Clustering Stream Data by Exploring the Evolution of Density Mountain

Shufeng Gong[1,3], Yanfeng Zhang[1,3], Ge Yu[1,2,3]
[1]Northeastern University, [2]Liaoning University
[3]Key laboratory of Medical Image Computing (Northeastern University), Ministry of Education
Shenyang, China
gongsf@stumail.neu.edu.cn, {zhangyf, yuge}@mail.neu.edu.cn

ABSTRACT

Stream clustering is a fundamental problem in many streaming data analysis applications. Comparing to classical batch-mode clustering, there are two key challenges in stream clustering: (i) Given that input data are changing continuously, how to incrementally update their clustering results efficiently? (ii) Given that clusters continuously evolve with the evolution of data, how to capture the cluster evolution activities? Unfortunately, most of existing stream clustering algorithms can neither update the cluster result in real-time nor track the evolution of clusters.

In this paper, we propose a stream clustering algorithm *EDMStream* by exploring the **E**volution of **D**ensity **M**ountain. The *density mountain* is used to abstract the data distribution, the changes of which indicate data distribution evolution. We track the evolution of clusters by monitoring the changes of density mountains. We further provide efficient data structures and filtering schemes to ensure that the update of density mountains is in real-time, which makes online clustering possible. The experimental results on synthetic and real datasets show that, comparing to the state-of-the-art stream clustering algorithms, e.g., D-Stream, DenStream, DBSTREAM and MR-Stream, our algorithm is able to response to a cluster update much faster (say 7-15x faster than the best of the competitors) and at the same time achieve comparable cluster quality. Furthermore, *EDMStream* successfully captures the cluster evolution activities.

PVLDB Reference Format:
Shufeng Gong, Yanfeng Zhang, Ge Yu. Clustering Stream Data by Exploring the Evolution of Density Mountain. *PVLDB*, 11(4): 393-405, 2017.
DOI: 10.1145/3164135.3164136

1. INTRODUCTION

Recent advances in both hardware and software have resulted in a large amount of data, such as sensor data, stock transition data, news, tweets and network flow data etc. A kind of such data that continuously and rapidly grow over time are referred to as data streams [2]. Clustering stream data is one of the most fundamental problems in many streaming data analysis applications. Basically, it groups streaming data on the basis of their similarity, where data evolves over time and arrives in an unbounded stream.

Discovering of the patterns hidden in streams is substantial and essential for understanding and further utilizing these data, and there are large number of efforts contributing it, such as [4, 6, 14]. Take the news recommendation system as an example. The news recommendation system aims to present the news that will interest users. The news are clustered according to their similarities, so that the news in the same cluster as that a user has visited is recommended to the user. As the news are generated continuously, the news can be treated as a news stream. Furthermore, the news clusters are evolving as fresh news coming out and outdated news fading out. In order to make a timely recommendation, it is crucial to capture the cluster evolution and update the news clusters in real-time. Stream clustering is widely used in a broad range of important applications, such as network intrusion detection, weather monitoring, web site analysis, and attracts many research efforts [3, 5, 7, 15, 26, 32].

Comparing to classical batch-mode clustering methods [25, 12, 33], there are two additional key challenges in stream clustering. First, stream data are supposed to arrive in a high speed. In order to reflect changes of the underlying stream data, stream clustering algorithms are required to *update clustering results quickly and frequently.* Second, multiple clusters might merge into a large cluster, and a single cluster might be split into multiple small clusters over time. In order to capture the cluster evolution activities, stream clustering algorithms are required to *have the ability of tracking cluster evolution.*

For the first challenge, most existing solutions [5, 7] summarize data points in stream using summary structures (e.g., micro-clusters [3, 5], grids[7]) and update these summaries upon receiving new points. Using the summary structures can reduce processing overhead. Then an offline batch-mode clustering algorithm is periodically performed on these summaries to update the clustering result. However, these stream clustering algorithms do not support incremental update and are still very expensive to offline update clustering results. For the second challenge, they leverage an additional offline cluster evolution detecting procedure (e.g., MONIC [27] and MEC [20]). Due to the expensive offline clustering and offline tracking step, the existing solutions can neither update the clustering result in real-time nor monitor the evolution of clusters in real-

Proceedings of the VLDB Endowment, Vol. 11, No. 4

DOI: 10.1145/3164135.3164136

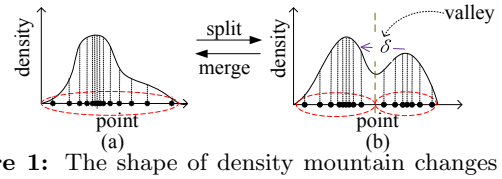


Figure 1: The shape of density mountain changes as the (1-dimension) data distribution evolves.

time. Thus, a stream clustering algorithm that can update clustering result and monitor the cluster evolution in real-time is desired.

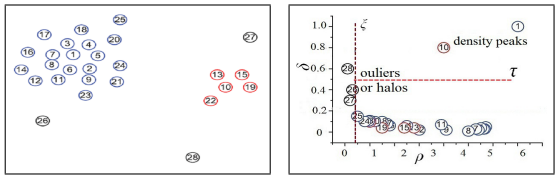
In this paper, we propose a density-based stream clustering algorithm *EDMStream*. We rely on the first assumption that cluster centers are surrounded by neighbors with lower local density[1]. Then we can draw the density distribution of data points as shown in Fig. 1(a), which refers to *density mountain*. The cluster center is at the mountain peak and the borderline points are at the mountain foot. Note that this is an illustrative figure and the points are in a *1-dimension* space. In general, the density mountain should be drawn in a multi-dimensional plot. We rely on the second assumption that the center point has a relatively large distance from any other higher density points. As shown in Fig. 1(b), there are two clusters corresponding to two density mountains, and there is a valley between two mountains. The right density mountain's peak has a relatively large distance to the higher density points, since the higher density points are located on left (higher) density mountain, while other points on the way up to the density peak are with relatively small distance to the higher density points. As a result, a wide density valley appears between two density mountains, and the nearest distance to higher density point (labeled as δ in Fig. 1(b)) plays a key role in identifying clusters. The cluster evolution of data stream can be detected as long as the distance to the nearest higher density point is large enough or small enough.

To quickly update clustering result, we first summarize a set of close points as a *cluster-cell* to reduce computation and maintenance cost. We then propose an efficient hierarchical tree-like structure *Dependency-Tree (DP-Tree)* to abstract density mountains. The DP-Tree maintains the relationships between cluster-cells (i.e., a cluster-cell and its nearest higher density cluster-cell) and implies the relationship between clusters. Our algorithm can quickly return update clustering results by efficiently updating the DP-Tree structure. Meanwhile, by tracking the update of the DP-Tree structure, we can also track the cluster evolution activities.

Another feature that distinguishes our algorithm from other existing solutions is the ability of adjusting itself and adapting to data distribution changes. Through a user-interaction step in the initialization phase, it can learn user preferences for cluster granularity. In terms of the user preference, it can dynamically adjust the cluster separation strategy for the evolving data stream. This can greatly improve the quality of clustering results as shown in our experimental results.

The contributions of this paper are summarized as follows.

- **(Effectiveness on Cluster Evolution Tracking)** We propose a novel cluster evolution detection strategy by monitoring density mountains. Comparing to existing solutions, it can not only achieve comparable cluster quality but also tell how the clusters evolve.
- **(Efficiency)** We design a highly efficient data structure Dependency-Tree to maintain the states of density mountains. By using two filtering strategies, a large amount of unnecessary tree update operations are avoided. It improves the performance a lot for cluster result updates and helps monitor cluster evolution efficiently.
- **(Adaptability)** We provide an automatic adjusting strategy. It learns the user preference through an initial user-interaction step and automatically updates the algorithm parameters according to data evolution, which makes our clustering algorithm self-adaptive to data distribution changes.
- We perform extensive experiments to evaluate *EDMStream*. We compare *EDMStream* with the state-of-art stream clustering algorithms, including D-Stream [7], DenStream [5], DBSTREAM [13] and MR-Stream [30]. The results show that *EDMStream* outperforms these algorithms on both effectiveness and efficiency. *EDMStream* only takes 7-23μs for each cluster update on a commodity PC. It exhibits 7-15x speedup over the best of competitors. We also demonstrate its adaptability to automatically adjust key parameters according to data evolution. Additionally, we also introduce a news recommendation use case and show its ability to track cluster evolution.



(a) Plane view (b) Decision graph

Figure 2: Density Peaks Clustering [25]

The remainder of this paper is organized as follows. Sec. 2 reviews the Density Peaks Clustering and proposes our Dependency-Tree structure. Sec. 3 presents our definitions and related concepts of stream clustering. In Sec. 4, we introduce *EDMStream* algorithm. Experimental results are shown in Sec. 5. Sec. 6 reviews the related work and Sec. 7 gives conclusion.

2. DP CLUSTERING AND DP-TREE

2.1 DP Clustering

Density Peaks (***DP***) Clustering [25] is a novel clustering algorithm recently proposed by Rodriguez and Laio [25]. The algorithm is based on two observations: (i) cluster centers are often surrounded by neighbors with lower local densities, and (ii) they are at a relatively large distance from any points with higher local densities. Correspondingly, DP computes two metrics for every data point: (i) its *local density* ρ and (ii) its distance δ from other points with higher density. DP uses the two metrics to locate density peaks, which are the cluster centers.

