Clustering Data Streams Based on Shared Density Between Micro-Clusters

Michael Hahsler, *Member, IEEE,* and Matthew Bolaños

Abstract—As more and more applications produce streaming data, clustering data streams has become an important technique for data and knowledge engineering. A typical approach is to summarize the data stream in real-time with an online process into a large number of so called micro-clusters. Micro-clusters represent local density estimates by aggregating the information of many data points in a defined area. On demand, a (modified) conventional clustering algorithm is used in a second offline step to recluster the micro-clusters into larger final clusters. For reclustering, the centers of the micro-clusters are used as pseudo points with the density estimates used as their weights. However, information about density in the area between micro-clusters is not preserved in the online process and reclustering is based on possibly inaccurate assumptions about the distribution of data within and between micro-clusters (e.g., uniform or Gaussian).

This paper describes DBSTREAM, the first micro-cluster-based online clustering component that explicitly captures the density between micro-clusters via a shared density graph. The density information in this graph is then exploited for reclustering based on actual density between adjacent micro-clusters. We discuss the space and time complexity of maintaining the shared density graph. Experiments on a wide range of synthetic and real data sets highlight that using shared density improves clustering quality over other popular data stream clustering methods which require the creation of a larger number of smaller micro-clusters to achieve comparable results.

Index Terms—Data mining, data stream clustering, density-based clustering.

1 Introduction

Clustering data streams [1], [2], [3], [4] has become an important technique for data and knowledge engineering. A data stream is an ordered and potentially unbounded sequence of data points. Such streams of constantly arriving data are generated for many types of applications and include GPS data from smart phones, web click-stream data, computer network monitoring data, telecommunication connection data, readings from sensor nets, stock quotes, etc.

Data stream clustering is typically done as a two-stage process with an online part which summarizes the data into many micro-clusters or grid cells and then, in an offline process, these micro-clusters (cells) are reclustered/merged into a smaller number of final clusters. Since the reclustering is an offline process and thus not time critical, it is typically not discussed in detail in papers about new data stream clustering algorithms. Most papers suggest to use an (sometimes slightly modified) existing conventional clustering algorithm (e.g., weighted k-means in CluStream [5]) where the micro-clusters are used as pseudo points. Another approach used in DenStream [6] is to use reachability where all micro-clusters which are less then a given distance from each other are linked together to form clusters. Grid-based algorithms typically merge adjacent dense grid cells to form larger clusters (see, e.g., the original version of D-Stream [7] and MR-Stream, [8]).

Current reclustering approaches completely ignore the data density in the area between the micro-clusters (grid cells) and thus might join micro-clusters (cells) which are close together but at the same time separated by a small area of low density. To address this problem, Tu and Chen [9] introduced an extension to the grid-based D-Stream algorithm [7] based on the concept of attraction between adjacent grids cells and showed its effectiveness.

In this paper, we develop and evaluate a new method to address this problem for micro-cluster-based algorithms. We introduce the concept of a shared density graph which explicitly captures the density of the original data between micro-clusters during clustering and then show how the graph can be used for reclustering micro-clusters. This is a novel approach since instead on relying on assumptions about the distribution of data points assigned to a micro-cluster (often a Gaussian distribution around a center), it estimates the density in the shared region between micro-clusters directly from the data. To the best of our knowledge, this paper is the first to propose and investigate using a shared-density-based reclustering approach for data stream clustering.

The paper is organized as follows. After a brief discussion of the background in Section 2, we present in Section 3 the shared density graph and the algorithms used to capture the density between micro-clusters in the online component. In Section 4 we describe the reclustering approach which is based on clustering or finding connected components in the shared density graph. In Section 5 we discuss the computational complexity of maintaining the shared density graph. Section 6 contains detailed experiments with synthetic and real data sets. We conclude the paper with Section 7.

- *M. Hahsler is with the Department of Engineering Management, Information, and Systems, Southern Methodist University, Dallas, TX 75226. E-mail: mhahsler@lyle.smu.edu*
- *M. Bolaños is with Research Now, Plano, TX 75024.*

Manuscript received April 19, 20xx; revised September 17, 20xx.

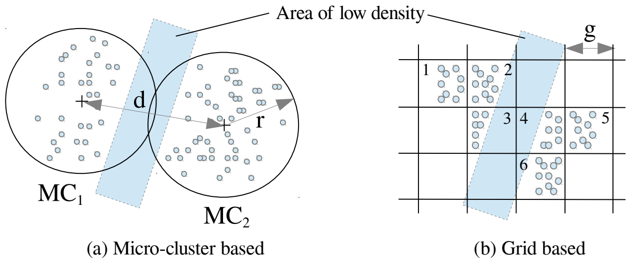
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2522412, IEEE Transactions on Knowledge and Data Engineering

2 BACKGROUND

Density-based clustering is a well-researched area and we can only give a very brief overview here. DBSCAN [10] and several of its improvements can be seen as the prototypical density-based clustering approach. DBSCAN estimates the density around each data point by counting the number of points in a user-specified eps-neighborhood and applies user-specified thresholds to identify core, border and noise points. In a second step, core points are joined into a cluster if they are density-reachable (i.e., there is a chain of core points where one falls inside the eps-neighborhood of the next). Finally, border points are assigned to clusters. Other approaches are based on kernel density estimation (e.g., DENCLUE [11]) or use shared nearest neighbors (e.g., SNN [12], CHAMELEON [13]).

However, these algorithms were not developed with data streams in mind. A data stream is an ordered and potentially unbounded sequence of data points $X = \langle \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots \rangle$. It is not possible to permanently store all the data in the stream which implies that repeated random access to the data is infeasible. Also, data streams exhibit concept drift over time where the position and/or shape of clusters changes, and new clusters may appear or existing clusters disappear. This makes the application of existing clustering algorithms difficult. Data stream clustering algorithms limit data access to a single pass over the data and adapt to concept drift. Over the last 10 years many algorithms for clustering data streams have been proposed [5], [6], [8], [9], [14], [15], [16], [17], [18], [19], [20]. Most data stream clustering algorithms use a two-stage online/offline approach [4]:

1) **Online:** Summarize the data using a set of k' micro-clusters organized in a space-efficient data structure which also enables fast lookup. Micro-clusters are representatives for sets of similar data points and are created using a single pass over the data (typically in real time when the data stream arrives). Micro-clusters are typically represented by cluster centers and additional statistics as weight (density) and dispersion (variance). Each new data point is assigned to its closest (in terms of a similarity function) micro-cluster. Some algorithms use a grid instead and non-empty grid cells represent micro-clusters (e.g., [8], [9]). If a new data point cannot be assigned to an existing micro-cluster, a new micro-cluster is created. The algorithm might also perform some housekeeping (merging or deleting micro-clusters) to keep the number of micro-clusters at a manageable size or to remove noise or information outdated due to concept drift.
2) **Offline:** When the user or the application requires a clustering, the k' micro-clusters are reclustered into k $(k \ll k')$ final clusters sometimes referred to as macro-clusters. Since the offline part is usually not regarded time critical, most researchers only state that they use a conventional clustering algorithm (typically k-means or a variation of DBSCAN [10]) by regarding the micro-cluster center positions as pseudo-points. The algorithms are often modified to take also the weight of micro-clusters into account.



