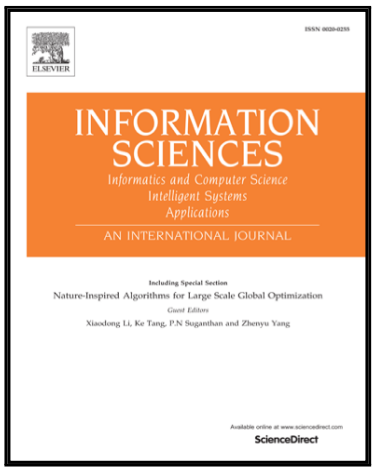


Synchronization-based Clustering on Evolving Data Stream

Junming Shao, Yue Tan, Lianli Gao, Qinli Yang, Claudia Plant, Ira Assent

PII: S0020-0255(18)30740-0
DOI: https://doi.org/10.1016/j.ins.2018.09.035
Reference: INS 13946

To appear in: *Information Sciences*

Received date: 6 May 2018
Revised date: 10 September 2018
Accepted date: 17 September 2018

Please cite this article as: Junming Shao, Yue Tan, Lianli Gao, Qinli Yang, Claudia Plant, Ira Assent, Synchronization-based Clustering on Evolving Data Stream, *Information Sciences* (2018), doi: https://doi.org/10.1016/j.ins.2018.09.035

Synchronization-based Clustering on Evolving Data Stream

Junming Shao,Yue Tan, Lianli Gao, Qinli Yang*

School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China

Claudia Plant

Faculty of Computer Science, University of Vienna, Vienna, A-1090 Austria

Ira Assent

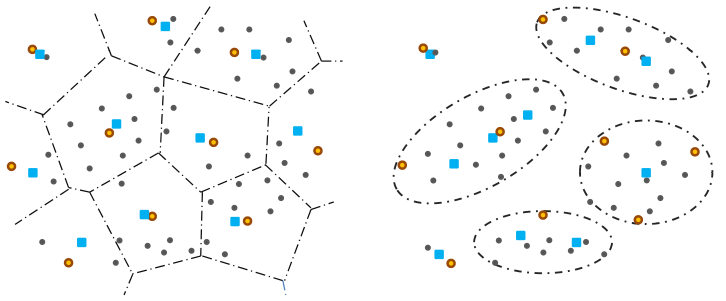
Department of Computer Science, Aarhus University, Aarhus DK-8200, Denmark

Abstract

Clustering streams of data is of increasing importance in many applications. In this paper, we propose a new synchronization-based clustering approach for evolving data streams, called SyncTree, which maintains all micro-clusters at different levels of granularity depending upon the data recency. Instead of using a sliding window or decay function to focus on recent data, SyncTree summarizes all continuously-arriving objects as synchronized micro-clusters sequentially in a batch fashion. Owing to the powerful concept of synchronization, the derived micro-clusters truly reflect the intrinsic cluster structure rather than summarize statistics of data, and old micro-clusters can be intuitively summarized at a higher level by iterative clustering to fit memory constraints. Building upon the hierarchical micro-clusters, SyncTree allows investigating the cluster structure of the data stream between any two time stamps in the past, and also provides a principled way to analyze the cluster evolution. Empirical results demonstrate that our method has good performance compared to state-of-the-art algorithms.

Keywords: Data stream, Clustering, Synchronization, Evolving analysis

*Corresponding author
Email address: qinli.yang@uestc.edu.cn (Qinli Yang*)



(a) CF-based data summarization (b) An ideal data summarization

Figure 1: A simple fictitious data stream chunk. (a) Traditional cluster feature based micro-clusters may produce a bad summarization of the data. Suppose ten micro-clusters (yellow points) are maintained at this moment. Incoming data examples (black points) will be assigned to the closest micro-clusters as indicated in the dashed areas, and the updated micro-clusters are indicated by blue points. (b) An ideal summarization of the incoming data examples with synchronized micro-clusters (indicated by blue points as well), where the local cluster structure is well preserved.

1. Introduction

Mining data streams has gained increasing attention in recent years due to its wide applications in real-time surveillance systems, telecommunication systems and sensor networks, etc. One primary task of data stream mining is clustering, which is a useful technique for structuring and organizing vast amounts of continuous examples in data streams. However, due to the potential infinite length and evolving nature of a data stream, finding its evolving cluster structure is still a challenging problem [19, 2, 6, 23, 36, 21, 22, 10, 11]. As new examples arrive continuously in a data stream, data stream clustering has to be performed in a single pass with limited memory and limited time. The evolving nature also means data stream clustering should be capable of clearly capturing changes.

During the past decades, many data stream clustering algorithms have been proposed. To handle the large volumes of streaming data with limited mem-

ory, algorithms usually first need to perform data abstraction, using a suitable data structure to maintain the summary statistics instead of storing all incoming examples. Currently, the most wide-spread data structure for data stream clustering is the Cluster Feature (CF) vector [40] (also called micro-cluster), which is a tuple $CF = (n, LS, SS)$ by encapsulating n objects with their linear sum LS and their squared sum SS. Due to the additivity of cluster feature entries, the incoming examples can be easily assigned to the most similar micro-cluster incrementally, and allow scaling clustering to large volumes of streaming data. However, there are two potential limitations of CF-based data representation which have not been studied before to the best of our knowledge. Firstly, the summary statistics maintained in cluster feature vector are insufficient to support high-quality cluster identification in some cases. (1) CF-based micro-clusters tend to fail for arriving data with non-Gaussian distributions or cluster evolution (see Figure 1(a), where the maintained micro-clusters cannot well represent the current data cluster structure). The reason is that the CF-based data summarization actually results in a partitioning of the data space into Voronoi cells. A better data representation is required to maintain the local cluster structure (i.e. not only update the summary statistics of data) (see Figure 1(b)). (2) To handle the concept drift, the wide-spread way is to use a sliding window or decay function to reflect the importance of recent data that should dominate the summary statistics in a CF-based micro-cluster. However, both window and decay function based strategies implicitly assume each data object in a data stream has a fixed decay rate or a same window size to survive (i.e. all objects have the same lifespan). It is not only a non-trivial task to determine an optimal window or decay speed for different data streams, and more importantly, the lifespan of the objects to a cluster are typically not prior knowledge, and they often differ largely. For example, considering the purchasing stream of customers on an E-commerce platform, we can find some consumption patterns of customers (i.e. clusters). However, these consumption patterns often dynamically changes over time, caused by the change of user preference. The lifespan of preference on products for each customer usually differs a lot. Be-

yond, as the cluster structure underlying the data streams usually changes over time (e.g. shrinking, expanding, disappearance or new emergence of clusters), it is important to monitor and analyze these changes in a principled way.

Recently, inspired by natural synchronization phenomena, many synchronization based data mining algorithms [5], [31], [3], [32], [30], have been proposed. In this paper, we extend the notion of synchronization into the context of data stream, and propose a new data stream clustering algorithm, called SyncTree. Unlike traditional CF-based data abstraction by storing the data summary statistics, a new data representation structure, synchronized micro-clusters, are introduced for data abstraction. To support data stream clustering, a hierarchical tree is presented to dynamically store the micro-clusters at different levels of granularity depending upon the data recency. Finally, relying on the attributes of synchronized micro-clusters, the cluster evolution are investigated intuitively.

1.1. Contributions

By introducing synchronization into the context of data stream, SyncTree shows several attractive benefits for data stream clustering, most importantly:

- **Effective Data Abstraction:** Building upon the synchron-ization-inspired clustering, continuous examples in data streams can be intuitively summarized and represented as a set of synchronized micro-clusters. Instead of storing summary statistics in CF-based micro-clusters, the synchronized micro-clusters allows a better data abstraction as each micro-cluster captures its local cluster structure of streaming data. Meanwhile, synchronized micro-clusters allow handling concept drifts. Equipped with the effective data abstraction, SyncTree thus allows finding high quality clusters (cf. Section 3.2, Fig. 6, Tab. 3).