The ***local density*** ρ_i of data point p_i is the number of points whose distance to p_i is smaller than d_c.

$$\rho_i = |\{p_j | |p_i, p_j| < d_c\}| \quad (1)$$

[1]Points with lower local density means that these points are in a low density region.

where $|p_i, p_j|$ is the distance[2] from point p_i to point p_j, and d_c is called the cutoff distance. We use the density value to distinguish ***outliers*** whose density is no bigger than a predefined value ξ (i.e., $\rho_i \leq \xi$).

The ***dependent distance*** δ_i of point p_i is computed as

$$\delta_i = \min_{j:\rho_j > \rho_i} (|p_i, p_j|) \quad (2)$$

It is the minimum distance from point p_i to any other point whose local density is higher than that of point p_i. Suppose point p_j is point p_i's the nearest neighbor[3] with higher density, i.e., $p_j = argmin_{j:\rho_j > \rho_i}(|p_i, p_j|)$. We say that point p_i is ***dependent*** on point p_j.

Let us think more about the local density ρ and dependent distance δ. A point with small ρ_i is likely to be an outlier no matter how large its δ_i is. Next, we focus on the points with relatively large ρ_i to study the effect of δ_i. Small δ_i implies that point p_i is surrounded by at least one higher density neighbor. Anomalously large δ_i implies that point p_i is far from another dense area and point p_i itself could be the density peak of its own region, since it has no higher density neighbor. δ_i is much larger than the typical nearest neighbor distance only for points that are local or global maxima in the density. Thus, cluster centers are recognized as points for which the value of δ_i is anomalously large as well as large ρ_i. This is also illustrated in Fig. 1, where the points in the same density mountain have relatively small dependent distance except for the density peak (i.e., cluster center).

If the dependent distance from a point p_i to its dependency p_j is not bigger than τ (i.e., $\delta_i \leq \tau$), we say it is ***strongly dependent***, otherwise it is ***weakly dependent***. For a set of points $\{p_1, p_2, \ldots, p_n\}$, there exists a strongly dependent chain such that point p_i $(1 \leq i \leq n-1)$ is strongly dependent on p_{i+1}, where the end point p_n is not strongly dependent on any other point (might be weakly dependent on other point). We call point p_n as any p_i's ***strongly dependent root***. Point p_i $(1 \leq i \leq j-1)$ is ***dependency-reachable to*** any p_j $(i+1 \leq j \leq n)$. Then, a cluster in DP algorithm can be defined as follows:

Definition *1. (**Cluster**) Let P be a set of points. A cluster C is a non-empty subset of P such that:*

- *(Maximality) If a point $p \in C$, then any non-outlier point q that is dependency-reachable to p also belongs to C.*
- *(Traceability) For any points $p_1, p_2, \ldots \in C$, they have the same strongly dependent root, which is the density peak in C.*

Fig. 2 illustrates the process of Density Peaks Clustering through a concrete example. Fig. 2a shows the distribution of a set of 2-D data points. Each point p_i is depicted on a *decision graph* by using (ρ_i, δ_i) as its x-y coordinate as shown in Fig. 2b. By observing the decision graph, the *density peaks* can be identified in the top right region since they are with relatively large ρ_i and large δ_i (i.e., $\rho_i > \xi$ and $\delta_i > \tau$). The *outliers* or *halos*[4] can be identified in the left region whose $\rho_i \leq \xi$. Since each point is only dependent on a single point, for each point there is a dependent chain ending at a density peak. After the density peaks (as cluster representatives) have been found, each remaining point is assigned to the same cluster as its dependent point.

[2]In this paper, the distance means the Euclidean distance unless particularly mentioned.

[3]If multiple higher density nearest neighbors' distances are equal, we randomly pick one among them.

[4]The cluster halos are the points that locate at the borders of clusters.

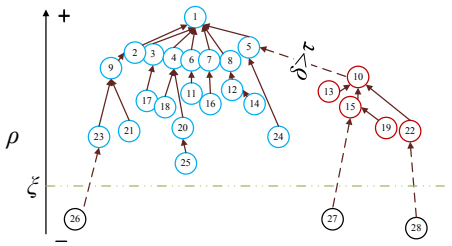


Figure 3: An illustrative example of DP-Tree. The dependency relationship is captured by setting arrows between points. The length of arrows indicates the dependent distance (δ). The solid arrows indicate strongly dependent relationship. The dashed arrows indicate weakly dependent relationship. The points residing at higher levels of DP-Tree are with higher densities (ρ). The root of DP-Tree is the absolute density peak with the highest local density.

2.2 Dependency Tree (DP-Tree)

To abstract the DP clustering, we propose a tree-like structure, *Dependency Tree*, which can track the correlations between points and between clusters. As mentioned in Sec. 2.1, the clustering process is achieved by tracking the dependency chain. The point-point dependency relationship implies the point-cluster correlations. In order to support online stream processing, an efficient data structure is desired to maintain the dependency relationship. Since each point is only dependent on a single point (except for the absolute density peak with the highest density), the point-point dependencies can be abstracted by a tree-like structure, which is denoted as *Dependency Tree* (***DP-Tree***). Fig. 3 shows an illustrative DP-Tree for the points shown in Fig. 2a.

Let us first divide the DP-Tree into two parts, i.e., the upper part (in which each node's density is larger than ξ) and the lower part (in which each node's density is smaller than or equal to ξ). The nodes that belong to the lower part are simply recognized as outliers. In the upper part of DP-Tree, for a given subtree T_i in DP-Tree, if all the links in the subtree are strongly dependent, T_i is a ***strongly dependent subtree***. If there is no other strongly dependent subtree T_j such that T_i is a subtree of T_j, we say T_i is a ***Maximal Strongly Dependent SubTree (MSDSubTree)***. Given the definition of MSDSubTree and the definition of cluster in Def. 1, the clustering based on DP-Tree is defined as follows.

Definition *2. (**Clustering based on DP-Tree**) The clustering based on DP-Tree is to find all the MSDSubTrees. Each MSDSubTree corresponds to a cluster. The root of a MSDSubTree is the cluster center of that cluster.*

The DP-Tree structure is highly efficient for maintaining volatile clusters. It can quickly response to a cluster update query and can be used for tracking cluster evolutions by its *hierarchical* structure. Once a new point arrives, it is directly linked to its dependent point in terms of its local density. More importantly, the new point may affect

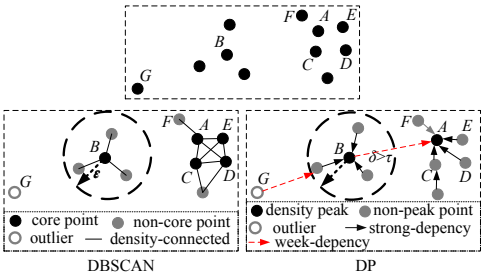


Figure 4: DBSCAN vs DP Clustering

its nearby points and change their cluster assignments. Many affected points can share the same predecessor, and they belong to the same cluster as their predecessor. We only need to change a pointer (the predecessor's pointer to another MSDSubTree) to complete this update, which greatly saves the maintaining cost.

2.3 DP Clustering vs. DBSCAN

Since the idea of DP clustering is pretty similar to that of DBSCAN, it is necessary to highlight the difference between them. The cluster defined in DBSCAN satisfies two criteria: *maximality* and *connectivity* [9, 10], while the cluster defined in DP clustering satisfies *maximality* and *traceability* (Def. 1). The maximality defined in both the two algorithms depicts the "reachable" property between two points, where the reachable property in both algorithms relies on point density information. However, the connectivity in DBSCAN depicts the density-connected property which is *symmetric*, while the traceability in DP depicts the density-dependent property which is *non-symmetric*. Therefore, in DBSCAN, the density-connected relationship between points can be abstracted as an undirected graph, and each connected component of the graph constitutes a preliminary cluster [11]. The clustering in DBSCAN is to find all the connected components from the density-connected graph. While in DP clustering, the density-dependent relationship between points can be abstracted with a tree-like structure (DP-Tree), and each MSDSubTree in the DP-Tree constitutes a cluster. The clustering in DP is to find all the MSDSubTrees from the DP-Tree.

Fig. 4 provides an example to demonstrate the difference between DBSCAN and DP. The data distribution is shown on the upper side of the figure. DBSCAN first identifies the high density points as core points (e.g., points *A*-*E* are core points because their densities are higher than a threshold), and connects these core points if they are closer than a threshold ε (e.g., points *A, C, D, E* are connected with each other, and point *B* is not connected to the other core points because *B* is far away from them). For each non-core point, it is connected to only one core point if they are close enough (e.g., point *F*), otherwise is treated as an outlier (e.g., point *G*). DBSCAN constructs such an undirected graph and aims to find all the connected components, each connected component corresponding to a cluster.

DP creates dependency connection from each point to its nearest neighbor with higher density (e.g., *B*-*F* all depends on *A* since *A* is their nearest higher density neighbor). Different from DBSCAN's density-connected undirected graph, in DP each point depends only on a single point, and the dependency connection is directed. Thus, DP constructs a dependency tree structure (DP-Tree) rather than an undirected graph. In addition, DP distinguishes the weakly dependency connections whose distances are longer than a threshold τ (e.g., the dependency connection from *B* to *A*), and uses these weakly dependent connections to separate the DP-Tree into multiple subtrees (e.g., the subtree rooted from *B* is separated from the subtree rooted from *A* because their connection is weakly dependent). Each MSDSubTree corresponds to a cluster, and the root of each subtree is recognized as the density peak (e.g., points *A* and *B*). The points with extremely low density are recognized as outliers (e.g., point *G*).