(a) Micro-cluster based

(b) Grid based

Fig. 1. Problem with reclustering when dense areas are separated by small areas of low density with (a) micro clusters and (b) grid cells.

Reclustering methods based solely on micro-clusters only take closeness of the micro-clusters into account. This makes it likely that two micro-clusters which are close to each other, but separated by an area of low density still will be merged into a cluster. Information about the density between micro-clusters is not available since the information does not get recorded in the online step and the original data points are no longer available. Figure 1(a) illustrates the problem where the micro-clusters MC_1 and MC_2 will be merged as long as their distance d is low. This is even true when density-based clustering methods (e.g., DBSCAN) are used in the offline reclustering step, since the reclustering is still exclusively based on the micro-cluster centers and weights.

Several density-based approaches have been proposed for data-stream clustering. Density-based data stream clustering algorithms like D-Stream [7] and MR-Stream [8] use the idea of density estimation in grid cells in the online step. In the reclustering step these algorithms group adjacent dense grid cells into clusters. However, Tu and Chen [9] show that this leads to a problem when the data points within each cell are not uniformly distributed and two dense cells are separated by a small area of low density. Figure 1(b) illustrates this problem where the grid cells 1 through 6 are merged because 3 and 4 are adjacent ignoring the area of low density separating them. This problem can be reduced by using a finer grid, however this comes at high computational cost. MR-Stream [8] approaches this problem by dynamically creating grids at multiple resolutions using a quadtree. LeaDen-Stream [20] addresses the same problem by introducing the concept of representing a MC by multiple mini-micro leaders and uses this finer representation for reclustering.

For non-streaming clustering, CHAMELEON [13] proposes a solution to the problem by using both closeness and interconnectivity for clustering. An extension to D-Stream [9] implements this concept for data stream clustering in the form of defining attraction between grid cells as a measure of interconnectivity. Attraction information is collected during the online clustering step. For each data point, that is added to a grid cell, a hypercube of a user-specified size is created and for each adjacent grid the fraction of the hypercube's volume intersecting with that grid cell is recorded as the attraction between the point and that grid cell. The attraction between a grid cell and one of its neighbors is defined as the sum of the attractions of all its assigned points with the neighboring cell. For reclustering, adjacent dense grid cells are only merged if the attraction between the cells is high enough. Attraction measures the closeness of data points in one cell to neighboring cells and not density. It is also not directly applicable to general micro-clusters. In the following we will develop a technique to obtain density-based connectivity estimated between micro-clusters directly from the data.

3 The DBSTREAM Online Component

Typical micro-cluster-based data stream clustering algorithms retain the density within each micro-cluster (MC) as some form of weight (e.g., the number of points assigned to the MC). Some algorithms also capture the dispersion of the points by recording variance. For reclustering, however, only the distances between the MCs and their weights are used. In this setting, MCs which are closer to each other are more likely to end up in the same cluster. This is even true if a density-based algorithm like DBSCAN [10] is used for reclustering since here only the position of the MC centers and their weights are used. The density in the area between MCs is not available since it is not retained during the online stage.

The basic idea of this work is that if we can capture not only the distance between two adjacent MCs but also the connectivity using the density of the original data in the area between the MCs, then the reclustering results may be improved. In the following we develop DBSTREAM which stands for density-based stream clustering.

3.1 Leader-based Clustering

Leader-based clustering was introduced by Hardigan [21] as a conventional clustering algorithm. It is straight-forward to apply the idea to data streams (see, e.g., [20]).

DBSTREAM represents each MC by a leader (a data point defining the MC's center) and the density in an area of a user-specified radius r (threshold) around the center. This is similar to DBSCAN's concept of counting the points is an eps-neighborhood, however, here the density is not estimated for each point, but only for each MC which can easily be achieved for streaming data. A new data point is assigned to an existing MC (leader) if it is within a fixed radius of its center. The assigned point increases the density estimate of the chosen cluster and the MC's center is updated to move towards the new data point. If the data point falls in the assignment area of several MCs then all of them are updated. If a data point cannot be assigned to any existing MC, a new MC (leader) is created for the point. Finding the potential clusters for a new data point is a fixed-radius nearest-neighbor problem [22] which can be efficiently dealt with for data of moderate dimensionality using spatial indexing data structures like a k-d tree [23]. Variations of this simple algorithm were suggested in [24] for outlier detection and in [25] for sequence modeling.

The base algorithm stores for each MC a weight which is the number of data points assigned to the MC (see w_1 to w_4 in Figure 2). The density can be approximated by this weight over the size of the MC's assignment area. Note that we use for simplicity the area here, however, the approach is not restricted to two-dimensional data. For higher-dimensional data volume is substituted for area.

Definition 3.1. The **density** of MC i is estimated by $\hat{\rho}_i = \frac{w_i}{A_i}$, where w_i is the weight and A_i, the area of the circle with radius r around the center of MC i.

3.2 Competitive Learning

New leaders are chosen as points which cannot be assigned to an existing MC. The positions of these newly formed MCs are most likely not ideal for the clustering. To remedy this problem, we use a competitive learning strategy introduced in [26] to move the MC centers towards each newly assigned point. To control the magnitude of the movement, we use a neighborhood function $h()$ similar to self-organizing maps [27]. In our implementation we use the popular Gaussian neighborhood function defined between two points, $\mathbf{a}$ and $\mathbf{b}$, as

$$h(\mathbf{a}, \mathbf{b}) = \exp(-||\mathbf{a} - \mathbf{b}||^2/(2\sigma^2))$$

with $\sigma = r/3$ indicating that the used neighborhood size is ± 3 standard deviations. Since we have a continuous stream, we do not use a learning rate to reduce the neighborhood size over time. This will accommodate slow concept drift and also has the desirable effect that MCs are drawn towards areas of higher density and ultimately will overlap, a prerequisite for capturing shared density between MCs.