- **Flexible Online Data Maintenance:** Thanks to the powerful concept of synchronization, old micro-clusters can be intuitively summarized at a higher level by further clustering to fit any space constraint. Therefore, SyncTree maintains all micro-clusters in a hierarchical tree from fine-

grained to coarse-grained depending upon the data recency, and thus also supports feasible offline clustering on streaming data from the past (cf. Section 3.3).

- **Intuitive Evolving Analysis:** SyncTree is capable of handling evolving clusters with non-Gaussian distributions. Moreover, it provides an intuitive way to analyze the evolving cluster structure based on the attributes of synchronized micro-clusters (cf. Section 3.5, Fig. 8).

The remainder of this paper is organized as follows: In the following section, we briefly survey related work. Section 3 presents our algorithm in detail. Section 4 contains an extensive experimental evaluation. We finally conclude the paper in Section 5.

2. Related Work

During the past several decades, many algorithms were proposed for data stream clustering, such as [2], [42], [12], [19], [8], [13], [41], [20], [14], [7], [16], to mention a few. Here, we only provide a brief survey on some closely related algorithms.

To mine evolving data streams with limited memory, an efficient approach in most data stream clustering algorithms is to summarize large amounts of incoming data and store the statistic summaries in a suitable data structure. The four widely used data structures include cluster feature vector, prototype array, coreset trees and grids [34]. Specifically, the cluster feature (CF) vector, represented by three components, was firstly proposed in BIRCH [40] for data summarization. Afterwards, CF has been employed and extended in many data stream clustering algorithms [2, 6, 19]. For instance, the CF vector was extended in the CluStream algorithm and defined as micro-cluster, which has two additional components [2]. DenStream [6] introduced density-based clustering following DBSCAN and weighted cluster feature vector into data stream clustering. Similar to DenStream, the ClusTree algorithm [19] proposes to use a

weighted CF vector, which is kept into a hierarchical tree (R-tree family). Two parameters are used to build this tree: the number of entries in a leaf node and the number of entries in non-leaf nodes. ClusTree provides strategies for dealing with time constraints for anytime clustering, that is, the possibility of interrupting the process of inserting new objects in the tree at any moment. Therefore, ClusTree can also adapt itself to fast and slow streams. In fast streams, ClusTree aggregates similar objects in order to do a faster insertion in the tree. In slow streams, the idle time is used to improve the quality of the clustering. These data structures can help maintain valuable information on incoming data. However, only the statistic summaries instead of the original data are stored in memory. Therefore, the CF-based data abstraction will unavoidably result in losing information, which may be of great importance for the clustering result.

To represent the greater importance of recent data to capture the evolving nature of stream data, moving window models have been proposed, namely sliding windows, damped windows and landmark windows. In the sliding window model, data obey the rule of first in, first out (FIFO). The model has been adopted in several data stream algorithms, such as [4]. In contrast to sliding windows, the damped window model assumes an exponential decay function to weight the objects from the stream. The more recent an objects is, the larger weight it receives. This mechanism can be found in many density-based clustering algorithms [6, 9]. The landmark window model processes chunks of data separated by landmarks and has been employed in many models such as [24, 1, 2]. The drawback in using any fixed-length window scheme is that it is difficult to choose an ideal window size. Moreover, the sliding windows and damped windows assume that the objects have the same lifespan which they may not.

To better understand the dynamic behavior of stream data, changes of clusters are expected to be detected and tracked as time evolves. Aggarwal [2] proposes a framework towards diagnosing changes in data streams. It creates temporal and spatial velocity profiles, which are used to predict different types of data evolution (i.e. dissolution, coagulation and shift). MONIC [38], [37] was

proposed for modeling and tracking internal and external clustering transitions. The tracking mechanism is based on the degree of overlap between two clusters. Another algorithm for change detection of clusters is MEC [25]. The basic idea is to use different metrics to identify the temporal relationships among clusters and visualize the cluster evolution in bipartite graphs. However, how to provide insights into the nature of cluster changes is still a challenging problem.

Synchronization-based Data Mining: Inspired by the powerful concept of synchronization, many synchronization models and data mining algorithms have been recently proposed and shown many desirable properties [3, 32, 39, 28, 33, 27]. For instance, Arenas et al. [3] apply the Kuramoto model for network analysis, and study the relationship between topological scales and dynamic time scales in complex networks. Böhm et al. [5] propose a new synchronization-based clustering algorithms by introducing local synchronization and minimum description length principle. For bioinformatics, Shao et al. [27] propose a two-sided model to find co-regulated groups with co-clustering. Ying et al. [39] propose a new strategy to scaling up synchronization-inspired clustering. Shao et al. [31] further propose a synchronization-based subspace clustering algorithm via weighted interactions in relevant subspaces. In contrast to other algorithms, synchronization-based clustering algorithms (*i*) yield the high-quality clusters with arbitrarily shapes, (*ii*) are robust to noise and (*iii*) allows a natural hierarchical data analysis.

In contrast to traditional data stream clustering, here we will employ the synchronization principle to form a new form of micro-clusters, which allows maintaining the local data cluster structure, and also provides an intuitive way to analyze the cluster evolution.

3. Synchronization-based stream clustering

In this section, we present the SyncTree algorithm for clustering evolving data streams.

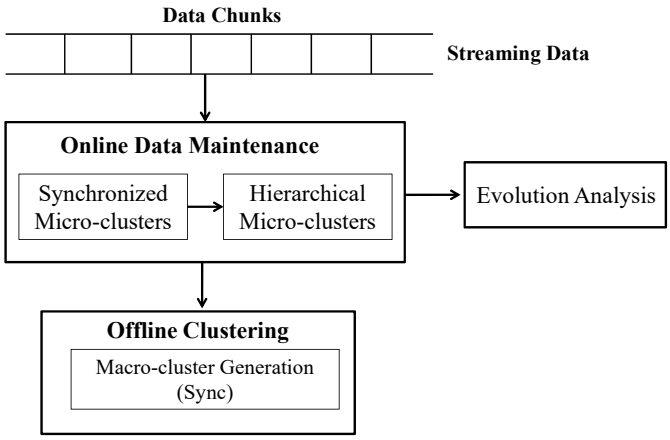


Figure 2: The overview of our proposed approach.

3.1. Overview

As introduced in Section 1, the basic idea of our algorithm is to propose a new data abstraction (i.e., synchronized micro-clusters) to capture the local cluster structure via synchronization-based clustering. To support data stream clustering, a hierarchical tree is further presented to dynamically store the micro-clusters at different levels of granularity depending upon the data recency. For offline clustering, the macro-clusters can be generated by simply using synchronization-based clustering. Finally, relying on the attributes of synchronized micro-clusters, the cluster evolution are investigated intuitively. Figure 2 gives the general framework of the proposed algorithm. In the following, we will first introduce the new proposed synchronized micro-clusters for data abstraction and then present a hierarchical tree to dynamically maintain online multi-level micro-clusters to fit available memory in Section 3.3. Based on the micro-clusters, Section 3.4 shows how to create macro-clusters. Finally we propose a descriptive way to analyze evolving clusterings in Section 3.5.

3.2. Synchronized Micro-clusters

Considering the potential limitations of CF-based data abstraction as illustrated earlier in Figure 1, here we introduce a new cluster structure to summarize

and represent the data based on synchronization-based clustering algorithms. The key idea of clustering approaches by synchronization [5, 29] is to regard each data object as a phase oscillator and simulate the dynamical behaviors of the objects over time. By the interaction with similar objects, the phase of an object gradually aligns with its neighborhood, resulting in a non-linear object movement driven by the local cluster structure. Finally, the objects in a cluster are synchronized together and have the same phase. For synchronization-based clustering algorithms, one salient feature is that the cluster formation is driven by the *local data structure*, and all objects in a cluster dynamically *move together* (a synchronized object). Therefore, it provides an intuitive way to use the synchronized object (called synchronized micro-cluster) to represent the local similar objects. This means that although the size of the new data set is smaller than the original data set, the cluster structure is still maintained and represented by those micro-clusters and outliers (a special form of micro-cluster) in the new data set. In addition, synchronization-based clustering algorithms are robust to noise or outlier objects. Thus, global noise or outlier objects in the data streams can be easily identified, and they might be further viewed as cluster objects in the future. More importantly, the powerful concept synchronization allows iterative clustering. Namely, the existed synchronized micro-clusters can be viewed as new objects (summarized objects) and cluster with the incoming instances again (like the additivity property of CF-based micro-cluster). This property leads our proposed algorithm not only allows maintaining online micro-clusters effectively, but also make it is suitable to handle the evolving changes in data streams. These benefits, will be illustrated and evaluated in the following sections.