3. PROBLEM STATEMENT

We aim to discover the potential clusters existing in data stream based on the two observations. 1) Dense areas are separated from each other by sparse areas; 2) Recent arrival data points play more important role in cluster representation than outdated data points. In this section, in terms of timeliness and unlimitedness of data streams, we introduce the basic conceptions that will be used in stream clustering.

3.1 Basic Conceptions

Data Stream. A data stream S is a sequence of data points with timestamp information $p_1^{t_1}$, $p_2^{t_2}$, ..., $p_N^{t_N}$, i.e., $S^N = \{p_i^{t_i}\}_{i=1}^N$, which is potentially unbounded ($N \to \infty$). Each data point is described by a d-dimensional attribute vector with its arrival timestamp t_i.

Decay Model. In most cases, the recent information from a stream reflects the emerging of new trends, e.g., weather monitoring and stock trade. The importance(freshness) of data should be decayed over time, so that the evolving characteristics of the stream can be captured. A common solution is to weight data points with an exponential time-dependent decay function [7, 5, 16]. The freshness of point i at time t is defined as

$$f_i^t = a^{\lambda(t-t_i)}. \tag{3}$$

This is a widely used decay function in many stream clustering algorithms [5, 15, 7]. The parameter a and λ control the form of decay function. The higher the absolute value of λ is, the faster the algorithm "forgets" old data. In this paper, we choose $a = 0.998$, $\lambda = 1$ such that f_i^t is in the range $(0, 1]$. Suppose $\{p_j | t_j < t, |p_i, p_j| < d_c\}$ is a set of existing points whose distances to p_i are smaller than d_c before time point t. Point p_i's local density at time t is the sum of nearby points' f_j^t rather than the number of nearby points as depicted in Equation (1).

$$\rho_i^t = \sum_{p_j : t_j < t, |p_i, p_j| < d_c} f_j^t \tag{4}$$

The decay model implies that 1) if no new nearby point arrives the point density is decreasing over time and 2) all points in a stream are decaying at the same pace. In other words, we have a decay function $\mathcal{D}^t()$ applied on the current stream S^n at any time t to decay the points.

$$\mathcal{D}^t(S^n) = \{f_1^t, f_2^t, \ldots, f_n^t\} \tag{5}$$

Stream Clustering. Under the decay model, stream clustering is defined as follows.

***Definition** 3. (**Stream Clustering**)* Given a data stream S^N and their decayed freshness $\mathcal{D}^t(S^N)$, stream clustering $\mathcal{C}^t()$ returns a set of disjoint clusters at any time t_1, t_2, ..., t_N. That is, for any n ($1 \le n \le N$), we have

$\mathcal{C}^{t_n}\big(S^n, \mathcal{D}^{t_n}(S^n)\big) = \{C_1^{t_n}, C_2^{t_n}, \ldots, C_{k^{t_n}}^{t_n}, C_o^{t_n}\}$, where $C_i^{t_n}$ $(1 \le i \le k^{t_n})$ is a subset of S^n at time t_n, $C_o^{t_n}$ is the set of outliers at time t_n, k^{t_n} is the number of clusters at time t_n, $S^n = C_1^{t_n} \cup C_2^{t_n} \cup \ldots \cup C_{k^{t_n}}^{t_n} \cup C_o^{t_n}$, $C_i^{t_n} \cap C_j^{t_n} = \emptyset$, and $C_i^{t_n} \cap C_o^{t_n} = \emptyset$ for any i and j.

Cluster Evolution. Clusters evolve continuously, i.e., $\mathcal{C}^{t_n}\big(S^n, \mathcal{D}^{t_n}(S^n)\big) \neq \mathcal{C}^{t_{n+1}}\big(S^{n+1}, \mathcal{D}^{t_{n+1}}(S^{n+1})\big)$. Specifically, the number of clusters may change, and the point-to-cluster assignment may change. By referring to the previous work [27, 20], we define five types of evolutions which are summarized in Table 1.

Table 1: Cluster evolution types

Type	Mathematical Notation
Emerge	$\emptyset \to C_i^{t_{n+1}}$
Disappear	$C_i^{t_n} \to \emptyset$
Split	$C_i^{t_n} \to \{C_{i_1}^{t_{n+1}}, \ldots, C_{i_x}^{t_{n+1}}\}$
Merge	$\{C_{i_1}^{t_n}, \ldots, C_{i_x}^{t_n}\} \to C_i^{t_{n+1}}$
Adjust	$C_i^{t_n} \to C_i^{t_{n+1}}$, $C_j^{t_n} \to C_j^{t_{n+1}}$ 1. $C_i^{t_{n+1}} = C_i^{t_n} \backslash \{p_1, \ldots, p_l\}$, $C_j^{t_{n+1}} = C_j^{t_n} \cup \{p_1, \ldots, p_l\}$ 2. $C_o^{t_{n+1}} = C_o^{t_n} \backslash \{p_1, \ldots, p_l\}$, $C_i^{t_{n+1}} = C_i^{t_n} \cup \{p_1, \ldots, p_l\}$ 3. $C_i^{t_{n+1}} = C_i^{t_n} \backslash \{p_1, \ldots, p_l\}$, $C_o^{t_{n+1}} = C_o^{t_n} \cup \{p_1, \ldots, p_l\}$

The ***emerge*** evolution means a new cluster's birth. The ***disappear*** evolution means an old cluster's death. The ***split*** evolution means that a cluster is split into two or more clusters. The ***merge*** evolution means that two or more clusters merge into one cluster. The ***adjust*** evolution happens when 1) some points move from one cluster to another cluster; 2) some outliers become denser and are merged to a cluster; 3) some marginal points in a cluster become outliers. The first four types will change the number of clusters, while the last one only changes the point-to-cluster assignments. Note that, the first four evolutions might occur along with cluster adjustment, and the three kinds of adjustment might occur concurrently.

3.2 Stream Data Summarization

If stream data are massive or even unlimited, it is not possible to store all data in main memory. Therefore, it is necessary to summarize stream data in an efficient way. We summarize a set of close points as a ***cluster-cell*** so as to reduce the memory/computation cost. The cluster-cell is formally defined as follows.

Definition 4. (cluster-cell) *A cluster-cell c summarizes a group of close points, which can be described by a three-tuple $\{s_c, \rho_c^t, \delta_c^t\}$ at time t.*

- *s_c is the* **seed point** *of a cluster-cell c. The cluster-cell c seeded by s_c summarizes a set of points whose distance to s_c is less than the distance to any other seed point and is less than or equal to a predefined radius r, i.e., $P_c = \{p_i : s_c = \arg\min_{s_k \in S_{seed}}(|p_i, s_k|), |p_i, s_c| \le r\}$ where S_{seed} is the set containing all seed points.*
- *ρ_c^t is the summarization of all cluster-cell points'* **timely density** *(abbrv. density) at time t, which is defined as follows.*

$$\rho_c^t = \sum_{p_i \in P_c} f_i^t. \tag{6}$$

where f_i^t is the freshness of p_i at time t defined in Equation (3).

- *δ_c^t is the* **dependent distance** *from s_c to its nearest cluster-cell seed point with higher cluster-cell density. Similar to Equation (2), δ_c^t is defined as follows.*

$$\delta_c^t = \min_{c': \rho_{c'}^t > \rho_c^t}(|s_c, s_{c'}|). \tag{7}$$

For processing, we will take cluster-cells as the basic unit instead of points. In other words, we will operate on a DP-Tree where each node is a cluster-cell instead of a data point. A new arrival point is not directly inserted to the DP-Tree but used to generate a new cluster-cell or increasing an existing cluster-cell's density. Both cases can lead to DP-Tree's update. On the other hand, the decaying of points will lead to the decaying of cluster-cells, which can also lead to DP-Tree's update. By using cluster-cell, we can approximately obtain the timely density of local regions and significantly reduce the memory/computation cost.

3.3 Basic Ideas

Stream Clustering using DP-Tree. In the context of DP-Tree, stream clustering is simply to find all MSDSubTrees from a ***dynamic*** DP-Tree. The DP-Tree is dynamic since new arrival points and decay model may cause tree structure's update.

Evolution Tracking using DP-Tree. In addition, cluster evolution can be tracked by monitoring how the DP-Tree changes. 1)Cluster *emergence/disappearance* can be tracked by finding new generated/disappeared MSDSubTrees; 2)Cluster *split* can be tracked when an MSDSubTree is split into multiple MSDSubTrees (one or more dependent links become longer than τ); 3)Cluster *merging* can be tracked when multiple MSDSubTrees merge into one MSDSubTree (one or more dependent links become shorter than τ). Cluster adjustment can be tracked by that: 1) multiple cluster-cells from an MSDSubTree are relinked to other MSDSubTrees; 2) multiple cluster-cells' densities become larger than ξ and they are included in the MSDSubTrees that have their dependencies; 3) multiple cluster-cells' densities becomes smaller than ξ and they are removed from their original MSDSubTrees.

Our Goal. To sum up, we aim to design an algorithm that can efficiently maintain and monitor the dynamic DP-Tree and return the MSDSubTrees quickly upon any change.

4. *EDMSTREAM*

In this section we propose *EDMStream* for clustering streaming data, and explain it in detail.

4.1 Algorithm Overview

EDMStream distinguishes itself from other existing stream clustering algorithms on the ability of updating clusters in real-time and tracking cluster evolution. We briefly overview the *EDMStream* algorithm in the following.