Note that moving centers could lead to collapsing MCs, i.e., the centers of two or more MCs converge to a single point. This behavior was already discussed in early work on self-organizing maps [28]. This will happen since the updating strategy makes sure that MCs are drawn to areas of maximal local density. Since several MCs representing the same area are unnecessary, many algorithms merge two converging MCs. However, experiments during development of our algorithm showed the following undesirable effect. New MCs are constantly created at the fringes of a dense area, then the MCs move towards the center and are merged while new MCs are again created at the fringes. This behavior is computationally expensive and degrades shared densities estimation. Therefore, we prevent collapsing MCs by restricting the movement for MCs in case they would come closer than r to each other. This makes sure that the centers do not enter the assignment radius of neighboring MCs but will end up being perfectly packed together [29] in dense areas giving us the optimal situation for estimating shared density.

3.3 Capturing Shared Density

Capturing shared density directly in the online component is a new concept introduced in this paper. The fact, that in dense areas MCs will have an overlapping assignment area, can be used to measure density between MCs by counting the points which are assigned to two or more MCs. The idea is that high density in the intersection area relative to the rest of the MCs' area means that the two MCs share an area of high density and should be part of the same macro-cluster. In the example in Figure 2 we see that MC$_2$ and MC$_3$ are close to each other and overlap. However, the shared weight $s_{2,3}$ is small compared to the weight of each of the two involved MCs indicating that the two MCs do not form a single area of high density. On the other hand, MC$_3$ and

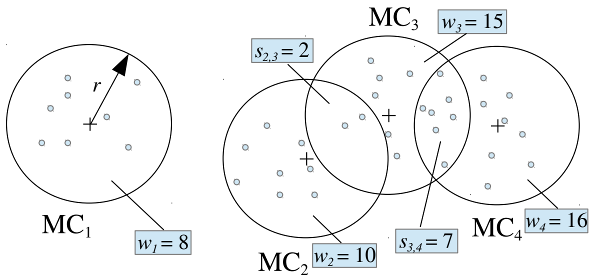


Fig. 2. MC$_1$ is a single MC. MC$_2$ and MC$_3$ are close to each other but the density between them is low relative to the two MCs densities while MC$_3$ and MC$_4$ are connected by a high density area.

MC$_4$ are more distant, but their shared weight $s_{3,4}$ is large indicating that both MCs form an area of high density and thus should form a single macro-cluster. The shared density between two MCs can be estimate by:

Definition 3.2. The **shared density** between two MCs, i and j, is estimated by $\hat{\rho}_{ij} = \frac{s_{ij}}{A_{ij}}$, where s_{ij} is the shared weight and A_{ij} is the size of the overlapping area between the MCs.

Based on shared densities we can define a shared density graph.

Definition 3.3. A **shared density graph** $G_{\mathrm{sd}} = (V, E)$ is an undirected weighted graph, where the set of vertices is the set of all MCs, i.e., $V(G_{\mathrm{sd}}) = \mathcal{MC}$, and the set of edges $E(G_{\mathrm{sd}}) = \{(v_i, v_j) \,|\, v_i, v_j \in V(G_{\mathrm{sd}}) \wedge \hat{\rho}_{ij} > 0\}$ represents all the pairs of MCs for which we have pairwise density estimates. Each edge is labeled with the pairwise density estimate $\hat{\rho}_{ij}$.

Note that most MCs will not share density with each other in a typical clustering. This leads to a very sparse shared density graph. This fact can be exploited for more efficient storage and manipulation of the graph. We represent the sparse graph by a weighted adjacency list **S**. Furthermore, during clustering we already find all fixed-radius nearest-neighbors. Therefore, obtaining shared weights does not incur any additional increase in search time.

3.4 Fading and Forgetting Data

To adapt to evolving data streams we use the exponential fading strategy introduced in DenStream [6] and used in many other algorithms. Cluster weights are faded in every time step by a factor of $2^{-\lambda}$, where $\lambda > 0$ is a user-specified fading factor. We implement fading in a similar way as in D-Stream [9], where fading is only applied when a value changes (e.g., the weight of a MC is updated). For example, if the current time-step is $t = 10$ and the weight w was last updated at $t_w = 5$ then we apply for fading the factor $2^{-\lambda(t-t_w)}$ resulting in the correct fading for five time steps. In order for this approach to work we have to keep a timestamp with the time when fading was applied last for each value that is subject to fading.

The leader-based clustering algorithm only creates new and updates existing MCs. Over time, noise will cause the creation of low-weight MCs and concept shift will make some MCs obsolete. Fading will reduce the weight of these MCs over time and the reclustering has a mechanism to exclude these MCs. However, these MCs will still be stored in memory and make finding the fixed-radius nearest neighbors during the online clustering process slower. This problem can be addressed by removing weak MCs and weak entries in the shared density graph. In the following we define weak MCs and weak shared densities.

Definition 3.4. We define MC mc_i as a **weak MC** if its weight w_i increases on average by less than one new data point in a user-specified time interval t_{gap}.

Definition 3.5. We define a **weak entry in the shared density graph** as an entry between two MCs, i and j, which on average increases its weight s_{ij} by less then α from new points in the time interval t_{gap}. α is the intersection factor related to the area of the overlap of the MCs relative to the area covered by MCs.

The rational of using α is that the overlap areas are smaller than the assignment areas of MCs and thus are likely to receive less weight. α will be discussed in detail in the reclustering phase.