Typically, a synchronization-based clustering algorithm needs three definitions to simulate a dynamic clustering process. In the following, we first give a short summary of all necessary definitions.

Definition 1 (ϵ-**Range Neighborhood**) Given a data set D, $\epsilon \in \mathcal{R}$ and $x \in D$, the ϵ-range neighborhood of an object x, denoted by $N_\epsilon(x)$, is defined

as:

$$N_\epsilon(x) = \{y \in \mathcal{D} | dist(y, x) \leq \epsilon\} \tag{1}$$

where $dist(y, x)$ is a metric distance function. The Euclidean distance is used in this study.

Definition 2 (**Interaction Model**) Let $x \in \mathcal{R}^d$ be an object in the data set $\mathcal{D}$, and x_i be the i-th dimension of the data object x, respectively. With an ϵ-range neighborhood interaction, the dynamics of each dimension x_i of the object x is defined as:

$$x_i^{t+1} = x_i^t + \frac{1}{|N_\epsilon(x^t)|} \cdot \sum_{y \in N_\epsilon(x^t)} \sin(y_i^t - x_i^t) \tag{2}$$

where $sin(\cdot)$ is the coupling function, which is used in almost synchronization-based models. x_i^{t+1} describes the renewal value of the i-th dimension of object x at $t = (0, \ldots, T)$ during the dynamic clustering.

To characterize the level of synchronization between oscillators during the synchronization process, a cluster order parameter R_c is defined to measure the coherence of the local oscillator population.

Definition 3 (**Cluster Order Parameter**) The cluster order parameter R_c is used to terminate the dynamic clustering by investigating the degree of local synchronization, which is defined as:

$$R_c = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{|N_\epsilon(x)|} \sum_{y \in N_\epsilon(x)} \mathbf{e}^{-||y-x||}. \tag{3}$$

The dynamic clustering terminates when $R_c(t)$ converges, which indicates local phase coherence. At this moment, all cluster objects have the same phase (common location in the feature vector space).

For dynamic clustering, each data object is viewed as a phase oscillator and has its own phase (feature vector) at the beginning. As time evolves, each object interacts with its ϵ-range neighborhood according to the interaction model (Eq. (2)). To guarantee a stable interaction of each object, we specify ϵ with the average value of the k-nearest neighbor distance determined from a sample of

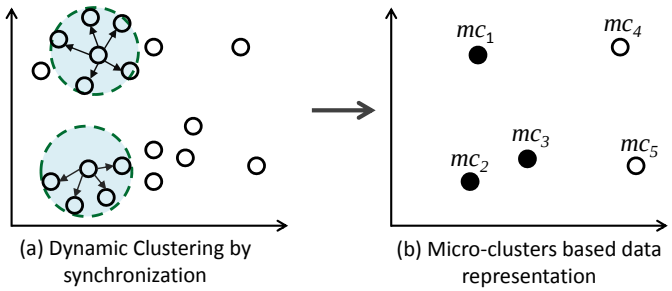


Figure 3: Illustration of synchronization-based clustering.

data from the stream as suggested in [29], where k is often small, ranging from 2 to 8. In this paper, we set $k = 3$. A small interaction range makes that the cluster structure for each data chunk is maintained although some regular examples may be regarded as noise sometimes. However, the potential noise examples if existing, pose no effect on clustering results since they will be re-clustered again during the data abstraction at a higher level, which we will introduce in next Section 3.4. The effect of the parameter is further evaluated in Section 4.4. With local interactions, the phase of an object gradually aligns with its neighborhood driven by the local cluster structure for each time step. Finally, the cluster order parameter is used to terminate dynamic clustering when R_c converges. As illustrated in Figure 3, each object interacts with its similar objects, and finally all locally similar objects are synchronized together and form micro-clusters. Meanwhile, since potential outlier objects (may become cluster objects later) do not interact well with other objects during the dynamic clustering, they maintain their original values and thus are easily identified.

Definition 4 Synchronized Micro-cluster (SM) For n objects synchronized together after dynamic clustering, we define the synchronized micro-cluster as a tuple $SM = (n, SC, CR, CD, T)$. SC indicates the synchronized coordinate of n objects after clustering, CR and CD are the cluster radius and cluster density, respectively. T is the average arrival time stamp of objects

synchronized as a micro-cluster. Specifically,

$$CR = \max_{x \in C} \{dist(x, SC)\} \tag{4}$$

$$CD = \frac{\sum_{x \in C} dist(x, SC)}{n} \tag{5}$$

where C is the set of objects from a synchronized micro-cluster. Using synchronized micro-clusters, the incoming data can not only be intuitively summarized, it also reflects the intrinsic cluster structure and gives a good data representation (See Figure 3). Since our approach works on data chunks, for each Synchronized Micro-cluster, it is initialized via synchronization-based clustering. These values (n, SC, CR, CD, T) are obtained from each data chunk, respectively. In contrast to CF-based data abstraction using summary statistics, synchronized micro-clusters allow representing non-Gaussian clusters, and more importantly, it envelops the local cluster structure and thus supports a high quality of clustering. In addition, it is also different from traditional spiking information representation in evolving connectionist systems (ECoS) [18, 35, 26, 17]. In ECoS, incoming instances that have a distance to an existing node (cluster centre, rule node) less than a certain threshold are allocated to the same cluster. Instances that do not fit into existing clusters, form new clusters. Cluster centres are continuously adjusted according to new data samples, and new clusters are created incrementally. Therefore, the update is k-means style and ECoS aims to learn evolving local models from these clusters, and associates a local output function for each cluster represented in a connectionist structure.

3.3. Online Micro-cluster Maintenance

In contrast to traditional data stream clustering, in this study, the incoming data is split into small-size data chunks sequentially, and each data chunk is summarized as a set of synchronized micro-clusters in a batch fashion. In this study, we set $chunkSize = 50$. Generate micro-clusters on small-size data chunk is to ensure the objects in the data chunk are equally important, and the cluster structure does not evolve. The sensitivity of $chunkSize$ is further evaluated in

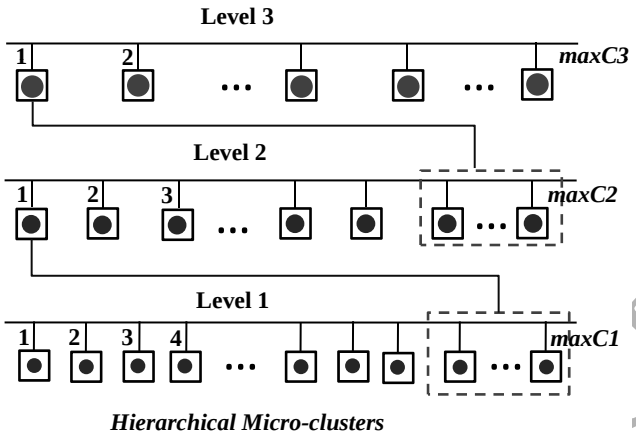


Figure 4: SyncTree: a hierarchical tree for online micro-clusters maintenance. $maxC1, maxC2, maxC3$ indicate the defined maximum number of instances for the level one to three, respectively.

Section 4.4. This mode has the attractive property that the derived clusters are time-independent and we thus do not need to specify a sliding window or decay function to characterize the greater importance of recent data as in CF-based micro-clusters. Specifically, as incoming examples continuously arrive and form small-size data chunks, the derived micro-clusters in two sequential data chunks naturally receive different time stamps. The micro-clusters in the same data chunk will have nearly the same time stamp, but we still retain their separate time stamps (i.e. micro-clusters have their own time stamps and they are mutually time independent).