Storage Structures. As shown in Fig. 5, two key storage structures are designed in *EDMStream*.

1) DP-Tree. *DP-Tree* is the data structure for abstracting density mountain. Each node in DP-Tree is a cluster-cell rather than a single point, which is for saving memory space and computation time as mentioned in Sec. 3.2.

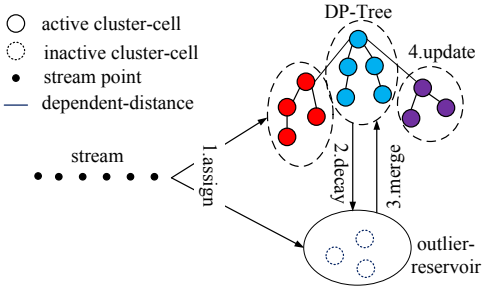


Figure 5: *EDMStream* Overview.

2) Outlier Reservoir. Due to the unlimitedness of stream, it is desirable to limit the size of DP-Tree in order to reduce maintenance overhead. On the other hand, due to the evolution of stream, the role of clusters and outliers may change. The outliers may form a new cluster if they absorbs new arrival points. The old cluster may decay to outliers if they have not absorbed points for long time. Therefore, we use an *outlier reservoir* for caching the cluster-cells with relatively low timely-density (i.e., temporal outliers), which are temporally not considered for clustering. Note that, a cluster-cell is moved to the outlier reservoir either because it contains only a few points (i.e., low local density) or because the contained points are outdated (i.e., low timely-density). The cluster-cells in the outlier reservoir are possible to absorb new points and be inserted to DP-Tree for clustering again.

Key Operations. *EDMStream* relies on **four** key operations as shown in Fig. 5.

1) New point assignment. A new point from stream is assigned to an existing cluster-cell (in DP-Tree or outlier reservoir) or forms a new outlier cluster-cell. A point p_i is assigned to a cluster-cell c if the two conditions are satisfied: i) the distance to the cluster seed s_c is smaller than or equal to r, i.e., $|p_i, s_c| \leq r$; ii) s_c is the closest cluster seed, i.e., $s_c = \arg\min_{s_k \in S_{seed}}(|p_i, s_k|)$ where S_{seed} is the set of all existing cluster-cell seeds. If no such cluster-cell exists, a new cluster-cell seeded by p_i is created and cached in outlier reservoir due to its low density.

2) Dependencies update. Due to the fading property, the densities of cluster-cells decrease over time. In addition, some of the cluster-cells might absorb new points. As a result, their densities increase, and their dependencies may change such that the DP-Tree is updated. We will present the details of dependencies update in Sec. 4.2.

3) Cluster-cell emergence (DP-Tree insertion). The timely-density of an existing cluster-cell in outlier reservoir increases after it absorbs a new point. It might be inserted into the DP-Tree for clustering once its density is large enough.

4) Cluster-cell decay (DP-Tree deletion). The timely-density of cluster-cells decays as the freshness of points is fading. If the density of a cluster-cell in DP-Tree is low enough, it might be temporally moved to the outlier reservoir. We will present the details of cluster-cell emergence/decay in Sec. 4.3.

Cluster Evolution Tracking. The operations 2), 3), and 4) lead to cluster evolution. As described in Sec. 3.3, we can track the evolution by monitoring the update of DP-Tree structure, including the changes of dependent distance (which will trigger the split or merge of MSDSubTrees), the insertion/deletion of cluster-cell nodes, and the movements of cluster-cell nodes between MSDSubTrees. The DP-Tree update operations along with the update time are then logged for future queries.

Initialization. Initially, a number of cluster-cells that absorb incoming points are cached in memory. Once the size of cached cluster-cells exceeds a predefined threshold, the density and the dependent distance of each cached cluster-cell are calculated in terms of Equation (6) and Equation (7) respectively. In the meantime, the dependencies of the cached cluster-cells are retrieved, which are used for initializing the DP-Tree structure. Furthermore, given τ a primary clustering result can be obtained as mentioned in Sec. 2.2. Next, we propose the techniques for efficiently updating the DP-Tree as new points coming.

4.2 Dependencies Update (DP-Tree Update)

The densities of all cluster-cells gradually decay as time goes by. But if cluster-cells absorb points, their densities should be increased. Moreover, the dependency relationship should also be updated accordingly.

Densities Update. The decayed density of summary structures has been well studied in the literature [5, 7, 9, 15, 26, 30]. Based on our time decay model, if a cluster-cell absorbs a points from t_j to t_{j+1}, its density is updated as follows. The proof can be referred to [31].

$$\rho_c^{t_{j+1}} = a^{\lambda(t_{j+1}-t_j)}\rho_c^{t_j} + 1. \tag{8}$$

Dependencies Update. The changes of densities may cause the dependency changes. Suppose that the density ρ_c of cluster-cell c increases and becomes larger than the density of c's original dependent cluster-cell. This means that c's dependent cluster-cell and its dependent distance δ_c should be updated according to Equation (7). It is also possible that c becomes the new dependency of other some cluster-cells, whose dependent distances should also be updated accordingly. Every time c absorbs new point, a large number of cluster-cells are involved in dependencies update. This can result in great computational burden, which poses challenge to real-time stream processing.

Let $\rho_c^{t_j}$ be the density of cluster-cell c at time t_j. Let $F_c^{t_j} = \{c' | \rho_c^{t_j} < \rho_{c'}^{t_j}\}$ be the set of cluster-cells whose density are higher than c at time t_j. From the point of view of DP-Tree, $F_c^{t_j}$ is the set of cluster-cells that are at higher levels than c at time t_j. We define $D_c^{t_j}$ as the ***dependent cluster-cell*** of c at time t_j.

$$D_c^{t_j} = \arg \min_{c':c' \in F_c^{t_j}} |s_c, s_{c'}|. \tag{9}$$

Since all cluster-cells' densities decay at the same rate, the order of their densities will not change from t_j to t_{j+1} except for the cluster-cell c' that absorbs new point. For each c, we just need to judge whether the updated c' newly appears in $F_c^{t_{j+1}}$ with respect to $F_c^{t_j}$. If so, c's dependency update is required, otherwise it can be avoided. This is because that, according to (9), as long as set F_c is consistent, c's dependency D_c will not change. From c''s perspective, only the nodes whose density are previously higher than or equal to $\rho_{c'}$ ($\rho_c^{t_j} \geq \rho_{c'}^{t_j}$) but currently lower than $\rho_{c'}$ ($\rho_c^{t_{j+1}} < \rho_{c'}^{t_{j+1}}$) are necessary to update dependencies. From the DP-Tree point of view, we are trying to find out the nodes that are previously are at higher levels than c' but

now at lower levels, and only update their dependencies. Therefore, in order to reduce update cost, we propose our first density filtering scheme through the following theorem.

THEOREM 1. ***(Density Filter)*** *Suppose another cluster-cell c' absorbs a point at time t_{j+1}.*

$$\text{If } \rho_c^{t_j} < \rho_{c'}^{t_j} \text{ or } \rho_c^{t_{j+1}} \geq \rho_{c'}^{t_{j+1}}, \text{then}$$
$$D_c^{t_j} = D_c^{t_{j+1}}.$$

That is, cluster-cell c's dependency will not change and it is not necessary to update c's dependencies.

PROOF. If $\rho_c^{t_j} < \rho_{c'}^{t_j}$, then $c' \in F_c^{t_j}$. After c' absorbs a new point at t_{j+1} we still have $\rho_c^{t_{j+1}} < \rho_{c'}^{t_{j+1}}$ and $c' \in F_c^{t_{j+1}}$. c' appears in both $F_c^{t_j}$ and $F_c^{t_{j+1}}$, i.e., $F_c^{t_j} = F_c^{t_{j+1}}$. Hence, $D_c^{t_j} = D_c^{t_{j+1}}$.

If $\rho_c^{t_{j+1}} \geq \rho_{c'}^{t_{j+1}}$, then $c' \notin F_c^{t_{j+1}}$. Even after c' absorbs a new point at time t_{j+1} $\rho_c^{t_{j+1}} \geq \rho_{c'}^{t_{j+1}}$, so at time t_j $\rho_c^{t_j} > \rho_{c'}^{t_j}$, i.e., $c' \notin F_c^{t_j}$. c' neither appears in $F_c^{t_j}$ nor $F_c^{t_{j+1}}$, i.e., $F_c^{t_j} = F_c^{t_{j+1}}$. Hence, $D_c^{t_j} = D_c^{t_{j+1}}$. □

In addition, we exploit the triangle inequality property and propose our second filtering scheme through the following theorem.

THEOREM 2. ***(Triangle Inequality Filter)*** *Suppose another cluster-cell c' absorbs a point p at time t_{j+1}.*

$$\text{If } \big||p, s_c| - |p, s_{c'}|\big| > \delta_c^{t_j}, \text{then}$$
$$D_c^{t_j} = D_c^{t_{j+1}}.$$

That is, cluster-cell c's dependency will not change and it is unnecessary to update c's dependencies.