Let us assume that we check every t_{gap} time step and remove weak MCs and weak entries in the shared density graph to recover memory and improve the clustering algorithm's processing speed. To ensure that we only remove weak entries, we can use the weight $w_{\mathrm{weak}} = 2^{-\lambda t_{\mathrm{gap}}}$. At any time, all entries that have a faded weight of less than w_{weak} are guaranteed to be weak. This is easy to see since any entry that gets on average an additional weight of $w' \geq 1$ during each t_{gap} interval will have a weight of at least $w' 2^{-\lambda t_{\mathrm{gap}}}$ which is greater or equal to w_{weak}. Noise entries (MCs and entries in the shared density graph) often receive only a single data point and will reach w_{weak} after t_{gap} time steps. Obsolete MCs or entries in the shared density graph stop to receive data points and thus their weight will be faded till it falls below w_{weak} and then they are removed. It is easy to show that for an entry with a weight w it will take $t = \log_2(w)/\lambda + t_{\mathrm{gap}}$ time steps to reach w_{weak}. For example, at $\lambda = 0.01$ and $t_{\mathrm{gap}} = 1000$ it will take 1333 time steps for an obsolete MC with a weight of $w = 10$ to fall below w_{weak}. The same logic applies to shared density entries using αw_{weak}. Note that the definition of weak entries and w_{weak} is only used for memory management purpose. Reclustering uses the definition of strong entries (see Section 4). Therefore, the quality of the final clustering is not affected by the choice of t_{gap} as long as it is not set to a time interval which is too short for actual MCs and entries in the shared density graph to receive at least one data point. This clearly depends on the expected number of MCs and therefore depends on the chosen clustering radius r and the structure of the data stream to be clustered. A low multiple of the number of expected MCs is typically sufficient. The parameter t_{gap} can also be dynamically adapted during running the clustering algorithm. For example t_{gap} can be reduced to mark more entries as weak and remove them more often if memory or processing speed gets low. On the other hand, t_{gap} can be increased during clustering if not enough structure of the data stream is retained.

3.5 The Complete Online Algorithm

Algorithm 1 shows our approach and the used clustering data structures and user-specified parameters in detail. Micro-clusters are stored as a set $\mathcal{MC}$. Each micro-cluster is represented by the tuple $(\mathbf{c}, w, t)$ representing the cluster center, the cluster weight and the last time it was updated, respectively. The weighted adjacency list $\mathbf{S}$ represents the sparse shared density graph which captures the weight of the data points shared by MCs. Since shared density estimates are also subject to fading, we also store a timestamp with each entry. Fading also shared density estimates is important since MCs are allowed to move which over time would lead to estimates of intersection areas the MC is not covering anymore.

The user-specified parameters r (the radius around the center of a MC within which data points will be assigned to the cluster) and λ (the fading rate) are part of the base algorithm. α, t_{gap} and $w_{\min}$ are parameters for reclustering and memory management and will be discussed later.

Updating the clustering by adding a new data point $\mathbf{x}$ to the clustering is defined by Algorithm 1. First, we find all MCs for which $\mathbf{x}$ falls within their radius. This is the same as asking which MCs are within r from $\mathbf{x}$, which is the fixed-radius nearest neighbor problem which can be efficiently solved for data of low to moderate dimensionality [22]. If no neighbor is found then a new MC with a weight of 1 is created for $\mathbf{x}$ (line 4 in Algorithm 1). If one or more neighbors are found then we update the MCs by applying the appropriate fading, increasing their weight and then we try to move them closer to $\mathbf{x}$ using the Gaussian neighborhood function $h()$ (lines 7–9).

Next, we update the shared density graph (lines 10–13). To prevent collapsing MCs, we restrict the movement for MCs in case they come closer than r to each other (lines 15–19). Finally, we update the time step.

The cleanup process is shown in Algorithm 2. It is executed every t_{gap} time steps and removes weak MCs and weak entries in the shared density graph to recover memory and improve the clustering algorithm's processing speed.

Algorithm 1 Update DBSTREAM clustering.

```
Require: Clustering data structures initially empty or 0
  $\mathcal{MC}$                                              ▷ set of MCs
  $mc \in \mathcal{MC}$ has elements $mc = (\mathbf{c}, w, t)$   ▷ center, weight,
  last update time
  $\mathbf{S}$        ▷ weighted adjacency list for shared density graph
  $s_{ij} \in \mathbf{S}$ has an additional field $t$        ▷ time of last update
  $t$                                                ▷ current time step

Require: User-specified parameters
  $r$                                             ▷ clustering threshold
  $\lambda$                                                  ▷ fading factor
  $t_{\text{gap}}$                                              ▷ cleanup interval
  $w_{\min}$                                             ▷ minimum weight
  $\alpha$                                             ▷ intersection factor

 1: function UPDATE($\mathbf{x}$)                        ▷ new data point $\mathbf{x}$
 2:     $\mathcal{N} \leftarrow$ findFixedRadiusNN($\mathbf{x}, \mathcal{MC}, r$)
 3:     if $|\mathcal{N}| < 1$ then                            ▷ create new MC
 4:         add ($\mathbf{c} = \mathbf{x}, t = t, w = 1$) to $\mathcal{MC}$
 5:     else                                        ▷ update existing MCs
 6:         for each $i \in \mathcal{N}$ do
 7:             $mc_i[w] \leftarrow mc_i[w]\ 2^{-\lambda(t - mc_i[t])} + 1$
 8:             $mc_i[\mathbf{c}] \leftarrow mc_i[\mathbf{c}] + h(\mathbf{x}, mc_i[\mathbf{c}])(\mathbf{x} - mc_i[\mathbf{c}])$
 9:             $mc_i[t] \leftarrow t$
                                              ▷ update shared density
10:             for each $j \in \mathcal{N}$ where $j > i$ do
11:                 $s_{ij} \leftarrow s_{ij}\ 2^{-\lambda(t - s_{ij}[t])} + 1$
12:                 $s_{ij}[t] \leftarrow t$
13:             end for
14:         end for
                                          ▷ prevent collapsing clusters
15:         for each $(i, j) \in \mathcal{N} \times \mathcal{N}$ and $j > i$ do
16:             if dist($mc_i[\mathbf{c}], mc_j[\mathbf{c}]$) $< r$ then
17:                 revert $mc_i[\mathbf{c}]$, $mc_j[\mathbf{c}]$ to previous positions
18:             end if
19:         end for
20:     end if
21:     $t \leftarrow t + 1$
22: end function
```