In case of limited memory, we cannot store all original micro-clusters infinitely. It is interesting to note that the micro-clusters can be further summarized by synchronization-based clustering as each micro-cluster can intuitively be viewed as a new data object. This means that we can store all micro-clusters at different levels of granularity depending upon the recency. Specifically, older micro-clusters are stored with a coarse granularity while the new micro-clusters are stored with a fine granularity. For example, the two micro-clusters (mc_2 and mc_3) in Figure 3 can be further summarized as a new micro-cluster. There-

Table 1: The formal description of cluster change based on the attributes of synchronized clusters.

Types	**Notation**	**Indicator**
Merge	$(X^1, \cdots, X^k) \Rightarrow Y$	$\forall i = (1, \cdots, k):$ $Y = BestMatch(X^i) \wedge dist(SC(X^i) \wedge SC(Y)) \leq CR(Y)$
Split	$X \Rightarrow (Y^1, \cdots, Y^k)$	$\forall i = (1, \cdots, k):$ $Y^i = BestMatch(X) \wedge dist(SC(Y^i), SC(X)) \leq CR(X)$
Expand	$X \uparrow Y$	$Y = BestMatch(X) \wedge CR(Y) > (1+\gamma) \cdot CR(X)$ $\wedge\ dist(SC(Y), SC(X)) < (CR(X) + CR(Y))$
Shrink	$X \downarrow Y$	$Y = BestMatch(X) \wedge CR(Y) < (1-\gamma) \cdot CR(X)$ $\wedge\ dist(SC(Y)SC(X)) < (CR(X) + CR(Y))$
Denser	$X \rightarrow Y$	$Y = BestMatch(X) \wedge CD(Y) > (1+\gamma) \cdot CD(X)$ $\wedge\ dist(SC(Y), SC(X)) < (CR(X) + CR(Y))$
Sparser	$X \leftarrow Y$	$Y = BestMatch(X) \wedge CD(Y) < (1-\gamma) \cdot CD(X)$ $\wedge\ dist(SC(Y), SC(X)) > (CR(X) + CR(Y))$
Shift	$X \rightarrow Y$	$Y = BestMatch(X)$ $\wedge\ 0 < dist(SC(Y), SC(X)) < (CR(X) + CR(Y))$
Disappear	$X \rightarrow \odot$	$Y = BestMatch(X)$ $\wedge\ dist(SC(Y)SC(X)) > (CR(X) + CR(Y))$
New	$\odot \rightarrow Y$	$X = BestMatch(Y)$ $\wedge\ dist(SC(Y), SC(X)) > (CR(X) + CR(Y))$

fore, to support efficient and effective online data maintenance, we propose a tree structure, which is a hierarchical tree storing all micro-clusters to fit available memory. Figure 4 illustrates the hierarchical tree structure to maintain online micro-clusters, where the finest micro-clusters are maintained in the first level. If out of memory, the oldest $ChunkSize$ micro-clusters are further summarized and stored at a higher level. Owing to the desirable properties of synchronization-based clustering, the old micro-clusters are well summarized at a higher level without losing local cluster structure. Building upon the online hierarchical micro-clusters maintenance, our algorithm can fit any space constraints by iterative clustering.

Although our model provides analysts a flexible way to store data, and inves-

tigate the change of cluster structure at any time interval, the old micro-clusters will be less meaningful, especially when the cluster structure evolves. Therefore, for old micro-clusters ($T_{current} - T_{generation}(SM) > T_\lambda$), they are purged, and we only store a representative snapshot of the old data (synchronized micro-clusters) to disk if necessary.

3.4. Macro-cluster Generation

The online micro-cluster maintenance captures the local cluster structure of data streams at a detailed level. However, in order to get the final clusters, we apply offline clustering, again via synchronization-based clustering. Macro-cluster generation can be easily performed at any given time horizon (*not limited to time horizon from the current clock, but any time interval*). The difference between the generation of micro-clusters and macro-clusters is only the granularity, where a macro-cluster can be thought of as a higher data abstraction of many micro-clusters.

Here, for offline clustering, a larger interaction range should be specified to yield meaningful macro-clusters. The optimal interaction range can be obtained using some model selection methods (e.g. minimum description length principle), however, they are time consuming as we need cluster the data many times with different interaction ranges. Here offline clustering in many algorithms like DenStream [6], we use $\epsilon^* = off \times \epsilon$, where $off = (1, 5)$. Usually, off=1.5 yields reasonable results according to our empirical experiments. The effect of the interaction range on clustering is further evaluated in Section 4.4.

3.5. Evolution Analysis of Clusters

For data streams, the cluster structure often evolves over time. In this study, we model and trace evolving clusters by investigating the macro-clusters between any two arbitrary length sequential data chunks, which is the same with MONIC [38], MEC [25] and TRACER [15]. It differs largely as we use the derived synchronized micro-clusters to investigate the changes instead of point to point comparison between two data chunks in these existing approaches.

Formally, let D_t and D_{t+1} be two sequential data chunks in a data stream D, and C_t and C_{t+1} be the clusterings at time step t and $t+1$, respectively. The objective is to analyze whether or not the clusters embedded in D_t and D_{t+1} have changed over time. In Fig. 5, we depict some changes of clusters at two timepoints. We can observe that the cluster change can be easily analyzed based on the attributes of mi(a)cro-clusters directly.

Formally, suppose that we have two clusterings C_t=$\{C_t^1, \cdots, C_t^m\}$ and $C_{t+1}^1, \cdots, C_{t+1}^n$ at any two sequential time points. Before investigating cluster changes, some definitions are given.

Definition 5 (**Best Cluster Match**) Let X be a cluster in the clustering C_t at t and Y be a cluster in C_{t+1} at $t+1$. Y is "the best match" for X in C_t if

$$Y = \arg\min_{Z \in C_{t+1}} dist(X, Z) \tag{6}$$

Relying on the description of each cluster ($SM = (n, SC, CR, CD, T)$) based on our definitions, the evolving cluster structure can be easily analyzed at any given time intervals. Taking the merge of clusters as an example (Figure 5), given two clusterings (C_t and C_{t+1}) by clustering on micro-clusters, for each cluster in C_t, we observe that the two clusters (C_{R_1} and C_{R_2}) are best match with the cluster C_{R_3} in C_{t+1}. Meanwhile, the two clusters C_{R_1} and C_{R_2} in C_t are located in the cluster C_{R_3} in C_{t+1}. It is thus intuitive to know that the two clusters at time step t are merged together as a new cluster in C_{t+1}. Formally, we can quality the clustering $(X_1, \cdots, X_k)$ at time step t merged as new clustering Y at time step $t+1$: $(X^1, \cdots, X^k) \Rightarrow Y$, where $\forall i = (1, \cdots, k)$: $Y = BestMatch(X^i) \wedge dist(SC(X^i), SC(Y)) \leq CR(Y)$. Similarly, it is easy to formalize the other types of cluster evolution such as split, expand, shrink, shift, denser, sparser, disappear and new clustering emerges in a descriptive way. The formal description of distinct types of cluster change based on the attributes of synchronized clusters is indicated in Table 1. Based on the definitions, the evolving cluster structure can be easily analyzed based on micro/macro-clusters intuitively.

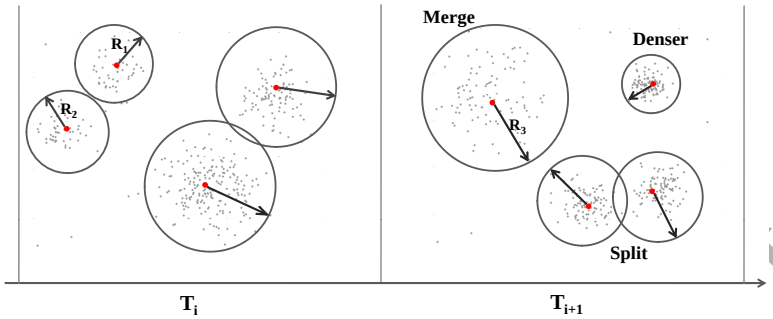


Figure 5: Illustration of evolving clusters. The cluster structure evolution can be investigated based on the attributes of the micro-clusters at different time steps (e.g. cluster radius and cluster density).