PROOF. In terms of triangle inequality, $|s_c, s_{c'}| > \big||p, s_c| - |p, s_{c'}|\big|$. If $\big||p, s_c| - |p, s_{c'}|\big| > \delta_c^{t_j}$, then $|s_c, s_{c'}| > \delta_c^{t_j}$. According to the definition of dependent cluster-cell, $D_c^{t_j}$ is the *nearest* higher density cluster-cell. Therefore, it is not possible to replace c's original dependent cluster-cell by c'. Then we have $D_c^{t_j} = D_c^{t_{j+1}}$. □

According to Theorem 1, we can avoid the dependency update of c if $\rho_c^{t_j} < \rho_{c'}^{t_j}$ or $\rho_c^{t_{j+1}} \geq \rho_{c'}^{t_{j+1}}$. According to Theorem 2, we can further reduce the number of dependency updates if $\big||p, s_c| - |p, s_{c'}|\big| > \delta_c^{t_j}$. Since $|p, s_c|$ and $|p, s_{c'}|$ have already been measured during the point assignment phase, the filtering cost is almost free. Our experimental results in Sec. 5.3.3 will show that a significant performance improvement is achieved.

4.3 Cluster-Cells Emergence and Decay

As discussed in Sec. 4.1, an outlier reservoir caching low timely-density cluster-cells is designed to maintain the outliers or halos. We only consider the dense cluster-cells for clustering. This is because that dense regions are more representative to reflect stream trends, and losing sight of sparse regions is helpful for distinguishing the true clusters. However, considering that the low density cluster-cells may become dense as they absorb new points, it is not a good idea to delete them immediately. Accordingly, we preserve these low density cluster-cells in outlier reservoir temporarily. We call the cluster-cells residing in DP-Tree as *active* cluster-cells and the ones in outlier-reservoir as *inactive* cluster-cells.

For ease of exposition, we assume a fixed point arrival rate v, i.e., $t_{i+1} - t_i$ is equal for any i and $v = \frac{1}{t_{i+1}-t_i}$. By referring to [5, 7], given a decay model with parameters a and λ, the sum of all data points' freshness $a^{\lambda(t-t_i)}$ for an unbounded data stream is a constant $\frac{v}{1-a^\lambda}$, i.e., $\sum_{i=1}^{n}\left(a^{\lambda(t_n-t_i)}\right) = \frac{v}{1-a^\lambda}$ where $n \to \infty$. Accordingly, we distinguish active and inactive cluster-cells as follows. A cluster-cell c at time t is active if $\rho_c^t \geq \frac{\beta \cdot v}{1-a^\lambda}$ and otherwise it is inactive. β is a tunable parameter that controls the threshold. The larger the value of β is, the less number of active cluster-cells is. Obviously, β is less than 1 since a single cluster-cell's density should not exceed the sum of all cluster-cells' densities (which is equal to the sum of all points' freshness), i.e., $\frac{\beta \cdot v}{1-a^\lambda} \leq \rho_c^t < \frac{v}{1-a^\lambda}$. On the other hand, since a new cluster-cell formed by a new arrival point should be considered as inactive, we have $\rho_c^t = 1 < \frac{\beta \cdot v}{1-a^\lambda}$. Thus, we have the range of β, i.e., $\frac{1-a^\lambda}{v} < \beta < 1$.

The active cluster-cell may become inactive and be moved from the DP-Tree to the outlier reservoir. Suppose a cluster-cell c becomes inactive at time t, i.e., $\rho_c^t < \frac{\beta \cdot v}{1-a^\lambda}$. Due to the fact that the density of cluster-cell c's successors are all lower than ρ_c^t, cluster-cell c's successor cluster-cells should also be moved to the outlier reservoir. It is unnecessary to judge their densities or update their dependent distances. On the other hand, the inactive cluster-cells may absorb new arrival points to increase density. Then they may become active cluster-cells and be inserted into the DP-Tree. The DP-Tree insertion leads to the dependencies update. We follow Theorem 1 and Theorem 2 to reduce the overhead of dependencies update.

As new data are continuously being collected, more and more cluster-cells could be created due to the expansion of data space. The maintenance of cluster-cells consumes large memory space. In practice, if data are old enough they can be ignored for clustering since they are invalid for discovering the hidden patterns and the trends in stream. For the sake of recycling memory space, we should delete the outdated cluster-cells. If the time of inactive cluster-cell has not absorbed any point is equal to the time for a new active cluster-cell being formed by new arrival points, the inactive cluster-cell can be deleted safely. Due to the space limitation, the technical details of recycling memory space are presented in our technical report [1]

4.4 Adaptive Tuning of τ

The parameter τ in *EDMStream* controls the degree of cluster separation and cluster granularity. Large τ tends to result in less number of large clusters, while small τ tends to result in much more small clusters. As the data distribution of stream evolves over time, the key parameter τ should not be set statically but dynamically to adapt to data evolution. When the stream points are loosely distributed, τ should be a larger value, and vice verse. The original DP Clustering [25] draws a decision graph (see Fig. 2b) to help users determine an appropriate τ value. However, it is not suitable for stream clustering since it is performed frequently rather than once, which is expensive and causes great inconvenience to users. Even though the user-interaction method does not work, it inspires us to learn the preference of users and propose an automatic tuning approach for τ. Due to the space limitation, the technical details of adaptive tuning of τ are presented in our technical report [1]

Table 2: Datasets

data set	instances	dim	clusters	r
SDS	20,000	2	2	0.3
HDS	100,000	10	20	60
		30	20	65
		100	20	68
		300	20	70
		1000	20	70
NADS	422,937	-	7231	0.4
KDDCUP99	494,021	34	23	100
CoverType	581,012	54	7	250
PAMAP2	447,000	51	13	5

5. EXPERIMENTAL EVALUATION

In this section, we present the experimental evaluation of *EDMStream*. All experiments are conducted on a commodity PC with 3.4GHz Intel Core i3 CPU and 8GB memory.

5.1 Preparation

Datasets. Our experiments involve two synthetic datasets (SDS and HDS) and four real datasets (NADS [18], KDDCUP99 [28], CoverType [22, 24] and PAMAP2 [23]). All the real datasets are with ground truth information. The SDS dataset [21] is generated with 2-D stream points to visually show cluster results. The HDS stream dataset is generated with various dimensions based on the approach mentioned in [29]. Both the synthetic and real datasets are converted into streams by taking the data input order as the order of streaming except for NADS. The NADS is a news stream dataset with time information (there is no dimension information for NADS since the data are short text). The features of these datasets are listed in Table 2.

Comparison Counterparts. We implement D-Stream [7], DenStream [5], DBSTREAM [13] and MR-Stream [30] for comparison[5], all of which are density-based. The counterparts all use an online component (i.e., data summarization structure) that maps each stream object to a grid [7, 30] or microcluster [5, 13] and an offline component that performs a batch-mode classical clustering algorithm to update clustering result.

Parameters Setup. In our experiment, we fix the data points arrival rate as 1,000 pt/s unless particularly mentioned. Different algorithms have different decay parameter requirements. In order to make equal decay effect, we carefully set the decay parameters as follows. In *EDMStream* and D-Stream, we set $a = 0.998$ and $\lambda = 1$ such that $a^\lambda = 0.998$. In MR-Stream, because $a = 1.002$ is fixed, so we set $\lambda = -1$ such that $a^\lambda = 0.998$. In DenStream and DBSTREAM, they fix $a = 2$, therefore we set $\lambda = 0.0028$ such that $a^\lambda = 0.998$. We use these parameter settings for achieving the same decay rate. In addition, we set $\beta = 0.0021$ as discussed in Sec. 4.3. For the radius r, we refer to the method of choosing d_c in [25]. We will discuss the effect of r in Sec. 5.7. We set the other parameters of the compared algorithms by referring to their papers.

5.2 Tracking Cluster Evolution

An important feature of a stream clustering algorithm is the ability to track the evolution of clusters. We first use a synthetic 2-D dataset SDS to visually show the ability of tracking clustering evolution. We then show a use case of news recommendation application to illustrate how to utilize the ability of clustering evolution tracking.

[5]The source codes of these algorithms are not public available except for DBSTREAM.

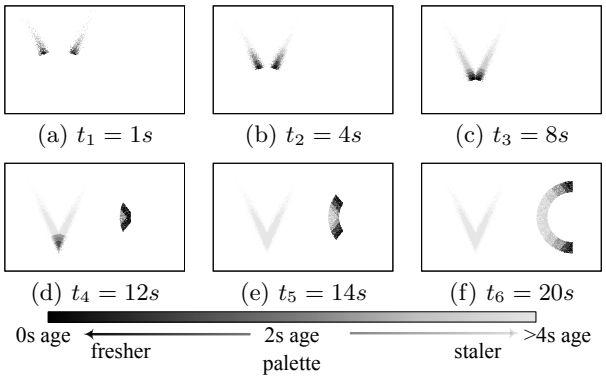


Figure 6: Data distribution snapshots with time decay information at different time points (SDS)

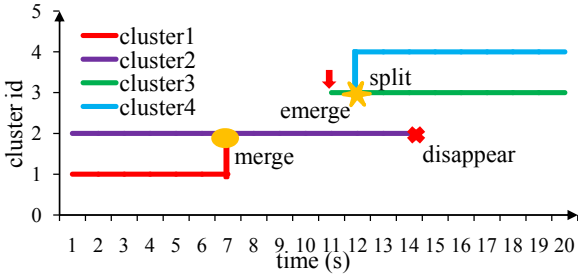


Figure 7: Cluster evolution activities (SDS)

5.2.1 *Synthetic Dataset*

Fig. 6 shows the data stream with 6 snapshots taken in $t_1 = 1s$, $t_2 = 4s$, $t_3 = 8s$, $t_4 = 12s$, $t_5 = 14s$, $t_6 = 20s$. Various degrees of grey indicate the freshness of data. The darker one is fresher, the lighter one is staler. SDS contains 20,000 points and the point arrival speed is set to 1,000 pt/s, so the stream ends at 20s.