Algorithm 2 Cleanup process to remove inactive micro-clusters and shared density entries from memory.

```
Require: $\lambda$, $\alpha$, $t_{\text{gap}}$, $t$, $\mathcal{MC}$ and $\mathbf{S}$ from the clustering.
function CLEANUP( )
    $w_{\text{weak}} = 2^{-\lambda t_{\text{gap}}}$
    for each $mc \in \mathcal{MC}$ do
        if $mc[w]\ 2^{-\lambda(t - mc[t])} < w_{\text{weak}}$ then
            remove weak $mc$ from $\mathcal{MC}$
        end if
    end for
    for each $s_{ij} \in \mathbf{S}$ do
        if $s_{ij}\ 2^{-\lambda(t - s_{ij}[t])} < \alpha w_{\text{weak}}$ then
            remove weak shared density $s_{ij}$ from $\mathbf{S}$
        end if
    end for
end function
```

4 Shared Density-Based Reclustering

Reclustering represents the algorithm's offline component which uses the data captured by the online component. For simplicity we discuss two-dimensional data first and later discuss implications for higher-dimensional data. For reclustering, we want to join MCs which are connected by areas of high density. This will allow us to form macro-clusters of arbitrary shape, similar to hierarchical clustering with single-linkage or DBSCAN's reachability, while avoiding joining MCs which are close to each other but are separated by an area of low density.

4.1 Micro-Cluster Connectivity

In two dimensions, the assignment area of a MC is given by $A = \pi r^2$. It is easy to show that the intersecting area between two circles with equal radius r and the centers exactly r apart from each other is $A^* = \frac{4\pi - 3\sqrt{3}}{6}\ r^2$. By normalizing the area of the circle to $A = 1$ (i.e., setting the radius to $r = \sqrt{1/\pi}$) we get an intersection area $A^* = 0.391$

or 39.1% of the circle's area. Since we expect adjacent MCs i and j which form a single macro-cluster in a dense area to eventually be packed together till the center of one MC is very close to the r boundary of the other, 39.1% is the upper bound of the intersecting area.

Less dense clusters will also have a lower shared density. To detect clusters of different density correctly, we need to define connectivity relative to the densities (weights) of the participating clusters. That is, for two MCs, i and j, which are next to each other in the same macro-cluster we expect that $\hat{\rho}_{ij} \approx (\hat{\rho}_i + \hat{\rho}_j)/2$, i.e., the density between the MCs is similar to the average density inside the MCs. To formalize this idea we introduce the connectivity graph.

Definition 4.1. The **connectivity graph** $G_c = (V, E)$ is an undirected weighted graph with the micro clusters as vertices, i.e., $V(G_c) = \mathcal{MC}$. The set of edges is defined by $E(G_c) = \{(v_i, v_j) \mid v_i, v_j \in V(G_c) \wedge c_{ij} > 0\}$, with $c_{ij} = \frac{s_{ij}}{(w_i + w_j)/2}$. s_{ij} is the weight in the intersecting area of MCs i and j and w_i and w_j are the MCs' weights. The edges are labeled with weights given by c_{ij}.

Note that the connectivity is not calculated as $\frac{\hat{\rho}_{ij}}{(\hat{\rho}_i + \hat{\rho}_j)/2}$ and thus has to be corrected for the difference in the size of the area of the MCs and the intersecting area. This can be easily done by introducing an intersection factor $\alpha_{ij} = A_{ij}/A_i$ which results in $\frac{\hat{\rho}_{ij}}{(\hat{\rho}_i + \hat{\rho}_j)/2} = \alpha_{ij} c_{ij}$. α_{ij} depends on the distance between the participating MCs i and j. Similar to the non-streaming algorithm CHAMELEON [13], we want to combine MCs which are close together and have high interconnectivity. This objective can be achieved by simply choosing a single global intersection factor α. This leads to the concept of α-connectedness.

Definition 4.2. Two MCs, i and j, are **α-connected** iff $c_{ij} \geq \alpha$, where α is the user-defined intersection factor.

For two-dimensional data the intersection factor α has a theoretical maximum of 0.391 for an area of uniform density when the two MCs are optimally packed (the centers are exactly r apart). However, in dynamic clustering situations MCs may not be perfectly packed all the time and minor variations in the observed density in the data are expected. Therefore, a smaller value than the theoretically obtained maximum of 0.391 will be used in practice. It is important to notice that a threshold on α is a single decision criterion which combines the fact that two MCs are very close to each other and that the density between them is sufficiently high. Two MCs have to be close together or the intersecting area and thus the expected weight in the intersection will be small and the density between the MCs has to be high relative to the density of the two MCs. This makes using the concept of α-connectedness very convenient.

For 2-dimensional data we suggest $\alpha = .3$ which is a less stringent cut-off than the theoretical maximum. Doing this will also connect MCs, even if they have not (yet) moved into a perfect packing arrangement. Note also that the definitions of α-connectedness uses the connectivity graph which depends on the density of the participating MCs and thus it can automatically handle clusters of vastly different density within a single clustering.

Algorithm 3 Reclustering using shared density graph.

```
Require: $\lambda$, $\alpha$, $w_{\min}$, $t$, $\mathcal{MC}$ and $\mathbf{S}$ from the clustering.
function RECLUSTER( )
    weighted adjacency list $\mathbf{C} \leftarrow \emptyset$ ▷ connectivity graph

    for each $s_{ij} \in \mathbf{S}$ do    ▷ construct connectivity graph
        if $\mathcal{MC}_i[w] \geq w_{\min} \wedge \mathcal{MC}_j[w] \geq w_{\min}$ then
            $c_{ij} \leftarrow \frac{s_{ij}}{(\mathcal{MC}_i[w]+\mathcal{MC}_j[w])/2}$
        end if
    end for
    return findConnectedComponents($\mathbf{C} \geq \alpha$)
end function
```

4.2 Noise Clusters

To remove noisy MCs from the final clustering, we have to detect these MCs. Noisy clusters are typically characterized as having low density represented by a small weight. Since the weight is related to the number of points covered by the MC, we use a user-set minimum weight threshold to identify noisy MCs. This is related to minPoints in DBSCAN or C_m used by D-Stream.

Definition 4.3. The set of **noisy MCs**, $\mathcal{MC}_{\text{noisy}}$, is the subset of $\mathcal{MC}$ containing the MCs with less than a user-specified minimum weight $w_{\min}$. That is, $\mathcal{MC}_{\text{noisy}} = \{\text{MC}_i \mid \text{MC}_i \in \mathcal{MC} \wedge w_i < w_{\min}\}$.

Given the definition of noisy and weak clusters, we can define strong MCs which should be used in the clustering.