For analyzing evolving clusterings, as usual in [38], [25], the window size is first need to specified to compare two clusterings between the two windows of data. The threshold (in this study λ) is usually used to determine whether two clusterings are significant different. In principle, the larger window size is, the less cluster changes can be found as the clusterings between the two windows of data are more stable, and vice versa. The parameter λ in Table 1 determines the change threshold according to the requirements of users. For instance, $\lambda = 0.3$ indicates that if the change of measure values (e.g. cluster radius CR or cluster density CD, etc.) at time step t exceeds 30% compared to those values at time step $t + 1$, we assume the cluster(s) is (are) evolved. Generally, the lower value of λ is, the more subtle changes can be detected.

4. Experimental Evaluation

In this section, we evaluate our proposed algorithm SyncTree on both synthetic and real world data streams. We will demonstrate the superiority of SyncTree over other comparing algorithms from both the cluster quality and the efficiency.

4.1. Experimental Setup

4.1.1. Data Sets

We evaluate SyncTree on synthetic and five genres of real-world data streams: Spam, Weather, Electricity, Forest, and Sensor, which are all publicly available. The data is streamed at constant speed in data input order as in [19, 2, 6].

Spam Filtering Data[1] is a collection of 9324 email messages derived from Spam Assassin collection. Each email is represented using 500 attributes in the boolean bag-of-words approach.

Nebraska Weather Data[2] is the part of the U.S. National Oceanic and Atmospheric Administration, which has compiled weather measurements from over 9000 weather stations worldwide. This data set includes 50 years (1949-1999) of daily measurements with eight features such as wind speed, pressure, temperature, among others from Offutt Air Force Base in Bellevue, Nebraska.

Electricity Data[3] contains 45,312 instances and was collected from the Australian New South Wales Electricity Market every 5 minutes. In this market, prices are not fixed but affected by demand and supply. The class label identifies the change in the price related to a moving average over the last 24 hours.

Covtype Data[4] consists of 581,012 instances describing seven forest cover types on a 30×30 meter grid obtained from the US Forest Service (USFS) Region 2 Resource Information System (RIS).

Sensor Data[5] collects information (temperature, humidity, light, and sensor voltage) from 54 sensors deployed in the Intel Berkeley Research Lab over a two month period (one reading per 1-3 minutes). In total, it contains 2,219,803 instances.

[1] http://spamassassin.apache.org/
[2] http://users.rowan.edu/ polikar/research/nse/
[3] http://moa.cms.waikato.ac.nz/datasets/
[4] http://archive.ics.uci.edu/ml/datasets/Covertype
[5] http://db.csail.mit.edu/labdata/labdata.html

4.1.2. Selection of comparison methods

To extensively evaluate the proposed algorithm SyncTree, we compare its performance to several representatives of data stream clustering paradigms.

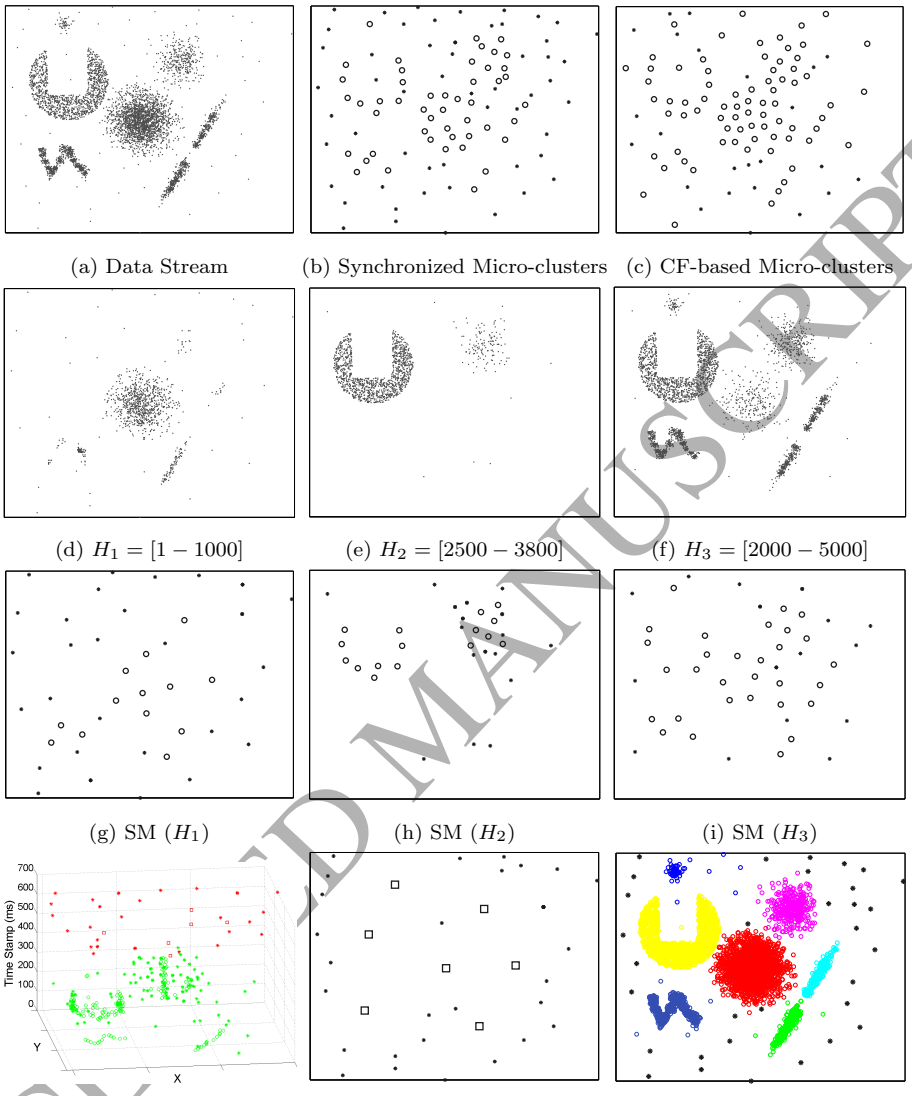
CluStream [2]: is typical data stream clustering algorithm by maintaining summary statistics (CF-based micro-clusters). These micro-clusters are stored at snapshots in time following a pyramidal pattern, and macro-clusters are obtained using an offline K-Means algorithm.

DenStream [6]: is an extension of DBSCAN in the context of data streams. Inherited from density-based clustering, it allows for investigating arbitrarily shaped clusters and outliers over time.

ClusTree [19]: propose a hierarchical tree to deal with time constraints for anytime clustering, which allows itself to adapt time-changing speed of data streams. It improves the quality of the clustering to slow streams and aggregates similar objects into a higher level to support any space clustering.

4.1.3. Parameter Setting

For fair comparison as much as possible, the parameters of the comparing algorithms are specified as follows. For CluStream and ClusTree, we set the important parameter K (i.e. the number of clusters) as the number of actual classes. Concerning the other parameters, they use the parameters suggested by authors. For all the comparing algorithms, one parameter called "radiusFactor" (a factor in Cluster Feature to determine whether an incoming instance will be assigned to the existing micro-clusters or should generate a new micro-cluster) is also fine tuned as it sensitive to the final results. The "radiusFactor" parameter, in conjunction with other parameters (fixated K and default parameters) to obtain the best results for all comparing algorithms. In addition, DenStream and SyncTree also try to yield comparable number of clusters for comparison. We have implemented SyncTree in Java, which is available at `http://dm.uestc.edu.cn/wp-content/uploads/code/SyncTree.rar`. CluStream, DenStream and ClusTree algorithms are implemented in the Java-based



(a) Data Stream (b) Synchronized Micro-clusters (c) CF-based Micro-clusters

(d) $H_1 = [1 - 1000]$ (e) $H_2 = [2500 - 3800]$ (f) $H_3 = [2000 - 5000]$

(g) SM (H_1) (h) SM (H_2) (i) SM (H_3)

(j) Hierarchical SM (k) Macro-clusters (l) Clustering results

Figure 6: Illustration of synchronized micro (macro)-clusters in a synthetic data stream with noise.