We show the cluster evolution tracking result in Fig. 7. Different color lines indicate different clusters. The length of lines indicates the lifecycle of clusters. Multiple branches split from one line or merging into one means cluster splitting or merging. According to Fig. 6, from 1s to 4s, we can see that the shapes and locations of two clusters are evolving, they are moving closer to each other. At 9s, these two clusters merge into one single cluster. At 12s, a new cluster emerges at right side, and the left cluster shrinks gradually. At 14s, the left cluster disappears completely, and the right cluster has been split into two different clusters. They are also moving to the opposite directions from each other. Finally at 20s, the two clusters move far away from each other. We can see that Fig. 7 successfully captures all the cluster evolution activities.

5.2.2 *Use Case Study: Monitoring Cluster Evolution in News Recommendation*

A real application of stream clustering is news recommendation. The news in the same cluster as that a user has read is recommended to the user. For the text dataset, the Jaccard distance is used. The density of cluster-cell is computed as defined in Equation 6. We run *EDMStream*

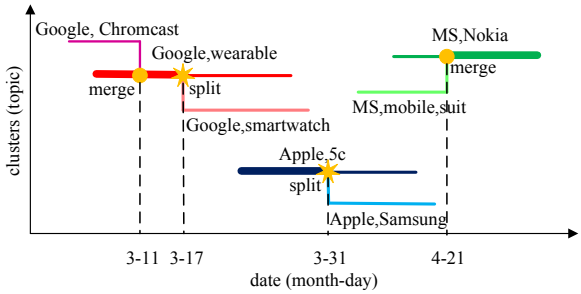


Figure 8: Cluster evolution activities (NADS)

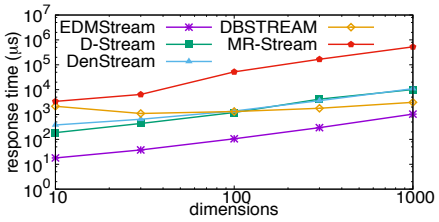


Figure 12: Response time while varying data dimensions (HDS)

on the NADS news stream and depict the cluster evolution tracking result in Fig. 8[6]. We show the tags of each news cluster which indicate topics. Furthermore, we also show the real events which we think lead to the cluster evolution in Table 3.

On 3-11, a cluster with tags {*Google*, *Chromcast*} merges into another cluster with tags {*Google*, *wearable*}. The reason is analyzed as follows. The news about Chromcast are not hot anymore, but these news are highly related to another news topic {*Google*, *wearable*}. Given that many news about "Google launches SDK for Android wearables" come out at that time, these two clusters merge together. Later, Google confirms smartwatch plans. These news firstly are classified into the {*Google*, *wearables*} cluster. But with the increase of popularity, they are split from original cluster and form a new cluster {*Google*, *smartwatch*} on 3-17. On 3-31, the news cluster {*Apple*, *Samsung*} is split from the original cluster {*Apple*, *5c*}, because there is a shocked event that Apple and Samsung battle for patent and the growing media focus on it instead of Apple's iphone 5c. Similarly, on 4-21 the cluster {*MS*, *mobile*, *suit*} merges into another cluster with tags {*MS*, *Nokia*} since the news about "Microsoft's acquisition of Nokia" become popular and the news about "Microsoft' mobility office suite" get less and less attention. From this experiment, we can see that our cluster evolution tracking method can successfully identify different types of cluster evolution activities, including cluster emerging, disappearing, splitting and the merging of clusters.

5.3 Efficiency

The ability of updating clustering result in real-time is crucial for stream clustering. We compare *EDMStream* with the competitor algorithms in terms of efficiency in the following.

5.3.1 Response Time

We run *EDMStream* and their competitor algorithms with fixed point arrival rate 1K/s. Fig. 9 shows the average response time of different algorithms in a time interval of 25 s. MR-Stream fails to process stream with 1K/s on all streams. DenStream fails on CoverType and PAMAP2 streams. DBSTREAM and D-Stream work well in the beginning but fail later. Only *EDMStream* is fast enough to process stream with 1K/s point rate. *EDMStream* requires much less response time than others due to the fact that *EDMStream* relies on online and incremental cluster update while the others relies an costly offline clustering step.

[6]There should be about 7281 clusters at the end of the stream. We only show a few cluster evolution tracking results in the figure.

5.3.2 Throughput

We run a stress test to see the maximum throughput that we can achieve. We remove the limitation of 1K/s point arrival rate and process as many points as possible. Fig. 10 shows the results of our *EDMStream* and its competitors. The throughput of *EDMStream* can be up to 10K-170K pt/s, which achieves 7-15x speedups than other algorithms.

5.3.3 Filtering Strategies

The clustering result update in *EDMStream* requires to update all the dependent distances (δ^t) of all cluster-cells, which is the most costly step. We propose two filtering strategies to avoid unnecessary updates (see Sec. 4.2). To illustrate the effect of these filtering strategies, we first run *EDMStream* without any filtering (i.e., wf). We then turn on the density filtering scheme described in Lemma 1 (i.e., df) and the triangle inequality filtering scheme described Lemma 2 (i.e., df + tif). The accumulated time for the dependencies update is depicted in Fig. 11. We can see that our filtering schemes greatly reduce the update time.

5.3.4 Varying Data Dimensions

We evaluate the *EDMStream* on synthetic streams HDS with different data dimensions, 10D, 30D, 100D, 300D, 1000D. In this experiment, we remove the speed limits for preventing some competitors to fail. Fig. 12 shows the average response time of *EDMStream* and its competitors on these streams. These algorithms exhibit similar trend when processing various-dimensions data streams. As the dimensionality increases, most algorithms require more time to update clustering result. This is under expectation, since more computation cost is needed for high dimensional data. The reason why DBSTREAM shows abnormal result is that, the performance of DBSTREAM is sensitive to the density of space and it runs faster on low density space. A stream with low dimensionality could lead to relatively high density of space, so there is a tradeoff between extra cost for high density data and the computation cost for high dimensional data.

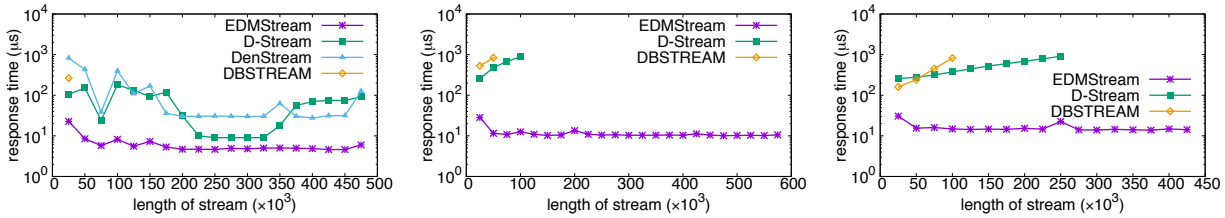
5.4 Cluster Quality

To test the clustering result quality, we take state-of-the-art algorithms D-Stream, DenStream, DBSTREAM and MR-Stream as our competitors. Furthermore, we also evaluate the clustering results quality by varying stream rates.

The commonly used cluster quality evaluation metrics fail to take the evolution of streams and the freshness of instances into account. Take the *F-measure* [8] as an example. The merge or split of clusters retrieved from stream points could be considered as false positive or false negative. In our evaluation, we use a recently proposed *CMM* (*Clustering Mapping Measure*) criterion [17], which is external criterion

Table 3: Cluster evolutions and their related events

Split			
original cluster	**cluster 1**	**cluster 2**	**event**
Google wearable	Google wearable	Google smartwatch	On 3-17, "google confirms smartwatch plans unveils android wear"
Apple 5c	Apple 5c	Apple Samsung	On 3-31, "apple samsung renew patent battle court"
Merge			
cluster 1	**cluster 2**	**merged cluster**	**event**
Google Chromast	Google wearable	Google wearable	On 3-11, "google exec promises wearables sdk developers"
MS mobile suit	MS Nokia	MS Nokia	On 4-21, "msft nok nokia phones renamed microsoft mobile"



(a) KDDCUP99 (b) CoverType (c) PAMAP2

Figure 9: The comparison of response time

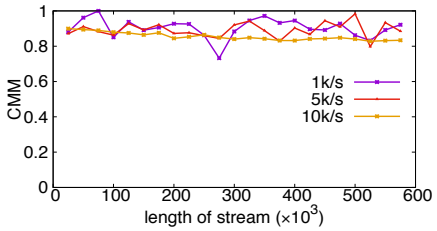


Figure 14: Cluster quality varying stream rate

taking into account the age of objects. The CMM measure can reflect errors related to emerging, splitting, or moving clusters, which are situations inherent to the streaming context. The CMM measure is a combination of penalties for each one of the following faults: 1) Missed objects. Clusters that are constantly moving may eventually lose objects, and thus CMM penalizes for these missed objects; 2) Misplaced objects. Clusters may eventually overlap over the course of the stream, and thus CMM penalizes for misplaced objects; 3) Noise inclusion. CMM penalizes for noisy objects being inserted into existing clusters. Basically, the CMM value ranges from 0 to 1. Larger CMM value indicates higher cluster quality, while smaller CMM value indicates lower cluster quality.