Definition 4.4. A **strong MC** is a MC that is not noisy or weak, i.e., $\mathcal{MC}_{\text{strong}} = \mathcal{MC} \setminus (\mathcal{MC}_{\text{noisy}} \cup \mathcal{MC}_{\text{weak}})$.

Note that t_{gap} is typically chosen such that $\mathcal{MC}_{\text{weak}} \subseteq \mathcal{MC}_{\text{noisy}}$ and therefore the exact choice of t_{gap} has no influence on the resulting clustering, only influencing runtime performance and memory requirements.

4.3 Higher-dimensional Data

In dimensions higher than two the intersection area becomes an intersection volume. To obtain the upper limit for the intersection factor α we use a simulation to estimate the maximal fraction of the shared volume of MCs (hyperspheres) that intersect in $d = 1, 2, \ldots, 10, 20$ and 50-dimensional space. The results are shown in Table 1. With increasing dimensionality the volume of each hypersphere increases much more than the volume of the intersection. This leads at higher dimensions to a situation where it becomes very unlikely that we observe many data points in the intersection. This is consistent with the problem know as the curse of dimensionality which effects distance-based clustering as well as Euclidean density estimation. This also effects other density based algorithms (e.g., D-Stream's attraction in [9]) in the same way. For high-dimensional data we plan to extend a subspace clustering approach like HPStream [15] to maintain a shared density graph in lower-dimensional subspaces.

4.4 The Offline Algorithm

Algorithm 3 shows the reclustering process. The parameters are the intersection factor α and the noise threshold $w_{\min}$.

TABLE 1
Maximal size of the intersection for two MCs (hyperspheres) and maximal number of neighboring MCs in d dimensions.

Dimensions d	1	2	3	4	5	6	7
Intersection area	50%	39%	31%	25%	21%	17%	14%
Neighbors $\lvert\mathcal{K}_d\rvert$	2	6	12	24	40	72	126

Dimensions d	8	9	10	20	50
Intersection area	12%	10%	8%	1.5%	.02%
Neighbors $\lvert\mathcal{K}_d\rvert$	240	272	$\geq$ 336	$\geq$ 17,400	$\geq$ 50,000,000

The connectivity graph $\mathbf{C}$ is constructed using only shared density entries between strong MCs. Finally, the edges in the connectivity graph with a connectivity value greater than the intersection threshold are used to find connected components representing the final clusters.

4.5 Relationship to Other Algorithms

DBSTREAM is closely related to DBSCAN [10] with two important differences. Similar to DenStream [6], density estimates are calculated for micro-clusters rather than the epsilon neighborhood around each point. This reduces computational complexity significantly. The second change is that DBSCAN's concept of reachability is replaced by α-connectivity. Reachability only reflects closeness of data points, while α-connectivity also incorporates interconnectivity introduced by CHAMELEON [13].

In general, the connectivity graph developed in this paper can be seen as a special case of a shared nearest neighbor graph where the neighbors shared by two MCs are given by the points in the shared area. As such it does not represent k shared nearest neighbors but the set of neighbors given by a fixed radius. DBSTREAM uses the most simple approach to partition the connectivity graph by using α as a global threshold and then finding connected components. However, any graph partitioning scheme, e.g., the ones used for CHAMELEON or spectral clustering, can be used to detect clusters.

Compared to D-Stream's concept of attraction which is used between grid cells, DBSTREAM's concept of α-connectivity is also applicable to micro-clusters. DBSTREAM's update strategy for micro cluster centers based on ideas from competitive learning allows the centers to move towards areas of maximal local density, while D-Stream's grid is fixed. This makes DBSTREAM more flexible which will be illustrated in the experiments by the fact that DBSTREAM typically needs fewer MCs to model the same data stream.

5 Computational Complexity

Space complexity of the clustering depends on the number of MCs that need to be stored in $\mathcal{MC}$. In the worse case, the maximum number of strong MCs at any time is t_{gap} MCs and is reached when every MC receives exactly a weight of one during each interval of t_{gap} time steps. Given the cleanup strategy in Algorithm 2, where we remove weak MCs every t_{gap} time steps, the algorithm never stores more than $k' = 2t_{\text{gap}}$ MCs.

The space complexity of $\mathcal{MC}$ is linear in the maximal number of MCs k'. The worst case size of the adjacency list of the shared density graph $\mathbf{S}$ depends on k' and the dimensionality of the data. In the 2D case each MC can have a maximum of $|\mathcal{N}| = 6$ neighbors (at optimal packing). Therefore, each of the k' MCs has in the adjacency list $\mathbf{S}$ at most six entries resulting in a space complexity of storing $\mathcal{MC}$ and $\mathbf{S}$ of $O(t_{\text{gap}})$.

For higher-dimensional data streams, the maximal number of possible adjacent hyperspheres is given by Newton's number also referred to as kissing number [29]. Newton's number defines the maximal number of hyperspheres which can touch a hypersphere of the same size without intersecting any other hypersphere. If we double the radius of all hyperspheres in this configuration then we get our scenario with sphere centers touching the surface of the center sphere. We use $\mathcal{K}_d$ do denote Newton's number in d dimensions. Newton's exact number is known only for some small dimensionality values d, and for many other dimensions only lower and upper bounds are known (see Table 1 for $d = 1$ to 50). Note, that Newton's number grows fast, reaches 196,560 for $d = 24$ and is unknown for most larger d. This growth would make storing the shared weights for high-dimensional data in a densely packed area very expensive. However, we also know that the maximal neighborhood size $|\mathcal{N}_{\max}| \leq \min(k' - 1, \mathcal{K}_d)$, since we cannot have more neighbors than we have MCs. Therefore, the space complexity of maintaining $\mathbf{S}$ is bounded by $O(k'|\mathcal{N}_{\max}|)$.

To analyze the algorithm's time complexity, we need to consider all parts of the clustering function. The fixed-radius nearest neighbor search can be done using linear search in $O(dnk')$, where d is the data dimensionality, n is the number of data points clustered and k' is the number of MCs. The time complexity can be improved to $O(d\,n\log(k'))$ using a spacial indexing data structure like a k-d tree [23]. Adding or updating a single MC is done in time linear in n. For the shared density graph we need to update in the worst case O($n\,|\mathcal{N}_{\max}|^2$) elements in $\mathbf{S}$. The overall time complexity of clustering is thus $O(d\,n\log(k') + n\,|\mathcal{N}_{\max}|^2)$. The cleanup function's time complexity depends on the size of $\mathcal{MC}$ which is linear in the number of MCs and $\mathbf{S}$ which depends on the maximal neighborhood size. Also, the cleanup function is only applied every t_{gap} points. This gives $O(\frac{n(k'+k'|\mathcal{N}_{\max}|)}{t_{\text{gap}}})$ for cleanup.

