the accuracy being guaranteed.

6.5.2. Running time comparison

The running time of the four competing data stream clustering methods on the 7 datastes is shown in Fig. 22. The overall time complexities of DP-Stream and other competing methods are all quadratic w.r.t. the size of reservoir, so their complexities have the same big *O* notations. The difference in time consumption therefore depends only on how the algorithms process a data stream with specific characteristic. DP-Stream has the virtue of assigning cluster label to a non-strange item right after its leading node is

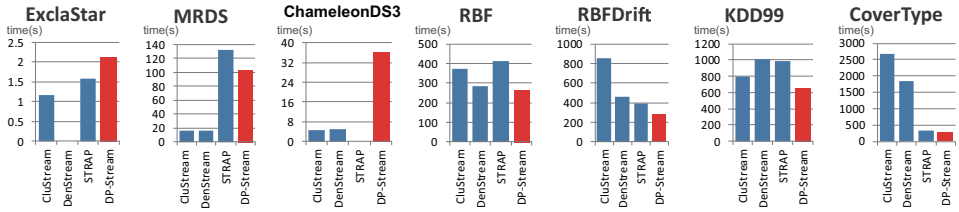


Fig. 22. Running time comparison. The bars for DenStream on ExclaStar and STRAP on Chammeleon are absent because the corresponding clusterings are not available.

decided. However, if a new item is recognized as strange, then its destination (existing cluster, new cluster, or outlier) will be postponed until more items arrive or even until the reservoir is full (refer to Fig. 6 for an illustration). So the more strange items DP-Stream encounters, the longer its processing time lasts. Among the 7 datasets, the 3 2D datasets have more strange items and the other 4 datasets of higher dimensionality have relatively less strange items. Therefore, DP-Stream exhibits lower efficiency on the first 3 datasets and higher efficiency on the last 4 datasets than the baselines.

7. Conclusions

In this paper, we developed the first data stream clustering algorithm which uses the density peaks to find any shape of clusters in data streams and permanently delivers the clustering result for the new items in linear time complexity. It gets rid of the "online-offline" paradigm with an ever evolving fat node leading tree. The process of updating the FNLT is also the process of generating the clustering result simultaneously. The FNLT has stable amount of fat nodes with the help of fading out mechanism, and some weak nodes are removed to emphasize on the recent data. The DP-Stream also has the capability of detecting concept drifts and filtering outliers. The experiments have shown the efficiency and accuracy of our approach on synthetic datasets and real world datasets, which verifies and demonstrates the result of theoretical analysis.

As demonstrated in the experiments on ChameleonDS3 and KDD'99, there are still spaces for DP-Stream to be improved. When the density peaks within a cluster are more than one, or in other words, the data distribution in a cluster is not smooth, DPClust tends to cluster them as many clusters, which may violate human intuition and the ground truth. Another case is when the number of clusters varies greatly in the evolving stream, the static *dc* policy has difficulty to find the right number of clusters. Therefore, we plan to address the issues in the future. In addition, the FNLT-based drift detection method introduced in this paper is new, and many aspects of this method still need to be explored, especially its performance evaluated with the corresponding measures such as false alarm, missing and delay.

Acknowledgments

This work has been supported by the National Key Research and Development Program of China under grant 2016YFB1000905, the National Natural Science Foundation of China under grants 61272060 and 61572091. The authors thank Dr. Xiangliang Zhang for providing the STRAP program and anonymous reviewers for their help in improving the manuscript.