MOA framework[6]. All experiments have been performed on a workstation with

[6]Available at http://moa.cms.waikato.ac.nz/

2.4 GHz CPU and 8 GB RAM.

4.1.4. Evaluation Metrics

To quantitatively evaluate SyncTree, we use the following performance metrics:

- **Quality**. We evaluate the cluster quality of SyncTree in terms of purity.
- **Efficiency**. We measure the run time as efficiency.
- **Sensitivity**. We test how sensitive the cluster quality against to parameter settings.

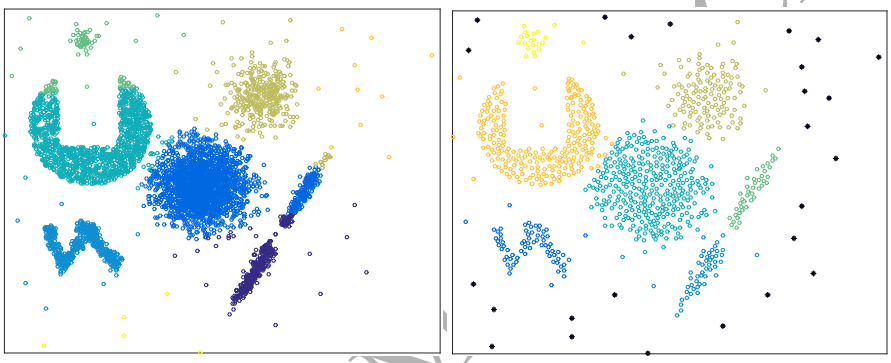
4.2. Proof of Concept

We start the evaluation with two-dimensional synthetic data streams to facilitate presentation and demonstrate the properties of SyncTree.

4.2.1. Synchronized Micro-clusters Well Preserve the Intrinsic Cluster Structure.

We first evaluate whether SyncTree can dynamically capture the intrinsic cluster structure on data streams based on synchronized micro-clusters. Here we first create a synthetic data set consisting of seven differently shaped clusters with noise. Afterwards, clusters are ordered sequentially, and for each cluster, the objects are selected randomly from it respectively, to generate a data stream. Finally, all outliers are randomly injected into the entire data stream. Figure 6(a) depicts the entire stream, and Figure 6 (b)-(c) show the derived synchronized micro-clusters for SyncTree and CF-based micro-clusters. 6 (d)-(f) further depicts the snapshots of the data stream at different time intervals (H_1, H_2 and H_3). Here the time intervals denote the corresponding order of objects in time (e.g. H_1 means the first object to the thousandth object). It is interesting to note that the derived micro-clusters allows preserving the cluster structure of the data set, even the global outliers are well pinpointed. Figure 6(k) and (l) further show the macro-clusters and final clustering results on the

whole data stream. For comparison, Figure 7 further illustrates the clustering results of SyncTree and BIRCH on the same cluster features. We can observe that SyncTree allows a better clustering results since it is more robust to noisy points and allows finding non-Gaussian clusters effectively. By comparing the SyncTree on the original feature vectors and cluster features, we can see that the latter is more sensitive to noise. These results further show that the synchronized micro-clusters provide a better data abstraction and support a higher quality of clustering.



(a) BIRCH (b) SyncTree

Figure 7: The clustering results of SyncTree and BIRCH on cluster features.

4.2.2. Effective Micro-clusters online maintenance.

After summarizing continuously arriving examples as synchronized micro-clusters, SyncTree stores all of them in a hierarchical tree at different levels. Figure 6(j) shows the two levels of microclusters. For better illustration, Table 2 shows the micro-clusters stored at different levels on five real-world data sets. The maximum number of micro-clusters at each level is flexible determined by users. If there is sufficient memory, all micro-clusters can be kept with finest granularity. Otherwise, old micro-clusters are further summarized and stored at a higher level with coarser granularity. Interestingly, building upon the online micro-cluster maintenance via the hierarchical tree, SyncTree also supports clustering for any given time intervals in the past (Figure 6 (g)-(h)

Table 2: Evaluation of quality of hierarchical synchronized micro-clusters on real-world data streams.

Data	#Obj	#Dim	#Class	Levels	#SM	Purity	Speed (pps)
Spam	9324	500	2	L_1	734	0.966	352
NEweather	18,159	8	2	L_1	1513	0.909	3705
Electricity	45,312	8	2	L_1	4993	0.943	24262
				L_2	1433	0.814	
Covtype	581,012	54	7	L_1	4973	0.966	5461
				L_2	4973	0.918	
				L_3	4969	0.870	
				L_4	4552	0.865	
Sensor	2,219,803	5	54	L_1	49912	0.965	22805
				L_2	49805	0.920	
				L_3	49885	0.863	
				L_4	23764	0.765	

shows the results for our earlier time interval examples). This is not possible in DenStream or CluStream where micro-clusters are only available in a time horizon from current time. The reason is that CF-based micro-clusters are summary statistics of objects with different time stamps, that are thus time-dependent.

4.2.3. Concept Drift Handling.

Here a simple fictitious synthetic data is generated, where the first 1000 points are formed as a Gaussian cluster (see Figure 8(a)), and then three new emerging clusters with different shapes are produced with later 200 points (Figure 8(b)). We evaluate whether different algorithms allow handling the emerging concepts reasonably. Figure 8(c)-(f) plots the micro-clusters stored by different algorithms after 200 objects have reached. We observe that CF-based algorithms are difficult to handle the evolving clusters as the mircro-clusters cannot

Table 3: Performance of different data stream clustering algorithms on real-world data sets.

Data	Methods	Purity		
		H=1000	H=500	H=200
Spam	**SyncTree**	**0.858±0.107**	**0.875±0.132**	**0.894±0.125**
	CluStream	0.781±0.246	0.811±0.160	0.840±0.153
	DenStream	0.773±0.120	0.813±0.179	0.802±0.241
	ClusTree	0.843±0.152	0.857±0.147	0.867±0.144
Electricity	**SyncTree**	0.607±0.054	0.624±0.053	**0.684± 0.079**
	CluStream	**0.618±0.060**	**0.629±0.079**	0.676 ± 0.088
	DenStream	0.607 ± 0.063	0.631±0.080	0.651 ± 0.109
	ClusTree	0.608 ± 0.068	0.628±0.071	0.674 ± 0.084
NEweather	**SyncTree**	**0.854±0.062**	**0.844±0.058**	**0.831± 0.065**
	CluStream	0.734 ± 0.054	0.747 ± 0.057	0.749 ± 0.087
	DenStream	0.796 ± 0.102	0.805±0.093	0.781 ± 0.0934
	ClusTree	0.733 ± 0.054	0.746±0.057	0.749 ± 0.087
Covtype	**SyncTree**	0.809±0.135	**0.850±0.119**	**0.889±0.132**
	CluStream	**0.812±0.088**	0.819±0.090	0.859±0.089
	DenStream	0.772±0.120	0.842±0.081	0.866±0.100
	ClusTree	0.809±0.093	0.787±0.099	0.811±0.114
Sensor	**SyncTree**	**0.839±0.096**	**0.837±0.106**	**0.873±0.096**
	CluStream	0.419 ± 0.170	0.532 ± 0.213	0.490 ± 0.182
	DenStream	0.648 ± 0.170	0.629 ± 0.162	0.630 ± 0.191
	ClusTree	0.486±0.167	0.544±0.158	0.531 ± 0.162

reflect the current data cluster structure well. The main reason behind it is because the new incoming examples assigned to micro-clusters depend on the statistic summary of previous examples in streaming data. However, for SyncTree, it allows a better adaption to the concept drift (Figure 8(c)) as the derived micro-clusters can preserve the local data structure with new incoming data (as the figure 1 illustrated in the introduction section.) and also the micro-clusters are time independent. Beyond, SyncTree also allows tracing evolving clusters by investigating the attributes of the synchronized micro/macro-clusters.