Reclustering consists of finding the minimum weight which involves sorting with a time complexity of $O(k'\log(k'))$, building the adjacency list for the connectivity graph in $O(k'|\mathcal{N}_{\max}|)$ operations, and then finding the connected components which takes time linear in the sum of the number of vertices and edges $O(k' + k'|\mathcal{N}_{\max}|)$. The whole reclustering process takes $O(k'\log(k') + 2k'|\mathcal{N}_{\max}| + k')$ operations.

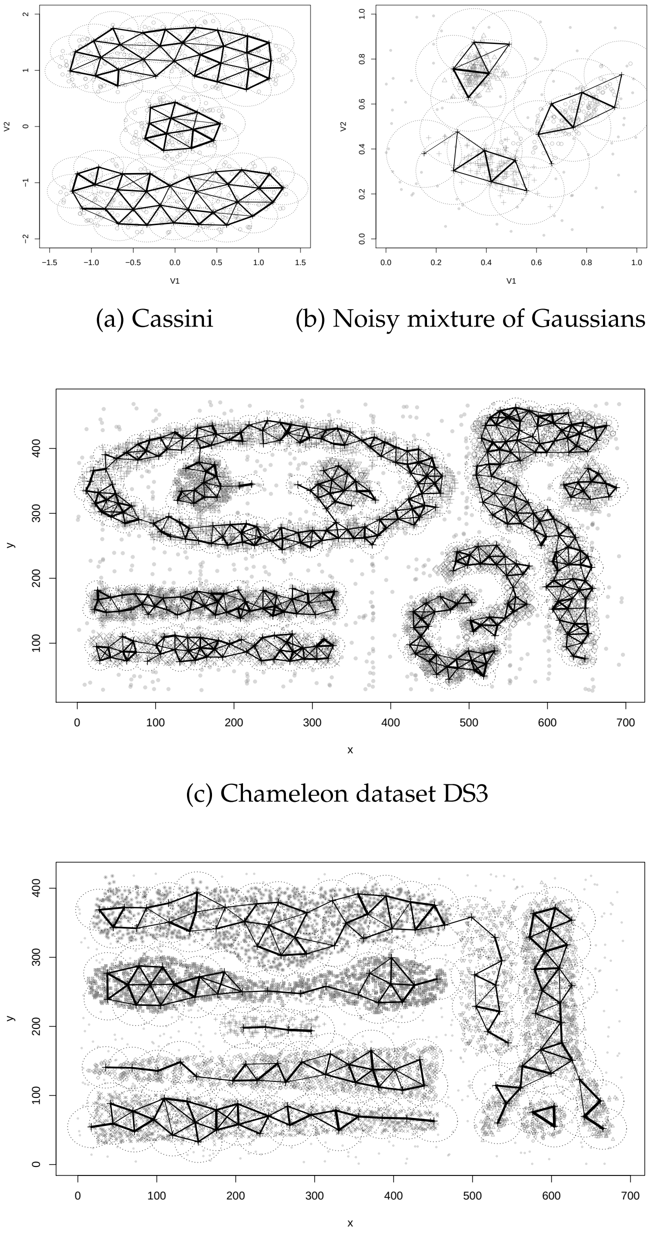
For low-dimensional data, d and $|\mathcal{N}_{\max}| = \mathcal{K}_d$ are small constants. For high-dimensional data, the worst case neighborhood size is $|\mathcal{N}_{\max}| = k' = 2t_{\text{gap}}$. The space and time complexity for low and high-dimensional data is compared in Table 2. Although, for high-dimensional data space and time complexity is in the worst case proportional to the square of the maximal number of MCs (which is controlled

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2522412, IEEE Transactions on Knowledge and Data Engineering

TABLE 2
Clustering complexity for low and high-dimensional data.

	low dimensionality	high dimensionality
space complexity	$O(t_{\text{gap}})$	$O(t^2_{\text{gap}})$
time complexity		
clustering	$O(n\log(2t_{\text{gap}}))$	$O(nt^2_{\text{gap}})$
cleanup	$O(n)$	$O(nt_{\text{gap}})$
reclustering	$O(t_{\text{gap}}\log(t_{\text{gap}}))$	$O(t^2_{\text{gap}})$

by t_{gap}), the experiments below show that in practice the number of edges per MC in the shared density graph stays even for higher-dimensional data at a very manageable size. In a simulation of a mixture of three Gaussians in 50 dimensions and a $t_{\text{gap}} = 1000$, the actual average number of entries per MC in $\mathbf{S}$ is below 20 compared to the theoretical maximum of $2t_{\text{gap}} = 2000$. Note that at this dimensionality $\mathcal{K}_{50}$ would already be more than 50,000,000. This results in very good performance in practice. The following experiments also show that shared density reclustering performs very well with a significantly smaller number of MCs than other approaches and thus all three operations can typically be performed online.



(a) Cassini (b) Noisy mixture of Gaussians

(c) Chameleon dataset DS3

(d) Chameleon dataset DS4

Fig. 3. Four example data sets clustered with DBSTREAM. MCs and their assignment area are shown as circles. The shared density graph is shown as solid lines connecting MCs.

6 Experiments

To perform our experiments and make them reproducible, we have implemented/interfaced all algorithms in a publicly available R-extension called **stream** [30]. **stream** provides an intuitive interface for experimenting with data streams and data stream algorithms. It includes generators for all the synthetic data used in this paper as well as a growing number of data stream mining algorithms including clustering algorithms available in the MOA (Massive Online Analysis) framework [31] and the algorithm discussed in this paper.

In this paper we use four synthetic data streams called Cassini, Noisy Mixture of Gaussians, and DS3 and DS4[1] used to evaluate CHAMELEON [13]. These data sets do not exhibit concept drift. For data with concept drift we use MOA's Random RBF Generator with Events. In addition we use several real data sets called Sensor[2], Forest Cover Type[3] and the KDD CUP'99 data[4] which are often used for comparing data stream clustering algorithms.