5.4.1 Compare with state-of-the-art algorithms

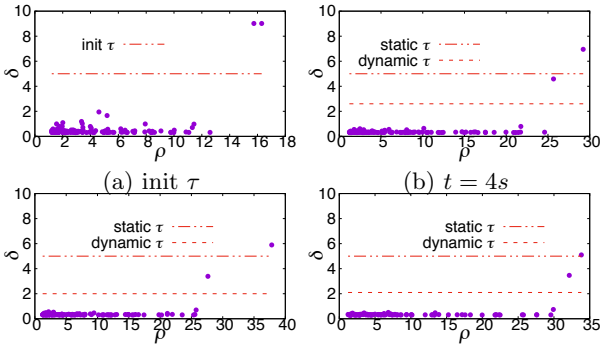
Since we focus on cluster quality in this experiment, we reduce the stream points rate as much as possible to let competitors run normally. We launch these algorithms on three real datasets and compare their CMM [17, 26] metrics. Fig. 13 shows their cluster CMM values over time. Our *EDMStream* has comparable cluster quality with other algorithms. It is notable that MR-Stream treats each point as a cluster on the CoverType and PAMAP2 datasets, we do not consider it. Generally speaking, *EDMStream*, DenStream, and DBSTREAM outperform D-Stream and MR-Stream in terms of the CMM metric.

5.4.2 Varying Stream Rate

We evaluate the cluster quality of *EDMStream* by varying the stream rates (1K/s, 5K/s, 10K/s) on the CoverType dataset. Because the competitors fail with higher stream rate, we just compare CMM metrics of *EDMStream* with different stream rates. The results are shown in Fig. 14. We can see that the cluster quality are robust with various stream rates. *EDMStream* is stable even in higher stream rate.

Table 4: The number of clusters changes over time (SDS)

t (s)	1	2	3	4	5	6	7	8	9	10
dynamic τ	2	2	2	2	2	2	1	1	1	1
static τ	2	2	2	1	1	1	1	1	1	1



(a) init τ (b) $t = 4s$

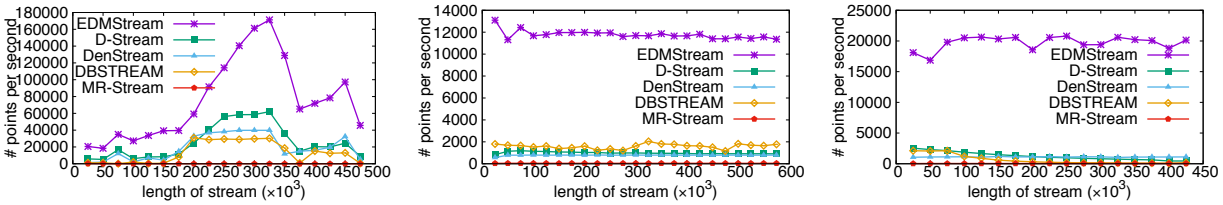
(c) $t = 5s$ (d) $t = 6s$

Figure 15: Decision graphs at different time points (SDS)

5.5 Adaptability: Dynamic τ vs. Static τ

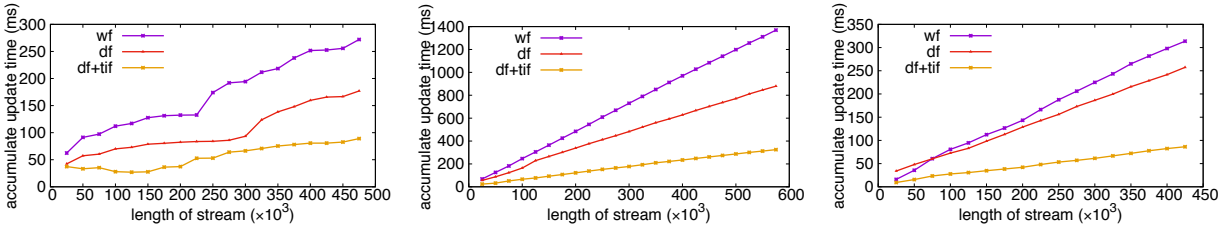
EDMStream has the ability of adjusting itself to be adapt to the data distribution evolution (see Sec. 4.4 and Fig. 6). Precisely, *EDMStream* can dynamically adjust the setting of the key parameter τ, which controls the cluster separation granularity. In order to show the effectiveness of our dynamical τ setting strategy, we compare our *dynamic* method with the *static* method, which sets τ as a constant.

We use the synthetic SDS to evaluate the *dynamic* method and the *static* method. Table 4 shows the number of clusters in the first 10 seconds, where the 1st second result is the init result with user participation. The result of other time



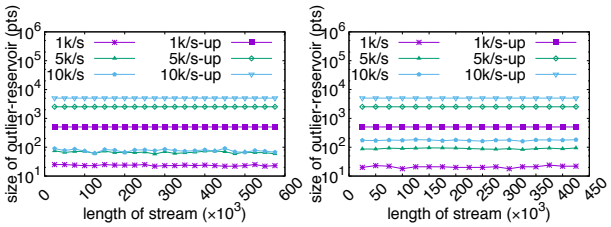
(a) KDDCUP99 (b) CoverType (c) PAMAP2

Figure 10: Comparison of throughput



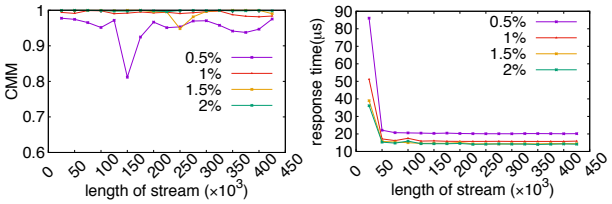
(a) KDDCUP99 (b) CoverType (c) PAMAP2

Figure 11: Accumulated time for dependencies update



(a) CoverTime (b) PAMAP2

Figure 16: Size of outlier reservoir



(a) Cluster quality (b) Response time

Figure 17: Effect when varying r (PAMAP2)

points is generated by algorithm automatically. We can see that the result is different at 4-6s. In order to identify the correct cluster result, we draw the decision graphs at initial time and at these three time points as shown in Fig. 15. In the initial step, we choose $\tau = 5$ since it can distinguish the density peaks from other objects. Then the *dynamic* method adjust τ value over time, while the *static* method keeps using the same $\tau = 5$.

As shown in Fig. 15, at 4s-6s, two density peaks are higher than the dynamic τ line, but only one is higher than the static τ line. That is, two clusters exist at 4s-6s by dynamically adjusting τ while only one exists by using fixed τ. Let us look at the original data distribution at 4s in Fig. 6b. It is obvious that the result of two clusters makes more sense. By using the static τ, it will fail to identify the other density peak and as a result obtain a wrong clustering result. Therefore, our stream cluster algorithm has self-adjustment ability, which is important for a stream clustering algorithm to run for a long period.

5.6 Size of Outlier Reservoir

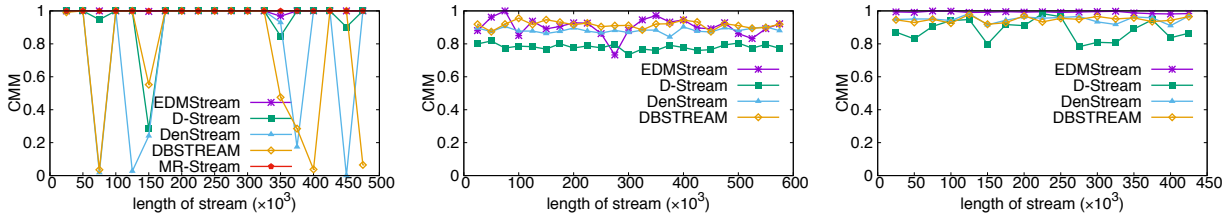
Tackling outliers is important for stream clustering, especially when massive outliers are mixed in stream. *EDMStream* buffers the potential outliers (inactive cluster-cells) in the outlier reservoir. As discussed in our technical report [1], given an average point arrival rate, the number of outliers has a theoretical upper bound in order to limit the size of outlier reservoir. In this experiment, we vary the point arrival rate (1K/s, 5K/s, 10K/s) and measure the outlier reservoir's size every 25 seconds. Fig. 16 shows the theoretical upper bounds of size (1K/s-up, 5K/s-up, 10K/s-up) and the measured sizes (1K/s, 5K/s, 10K/s). We see that the actual reservoir size is far less than the upper bound. The size of outlier reservoir can be predicted given the average point arrival rate. In order to reduce the size of outlier reservoir, users can accordingly adjust the decay model parameters to prolong point's freshness.

5.7 Effect of Cluster-Cell Radius r

The setting of cluster-cell radius r has the effect on the cluster quality as well as processing speed. We test the clustering quality by varying r. As suggested in the original Density Peaks Clustering paper [25], r is chosen from 0.5% to 2% of the distance of all pairs of objects in ascending order. Fig. 17 shows the cluster quality and response time when varying r. Smaller r means more fine-grain cluster-cells. As result, it results in higher quality clusters but more computation overhead (i.e., longer response time). On the contrary, larger r means less coarse-grain clusters so as to obtain lower quality clusters but return result faster.

6. RELATED WORK

Clustering is one of the most important topics in data mining and has been very extensively studied. In recent years, real-time analysis and mining of stream data have attracted much attention from the research community.



(a) KDDCUP99 (b) CoverType (c) PAMAP2

Figure 13: The comparison of clustering result quality

Many stream data clustering algorithms have also been proposed.