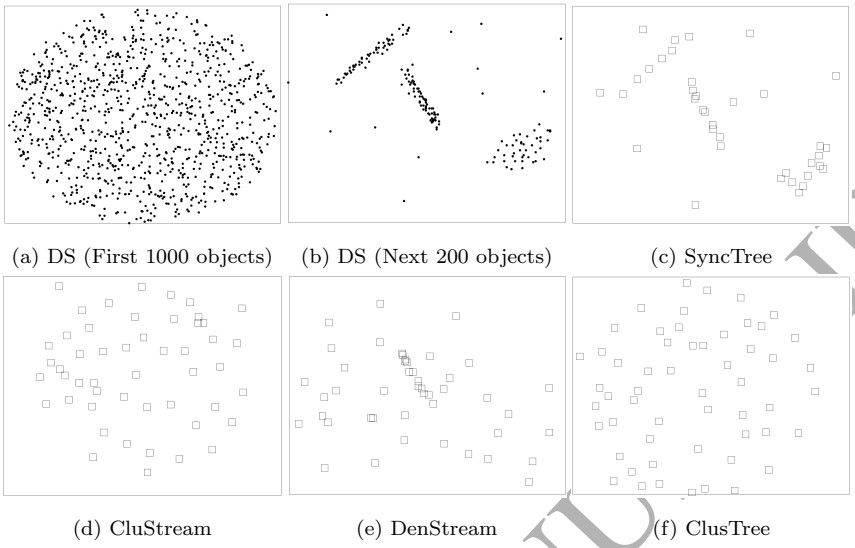
Table 4: Performance of different data stream clustering algorithms on real-world data sets.

Data	Methods	Time (ms)			Memory(M)
		H=1000	H=500	H=200	
Spam	**SyncTree**	**51028**	**46494**	**33847**	70.90
	CluStream	1002348	655379	92792	398.82
	DenStream	2999029	248423	202363	185.30
	ClusTree	986757	643691	89636	70.98
Electricity	**SyncTree**	2402	2282	2215	34.70
	CluStream	6637	5485	2385	33.42
	DenStream	13094	13794	17486	65.57
	ClusTree	**1193**	**976**	**978**	38.56
NEweather	**SyncTree**	2402	2282	2215	89.89
	CluStream	3896	3506	2594	34.71
	DenStream	3995	3789	3557	28.28
	ClusTree	**1557**	**1267**	**1506**	33.42
Covtype	**SyncTree**	**24123**	**22186**	**19231**	57.48
	CluStream	242927	226695	114008	455.35
	DenStream	1096678	3095871	7916931	136.14
	ClusTree	92379	70074	59108	537.56
Sensor	**SyncTree**	152230	129700	107051	265.82
	CluStream	385243	313264	177277	214.14
	DenStream	784905	772968	623023	67.62
	ClusTree	**35753**	**29762**	**26732**	511.84

4.3. Clustering Quality Evaluation

4.3.1. Evaluation of Micro-clusters

In this section, we evaluate the cluster quality of micro-clusters at different levels. In general, SyncTree stores all micro-clusters with finest granularity if memory is sufficient. In order to evaluate the cluster quality of micro-clusters at different levels, we reduce the granularity by manually setting the maximum number of micro-clusters to 5,000 for Spam, Electricity and Covtype data sets. For Sensor data, due to its large volume, we set it to 50,000 per level. Here,



(a) DS (First 1000 objects) (b) DS (Next 200 objects) (c) SyncTree

(d) CluStream (e) DenStream (f) ClusTree

Figure 8: Illustration of concept drift handling based on different algorithms.

we give a large number of derived synchronized micro-clusters (SM) to better illustrate the micro-clusters from fine-grained to coarse-gained granularity, and the corresponding qualities of these SMs at different levels. In real-world situations, we can stored a flexible number of micro-clusters by hierarchical clustering depending on user's application. Table 2 shows the experimental results on four real-world data sets, including the number of micro-clusters at each level, the cluster quality (purity) and the speed (the number of objects per second (pps) that can be processed by SyncTree). As expected, the cluster quality of micro-clusters at lowest levels is best, and decreases for higher levels where the granularity is reduced. However, even at the fourth level, the derived micro-clusters still yield high cluster purity (86.5% and 76.5% for Covtype data and Sensor data, respectively). From the table, in contrast to existing data stream clustering algorithms, we can see that SyncTree allows supporting data stream clustering at different levels, and more importantly, shows a high cluster quality.

4.3.2. Comparison to State-of-the-art Approaches

In this section, we compare SyncTree to state-of-the-art algorithms. Table 3 gives the overall clustering results on different real-world data streams at different time horizons H (given as the number of objects from the current time). The results show the superior quality of SyncTree, as it relies on a good data representation in synchronized micro-clusters and is is robust to outliers. From the table, we can see that the purities of derived cluster on different time horizons are good and outperform other comparing algorithms including CluStream, DenStream and CluStream. Especially, for Spam and Sensor data, SyncTree shows a great improvement. In addition, SyncTree is also time efficient as the synchronized clusters do not need to update any weights and summary statistics for each incoming object as is the case for CF-based micro-clusters. Moreover, the computational time and used memory of different algorithms are further reported in Table 4. We can see that the running time and memory consumption of SyncTree is comparable to other CF-based algorithms. For the Spam and Covtype data sets, SyncTree is faster than comparing algorithms, which further indicates that SyncTree is time efficient for high-dimensional data sets since the two data sets are high dimensional.

4.4. Sensitivity Analysis

In this section, we perform sensitivity analysis for SyncTree with respects to the two key input parameters: interaction range ϵ, and chunk size $ChunkSize$.

Interaction Range: Here, we need to specify the parameter value of the interaction range ϵ for online micro-cluster generation. For this, we examine values for ϵ varying from 0.1 to 0.5. As we can see in Figure 9, SyncTree is generally very stable, but for values of 0.3 or larger, the quality decreases as different clusters are synchronized together by the overly large interaction range.

Chunk Size: We assess the effect of the chunk size on clustering with different settings ranging from 20 to 500. As shown in Figure 10, the clustering results are largely unchanged until $chunkSize = 150$. We therefore increased the chunk

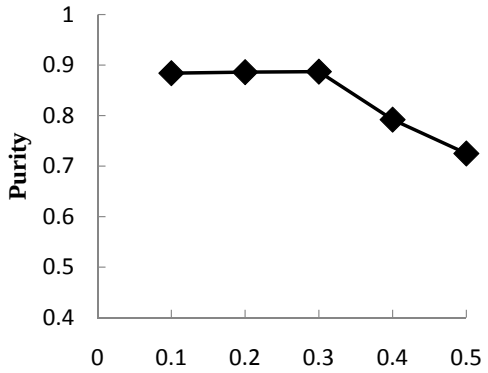


Figure 9: The sensitivity analysis with interaction range ϵ.

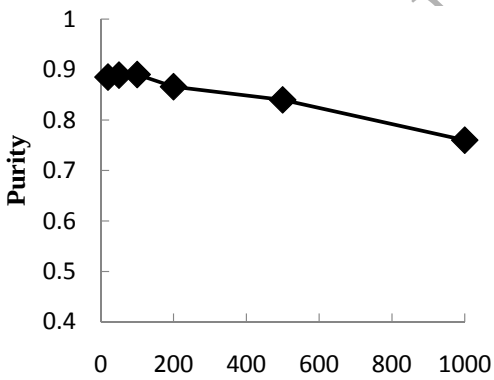


Figure 10: The sensitivity analysis with chunk size *chunkSize*.

size parameter also to large values of up to 500 points in the stream, where we observe a slow decrease in purity. Naturally, we can expect streams, especially those with extreme changes in cluster distribution, to benefit from smaller chunk sizes which corresponds to a fine starting granularity. As memory is exhausted, SyncTree automatically adapts and aggregates the data at a coarser granularity for higher levels in the tree.

4.5. Evolution Analysis

In this section, we discuss SyncTree's ability to track and analyze the cluster evolution. Taking the Electricity data set as an example, with $\lambda = 0.3$, we

identified 19 changes in total, where 15 were shifts, two indicated sparser clusters and one a cluster becoming denser. These changes mainly occurred with major changes of the price. For smaller values of λ, more subtle changes can be detected, but also brings more insignificant changes. Similarly, for the Spam data set, 52 changes are found, of which most are "shift", and four new clusters are generated, which well correspond to the time points at which new "spam" or "legitimate" emails arrived.