References

[1] J. Gama, M.M. Gaber, Learning from data streams: processing techniques in sensor networks, Springer 25 (1) (2007) 1–4.
[2] R. Kohavi, Mining e-commerce data: the good, the bad, and the ugly, in: Knowledge Discovery and Data Mining, 2001, pp. 8–13.
[3] C.C. Aggarwal, J. Han, J. Wang, P.S. Yu, A framework for projected clustering of high dimensional data streams, in: Proceedings of the Thirtieth International Conference on Very Large Data Bases-Volume 30, VLDB Endowment, 2004, pp. 852–863.
[4] C.C. Aggarwal, C.K. Reddy, Data Clustering : Algorithms and Applications, CRC Press, 2014.
[5] L.O. Callaghan, N. Mishra, A. Meyerson, S. Guha, R. Motwani, Streaming-data algorithms for high-quality clustering, in: Proceedings of 18th International Conference on Data Engineering, 2002, pp. 685–694.
[6] F. Cao, M. Ester, W. Qian, A. Zhou, Density-based clustering over an evolving data stream with noise., in: In SIAM Conference on Data Mining, 2006, pp. 328–339.
[7] L. Wan, W.K. Ng, X.H. Dang, P.S. Yu, K. Zhang, Density-based clustering of data streams at multiple resolutions, ACM Trans. Knowl. Discov. Data 3 (3) (2009) 49–50.
[8] L. Tu, Y. Chen, Stream data clustering based on grid density and attraction, ACM Trans. Knowl. Discov. Data 3 (3) (2009). UNSP 12.
[9] X. Zhang, C. Furtlehner, C. Germain-Renaud, M. Sebag, Data stream clustering with affinity propagation, IEEE Trans. Knowl. Data Eng. 26 (7) (2014) 1644–1656.
[10] E. Lughofer, M. Sayed-Mouchaweh, Autonomous data stream clustering implementing split-and-merge concepts–towards a plug-and-play approach, Inf. Sci. 304 (2015) 54–79.
[11] M. Hahsler, M. Bolanos, Clustering data streams based on shared density between micro-clusters, IEEE Trans. Knowl. Data Eng. 28 (6) (2016) 1449–1461.
[12] S.-S. Ho, H. Wechsler, A martingale framework for detecting changes in data streams by testing exchangeability, IEEE Trans. Pattern Anal. Mach. Intell. 32 (12) (2010) 2113–2127.
[13] A. Rodriguez, A. Laio, Clustering by fast search and find of density peaks, Science 344 (6191) (2014) 1492–1496.
[14] M.M. Gaber, A. Zaslavsky, S. Krishnaswamy, Mining data streams: a review, ACM Sigmod Record 34 (2) (2005) 18–26.
[15] C. Heinz, B. Seeger, Cluster kernels: resource-aware kernel density estimators over streaming data, IEEE Trans. Knowl. Data Eng. 20 (7) (2008) 880–893.
[16] C.C. Aggarwal, P.S. Yu, J. Han, J. Wang, A framework for clustering evolving data streams, Vldb 29 (2003) 81–92.
[17] M. Ester, H. Kriegel, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, Knowl. Discov. Data Mining (1996) 226–231.
[18] B.J. Frey, D. Dueck, Clustering by passing messages between data points., Science 315 (5814) (2007) 972–976.
[19] M.R. Ackermann, M. Märtens, C. Raupach, K. Swierkot, C. Lammersen, C. Sohler, Streamkm++: a clustering algorithm for data streams, J. Exp. Algorithmics 17 (2012) 173–187.
[20] E. Lughofer, Extensions of vector quantization for incremental clustering, Pattern Recognit. 41 (3) (2008) 995–1011.
[21] M.Z.-u. Rehman, T. Li, Y. Yang, H. Wang, Hyper-ellipsoidal clustering technique for evolving data stream, Knowl. Based Syst. 70 (2014) 3–14.
[22] Q. Tu, J.F. Lu, B. Yuan, J.B. Tang, J.Y. Yang, Density-based hierarchical clustering for streaming data, Pattern Recognit. Lett. 33 (5) (2012) 641–645.
[23] P.P. Rodrigues, J. Gama, J.P. Pedroso, Hierarchical clustering of time-series data streams, IEEE Trans. Knowl. Data Eng. 20 (5) (2008) 615–627.
[24] K. Koshijima, H. Hino, N. Murata, Change-point detection in a sequence of bags-of-data, IEEE Trans. Knowl. Data Eng. 27 (10) (2015) 2632–2644.
[25] A. Hinneburg, D.A. Keim, A general approach to clustering in large databases with noise, Knowl. Inf. Syst. 5 (4) (2003) 387–415.
[26] A. Hinneburg, H.-H. Gabriel, Denclue 2.0: Fast clustering based on kernel density estimation, in: Advances in Intelligent Data Analysis VII, Springer, 2007, pp. 70–80.
[27] J. Xu, G. Wang, W. Deng, DenPEHC: density peak based efficient hierarchical clustering, Inf. Sci. 373 (2016) 200–218.
[28] V.J. Hodge, J. Austin, A survey of outlier detection methodologies, Artif. Intell. Rev. 22 (2) (2004) 85–126.
[29] M. Radovanovic, A. Nanopoulos, M. Ivanovic, Reverse nearest neighbors in unsupervised distance-based outlier detection, IEEE Trans. Knowl. Data Eng. 27 (5) (2015) 1369–1382.
[30] J. Huang, Q. Zhu, L. Yang, J. Feng, A non-parameter outlier detection algorithm based on natural neighbor, Knowl. Based Syst.. 92 (2016) 71–77.
[31] M.E. Basseville, I.V. Nikiforov, Detection of Abrupt Changes: Theory and Application, Prentice Hall, 1993.

[32] L. Hubert, P. Arabie, Comparing partitions, J. classification 2 (1) (1985) 193–218.
[33] N.X. Vinh, J. Epps, J. Bailey, Information theoretic measures for clusterings comparison: is a correction for chance necessary? in: Proceedings of the 26th Annual International Conference on Machine Learning, ACM, 2009, pp. 1073–1080.
[34] P.J. Rousseeuw, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, J. Comput. Appl. Math. 20 (1987) 53–65.
[35] G. Karypis, E.-H. Han, V. Kumar, Chameleon: hierarchical clustering using dynamic modeling, Computer 32 (8) (1999) 68–75.
[36] X.-F. Wang, Y. Xu, Fast clustering using adaptive density peak detection, Stat. Methods Med. Res. 0 (0) (2015) 1–14.