Offline Clustering vs. Online Clustering. A class of stream clustering algorithms use a two-step approach, such as CluStream [3], DenStream [5] and D-Stream [7], MR-Stream [30] and DBSTREAM [13]. The procedure of clustering contains an online data abstraction step and an offline clustering step. In the data abstraction step, the data stream is summarized using a specific data structure (e.g., micro-cluster and grid). The offline clustering step is triggered upon request, and a classical clustering method (e.g., k-means [19] or DBSCAN [9]) are used on these summarized data to obtain the cluster result. Our *EDMStream* relies on online cluster result updates, which can return cluster results in real-time.

DBSCAN-based Clustering vs. DP-based Clustering. A number of stream clustering algorithms are based on DBSCAN [9] algorithm, e.g., DenStream [5], D-Stream [7], MR-Stream [30] and DBSTREAM [13]. DBSCAN uses the density-connected information to build a density-connected graph. The clustering is to find all the *maximal density-connected components* from the graph. DP clustering relies on the dependency relationship to build a DP-tree. The clustering is to find all the *maximal strongly dependent subtrees* from the DP-Tree. As the tree (in DP) or graph (in DBSCAN) is continuously updating, we have to query the specific subtrees or subgraphs quickly according to the update. The *hierarchical* tree structure is naturally more suitable for such a query than the *flat* graph structure. This is because that it is much easier to identify the affected parts in tree than in graph. We only need to handle the affected successors of the update node in the tree. However in graph structure, we have to re-evaluate the whole graph since there is no obvious affected part. This is why we rely on DP clustering and density mountain.

CF-tree vs. DP-Tree. One of the earliest stream clustering algorithm is BIRCH [31]. BIRCH uses *cluster feature vector* (CF) to build CF-tree which abstracts the agglomerative hierarchical clustering results. Though both *EDMStream* and BIRCH use tree like structure to abstract clusters, i.e., DP-Tree and CF-Tree, these two trees are fundamentally different. DP-Tree is used to abstract the density mountains, where the nodes represents stream points or cluster-cells, and the links depict the dependency relationship between points. CF-Tree is used to manage the hierarchical clustering results, where the nodes represent certain grain level clusters, and the links depict the relationship between coarse-grain clusters and fine-grain clusters.

Dynamic Clustering vs. Stream Clustering. Recently, Gan and Tao [11] propose a new DBSCAN-based algorithm for dynamic clustering, which can return updated clustering results very quickly (say in microseconds). The proposed dynamic clustering algorithm efficiently maintains data clusters along with point insertion/deletion occurring in the underlying dataset. While in stream clustering, we assume that data points are inserted continuously with timestamp information. The timestamp information is a key feature for streaming objects, which defines a freshness level. Data can also be decayed to meaningless levels. The time information reflects the emerging trends and is important for tracking cluster evolution. Although both dynamic clustering and stream clustering aim to return the updated clustering results in real-time, stream clustering distinguishes fresh data from stale data and tends to give the fresh data more weight in clustering.

Cluster Evolution Tracking. All the above stream clustering algorithms fail to capture the cluster evolution activities. A few algorithms are proposed to monitor cluster evolution, e.g., MONIC [27] and MEC [20]. They trace the evolution of clusters by identifying the overlapping degree between two clusters. In MEC [20], evolution tracking mechanism relies on the identification of the temporal relationship among them. In *EDMStream*, we take full advantage of DP-Tree to track the evolutions of clusters. We can track the evolution according to the updates of DP-Tree structure.

7. CONCLUSION

In this paper, we propose the *EDMStream* algorithm, an effective and efficient method for clustering stream data. This method can not only track the evolution of stream data by monitoring the density mountain but also response cluster update in real-time. By using the DP-Tree structure and a number of filtering schemes, the cluster result updates can be completed very quickly. *EDMStream* also has the ability of adjusting itself to adapt to data evolution. This feature distinguishes it from most other stream clustering algorithms. Our experimental results demonstrate the effectiveness, efficiency, and adaptability of our algorithm. *EDMStream* responses to a cluster update in 7-23 μs on average by using commodity PC and at the same time achieves comparable cluster quality. Furthermore, *EDMStream* successfully captures the cluster evolution activities. The high effectiveness and adaptability significantly improve the scalability and practicability of *EDMStream*.

Acknowledgement. This work was partially supported by the National Natural Science Foundation of China (61433008, 61672141, 61528203, U1435216), Fundamental Research Funds for the Central Universities (N161604008), State Key Laboratory of Computer Architecture, CAS (CARCH201610). Ge Yu is the corresponding author.

8. REFERENCES

[1] https://arxiv.org/pdf/1710.00867.pdf.

[2] C. C. Aggarwal. *Data streams: models and algorithms*, volume 31. Springer Science and Business Media, 2007.

[3] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *Proceedings of the VLDB*, pages 81–92, 2003.

[4] N. Begum and E. Keogh. Rare time series motif discovery from unbounded streams. *VLDBJ*, 8(2):149–160, 2014.

[5] F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *Proceedings of the SDM*, pages 328–339, 2006.

[6] L. Cao, Q. Wang, and E. A. Rundensteiner. Interactive outlier exploration in big data streams. *VLDBJ*, 7(13):1621–1624, 2014.

[7] Y. Chen and L. Tu. Density-based clustering for real-time stream data. In *Proceedings of the SIGKDD*, pages 133–142, 2007.

[8] H. S. Christopher D. Manning, Prabhakar Raghavan. *Introduction to Information Retrieval.* Cambridge University Press, 1993.

[9] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the KDD*, pages 226–231, 1996.

[10] J. Gan and Y. Tao. Dbscan revisited: mis-claim, un-fixability, and approximation. In *Proceedings of SIGMOD*, pages 519–530, 2015.

[11] J. Gan and Y. Tao. Dynamic density based clustering. In *Proceedings of the SIGMOD*, pages 1493–1507. ACM, 2017.

[12] S. Gong and Y. Zhang. Eddpc:an efficient distributed density peaks clustering algorithm. *Computer Research and Development*, 53(6):1400–1409, 2016.

[13] M. Hahsler and M. Bolaños. Clustering data streams based on shared density between micro-clusters. *IEEE TKDE*, 28(6):1449–1461, 2016.

[14] H. Huang and S. P. Kasiviswanathan. Streaming anomaly detection using randomized matrix sketching. *VLDBJ*, 9(3):192–203, 2015.

[15] C. Isaksson, M. H. Dunham, and M. Hahsler. *SOStream: Self Organizing Density-Based Clustering Over Data Stream.* Springer Berlin Heidelberg, 2012.

[16] P. Kranen, I. Assent, C. Baldauf, and T. Seidl. The clustree: indexing micro-clusters for anytime stream mining. *KAIS*, 29(2):249–272, 2011.

[17] H. Kremer, P. Kranen, T. Jansen, T. Seidl, A. Bifet, G. Holmes, and B. Pfahringer. An effective evaluation measure for clustering on evolving data streams. In *Proceedings of the KDD*, pages 868–876, 2011.

[18] M. Lichman. UCI machine learning repository, http://archive.ics.uci.edu/ml, 2013.

[19] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Berkeley symposium on mathematical statistics and probability*, pages 281–297, 1967.

[20] M. Oliveira and J. Gama. A framework to monitor clusters evolution applied to economy and finance problems. *Intelligent Data Analysis*, 16(1):93–111, 2012.

[21] Y. Pei and O. Zaïane. A synthetic data generator for clustering and outlier analysis. *Technical Report*, 2006.

[22] M. Ranasinghe, G. BeeHua, and T. Barathithasan. Estimating willingness to pay for urban water supply: a comparison of artificial neural networks and multiple regression analysis. *Impact Assessment and Project Appraisal*, 17(4):273–281, 1999.

[23] A. Reiss and D. Stricker. Creating and benchmarking a new dataset for physical activity monitoring. In *Proceedings of the Affect and Behaviour Related Assistance*, pages 1–8, 2012.

[24] A. Reiss and D. Stricker. Introducing a new benchmarked dataset for activity monitoring. In *Proceedings of the ISWC*, pages 108–109, 2012.

[25] A. Rodriguez and A. Laio. Clustering by fast search and find of density peaks. *Science*, 344(6191):1492–1496, 2014.

[26] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. Carvalho, C. P. L. F. De, and J. Gama. Data stream clustering: A survey. *ACM Computing Surveys*, 46(1):125–134, 2013.

[27] M. Spiliopoulou, I. Ntoutsi, Y. Theodoridis, and R. Schult. Monic: modeling and monitoring cluster transitions. In *Proceedings of the SIGKDD*, pages 706–711, 2006.

[28] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. K. Chan. Cost-based modeling and evaluation for data mining with application to fraud and intrusion detection: Results from the jam project. In *Proceedings of the KDD*, pages 130–144, 1999.

[29] J. R. Vennam and S. Vadapalli. Syndeca: A tool to generate synthetic datasets for evaluation of clustering algorithms. In *Proceedings of the COMAD*, pages 27–36, 2005.

[30] L. Wan, W. K. Ng, X. H. Dang, P. S. Yu, and K. Zhang. Density-based clustering of data streams at multiple resolutions. *ACM TKDD*, 3(3):49–50, 2009.

[31] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. In *Proceedings of the SIGMOD*, pages 103–114, 1996.

[32] X. Zhang, C. Furtlehner, C. Germain-Renaud, and M. Sebag. Data stream clustering with affinity propagation. *IEEE TKDE*, 26(7):1644–1656, 2014.

[33] Y. Zhang, S. Chen, and G. Yu. Efficient distributed density peaks for clustering large data sets in mapreduce. *IEEE TKDE*, 28(12):3218–3230, 2016.