5. Discussion and Conclusion

Considering the potential limitations of current CF-based methods (e.g. DenStream, CluStream, ClusTree), this study aims to identify evolving clusters by using synchronized prototypes instead of statistic summary in CF. We hope the new data representation structure will give some contributions to the context of data stream clustering. This study might be of great importance as SyncTree has some remarkable properties such as effective data abstraction (local data structure is well preserved in each micro-cluster), efficient online data maintenance (allows storing a flexible number of fine-grained to coarse-grained micro-clusters depending on user's application and memory limitation), and supporting to cluster data stream between any two timestamps in the past (due to derived micro-clusters are time independent). Moreover, relying on the attributes of synchronized-based data representation (micro/macro-clusters), the evolving clusters can be intuitively investigated in a descriptive fashion. In comprehensive experiments, we have shown that SyncTree achieves good overall performance compared to state-of-the-art algorithms in terms of both effectiveness and efficiency. However, like traditional window-dependent evolving analysis, the evolution analysis of derived clusters heavily depends on the size of data chunk. In addition, since our algorithm works on vector data like traditional data stream clustering algorithms, it cannot handle data sets with different attribute types, such as categorical data. In future research, we aim to extend the notion of synchronization for distributed data stream clustering.

6. Acknowledgments

This work is supported by the National Natural Science Foundation of China (61403062, 61433014, 41601025), Science-Technology Foundation for Young Scientist of SiChuan Province (2016JQ0007), Fok Ying-Tong Education Foundation for Young Teachers in the Higher Education Institutions of China (161062), National key research and development program (2016YFB0502300) and the Sichuan Provincial Soft Science Research Program (2017ZR0208).

References

[1] M. R. Ackermann, M. Märtens, C. Raupach, K. Swierkot, C. Lammersen, and C. Sohler. Streamkm++: A clustering algorithm for data streams. *ACM Journal of Experimental Algorithmics*, 17(1), 2012.

[2] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on very large data bases*, pages 81–92, 2003.

[3] A. Arenas, A. Diaz-Guilera, and C. J. Pérez-Vicente. Synchronization reveals topological scales in complex networks. *Physical review letters*, 96(11):114102, 2006.

[4] B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan. Maintaining variance and k-medians over data stream windows. In *Proceedings of the twenty-second ACM SIGMOD SIGACT-SIGART symposium on Principles of database systems*, pages 234–243, 2003.

[5] C. Böhm, C. Plant, J. Shao, and Q. Yang. Clustering by synchronization. In *ACM SIGKDD*, pages 583–592, 2010.

[6] F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *SDM*, volume 6, pages 328–339, 2006.

[7] J.-Y. Chen and H.-H. He. A fast density-based data stream clustering algorithm with cluster centers self-determined for mixed data. *Information Sciences*, 345:271–293, 2016.

[8] L. Chen, L.-J. Zou, and L. Tu. A clustering algorithm for multiple data streams based on spectral component similarity. *Information Sciences*, 183(1):35–47, 2012.

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

[10] J. de Jesús Rubio. Usnfis: uniform stable neuro fuzzy inference system. *Neurocomputing*, 262:57–66, 2017.

[11] J. de Jesús Rubio. Error convergence analysis of the sufin and csufin. *Applied Soft Computing*, 2018.

[12] J. Gama, P. P. Rodrigues, and L. Lopes. Clustering distributed sensor data streams using local processing and reduced communication. *Intelligent Data Analysis*, 15(1):3–28, 2011.

[13] A. Guerrieri and A. Montresor. Ds-means: Distributed data stream clustering. In *European Conference on Parallel Processing*, pages 260–271. Springer, 2012.

[14] S. Guha and N. Mishra. Clustering data streams. In *Data Stream Management*, pages 169–187. Springer, 2016.

[15] B. Hawwash and O. Nasraoui. Stream-dashboard: a framework for mining, tracking and validating clusters in a data stream. In *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, pages 109–117. ACM, 2012.

[16] R. Hyde, P. Angelov, and A. MacKenzie. Fully online clustering of evolving data streams into arbitrarily shaped clusters. *Information Sciences*, 382:96–114, 2017.

[17] N. Kasabov. *Springer Handbook of Bio-/neuro-informatics*. Springer Science & Business Media, 2013.

[18] N. K. Kasabov. *Evolving connectionist systems: the knowledge engineering approach*. Springer Science & Business Media, 2007.

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

[20] S. Liang, E. Yilmaz, and E. Kanoulas. Dynamic clustering of streaming short documents. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 995–1004. ACM, 2016.

[21] E. Lughofer. Single-pass active learning with conflict and ignorance. *Evolving Systems*, 3(4):251–271, 2012.

[22] E. Lughofer, M. Pratama, and I. Skrjanc. Incremental rule splitting in generalized evolving fuzzy systems for autonomous drift compensation. *IEEE Transactions on Fuzzy Systems*, 2017.

[23] Z. Miller, B. Dickinson, W. Deitrick, W. Hu, and A. H. Wang. Twitter spammer detection using data stream clustering. *Information Sciences*, 260:64–73, 2014.

[24] L. O'callaghan, A. Meyerson, R. Motwani, N. Mishra, and S. Guha. Streaming-data algorithms for high-quality clustering. In *ICDE*, page 0685, 2002.

[25] 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.

[26] S. Schliebs and N. Kasabov. Evolving spiking neural networka survey. *Evolving Systems*, 4(2):87–98, 2013.

[27] J. Shao, C. Gao, W. Zeng, J. Song, and Q. Yang. Synchronization-inspired co-clustering and its application to gene expression data. In *2017 IEEE 11th International Conference on Data Mining*. IEEE, 2017.

[28] J. Shao, Z. Han, Q. Yang, and T. Zhou. Community detection based on distance dynamics. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1075–1084, 2015.

[29] J. Shao, X. He, C. Böhm, Q. Yang, and C. Plant. Synchronization-inspired partitioning and hierarchical clustering. *IEEE Transactions on Knowledge and Data Engineering*, 25(4):893–905, 2013.

[30] J. Shao, F. Huang, Q. Yang, and G. Luo. Robust prototype-based learning on data streams. *IEEE Transactions on Knowledge and Data Engineering*, 2017.

[31] J. Shao, C. Plant, Q. Yang, and C. Böhm. Detection of arbitrarily oriented synchronized clusters in high-dimensional data. In *2011 IEEE 11th International Conference on Data Mining*, pages 607–616, 2011.

[32] J. Shao, X. Wang, Q. Yang, C. Plant, and C. Böhm. Synchronization-based scalable subspace clustering of high-dimensional data. *Knowledge and Information Systems*, pages 1–29, 2016.

[33] J. Shao, Q. Yang, H.-V. Dang, B. Schmidt, and S. Kramer. Scalable clustering by iterative partitioning and point attractor representation. *ACM Transactions on Knowledge Discovery from Data*, 11(1):5, 2016.

[34] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. de Carvalho, and J. Gama. Data stream clustering: A survey. *ACM Computing Surveys (CSUR)*, 46(1):13, 2013.

[35] S. Soltic and N. Kasabov. Knowledge extraction from evolving spiking neural networks with rank order population coding. *International Journal of Neural Systems*, 20(06):437–445, 2010.

[36] G. Song, Y. Ye, H. Zhang, X. Xu, R. Y. Lau, and F. Liu. Dynamic clustering forest: an ensemble framework to efficiently classify textual data stream with concept drift. *Information Sciences*, 357:125–143, 2016.

[37] M. Spiliopoulou, E. Ntoutsi, Y. Theodoridis, and R. Schult. Monic and followups on modeling and monitoring cluster transitions. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 622–626. Springer, 2013.

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

[39] W. Ying, F.-L. Chung, and S. Wang. Scaling up synchronization-inspired partitioning clustering. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):2045–2057, 2014.

[40] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. *ACM SIGMOD Record*, 25(2):103–114, 1996.

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

[42] A. Zhou, F. Cao, W. Qian, and C. Jin. Tracking clusters in evolving data streams over sliding windows. *Knowledge and Information Systems*, 15(2):181–214, 2008.