Kremer et al. [32] discuss internal and external evaluation measures for the quality of data stream clustering. We conducted experiments with a large set of evaluation measures (purity, precision, recall, F-measure, sum of squared distances, silhouette coefficient, mutual information, adjusted Rand index). In this study we mainly report the adjusted Rand index to evaluate the average agreement of the known cluster structure (ground truth) of the data stream with the found structure. The adjusted Rand index (adjusted for expected random agreements) is widely accepted as the appropriate measure to compare the quality of different partitions given the ground truth [33]. Zero indicates that the found agreements can be entirely explained by chance and the closer the index is to one, the better the agreement. For clustering with concept drift, we also report average purity and average within cluster sum of squares. However, like most other measures, these make comparison difficult. For example, average purity (equivalent to precision and part of the F-measure) depends on the number of clusters and thus makes comparison of clusterings with a different number of clusters invalid. The Within cluster Sum of Squares (WSS) favors algorithms which produce spherical clusters (e.g., k-means-type algorithms). A smaller WSS represent tighter clusters and thus a better clustering. However, WSS always will get smaller with an increasing number of clusters. We report these measures here for comparison since they are

[1] http://glaros.dtc.umn.edu/gkhome/cluto/cluto/download
[2] http://www.cse.fau.edu/~xqzhu/stream.html
[3] http://archive.ics.uci.edu/ml/datasets/Covertype
[4] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

used in many data stream clustering papers.

6.1 Clustering Static Structures

A data stream clustering algorithm needs to be able to adapt to concept drift, however, if a clustering algorithm is not able to cluster static structures, then its ability to adapt does not matter. Therefore, we use in our first set of experiments data with fixed known patterns. Four data streams are chosen because they are representatives for difficult static clustering problems. Example points for all four data streams are shown in grey in Figure 3. Cassini is a well know artificial dataset with three clusters of uniform density, where two clusters are concave and close to the center cluster. The Noisy Mixture of Gaussians stream contain three clusters, where the centers of the Gaussians and the covariance matrices are randomly chosen for each new stream. The Gaussians often overlap and 50% uniform noise is added. Finally, we consider two datasets introduced for the CHAMELEON clustering algorithm (DS3 and DS4). These datasets contain several clustering challenges including nested clusters of non-convex shape and non-uniform noise.

Figure 3 shows example clustering results for each data stream. Data points are shown in grey and the micro-cluster centers are shown in black with a dotted circle representing each MC's assignment area. The edges in the shared density graph are represented by black lines connecting MCs. We see that the shared density graph picks up the general structure of the clusters very well. In Figures 3 (c) and (d) we see several errors where the shared density graph connects neighboring clusters. These connections are typically only single edges and could be avoided using a more sophisticated, but also slower graph partitioning approach.

To investigate the effectiveness of using a shared density graph introduced for reclustering in DBSTREAM, we perform the following simulation study with 10,000 data points per stream. We use such short streams since the structures are static and only minor changes occur after an initial learning phase where the algorithms place MCs.

We create a data stream and then use DBSTREAM with shared density graph using different clustering radii. To investigate the contribution of the shared density reclustering, we also report the results for the leader-based algorithm used in DBSTREAM, but with pure reachability reclustering. For comparison we also use the original D-Stream with reachability, D-Stream with attraction, DenStream and CluStream. All algorithms were tuned such that they produce a comparable number of MCs (within ±10% of the DBSTREAM results). To find the best parameters, we use grid search over a set of reasonable parameters for each algorithm. We searched for DBSTREAM the best combination of $w_{\min} = \{1, 2, 3\}$ and $\alpha = \{0.1, 0.2, 0.3\}$. For D-Stream we searched a gridsize of the same range as DBSTREAM's r (in 0.01 increments) and $C_m = \{1, 2, 3\}$. For DenStream we searched an ϵ in the same range as DBSTREAM's r, $\mu = \{1, 2, \ldots, 20\}$ and $\beta = \{0.2, 0.4\}$. For CluStream we set the number of micro-clusters to the number produced by DBSTREAM. We repeat this procedure with 10 random samples and then evaluate average cluster quality.

TABLE 3
Adjusted Rand index for the Cassini data.

r	0.8	0.6	0.4	0.2	0.1
# of MCs	14	22	44	144	482
DBSTREAM	**0.89**	**0.99**	**0.97**	**1.00**	**0.99**
without shared density	0.35	0.42	0.50	**1.00**	**0.99**
D-Stream	-0.02	-0.02	0.34	0.87	0.55
D-Stream + attraction	0.62	0.83	0.71	0.85	0.51
DenStream	-	0.10	0.58	0.48	-
CluStream	0.52	0.54	0.50	0.34	0.07

TABLE 4
Adjusted Rand index for Noisy mixture of 3 random Gaussians with 50% noise.

r	0.2	0.1	0.05	0.03	0.01
# of MCs	5	13	35	71	308
DBSTREAM	**0.60**	**0.78**	**0.74**	0.63	0.10
without shared density	0.49	0.63	0.70	**0.68**	0.31
D-Stream	0.27	0.32	0.32	0.47	0.22
D-Stream + attraction	0.42	0.42	0.47	0.47	0.20
DenStream	-	0.61	0.64	-	-
CluStream	0.47	0.57	0.52	0.61	**0.61**

TABLE 5
Adjusted Rand index for Chameleon dataset DS3.

r	35	30	25	20	15
# of MCs	160	155	198	296	406
DBSTREAM	**0.81**	**0.74**	**0.78**	**0.76**	**0.88**
without shared density	0.0004	0.22	0.21	0.49	0.72
D-Stream	0.29	0.37	0.61	0.46	0.37
D-Stream + attraction	0.29	0.37	0.61	0.46	0.37
DenStream	0.22	0.22	0.01	-	-
CluStream	0.34	0.27	0.26	0.18	0.08

TABLE 6
Adjusted Rand index for the Chameleon dataset DS4.

r	0.7	0.5	0.3	0.2	0.1
# of MCs	13	24	58	120	434
DBSTREAM	0.72	**0.97**	**1.00**	**1.00**	0.84
without shared density	0.70	0.92	0.94	0.98	**0.99**
D-Stream	0.00	0.43	0.94	0.96	0.84
D-Stream + attraction	**0.96**	0.74	0.88	0.99	0.84
DenStream	0.51	0.44	0.58	-	-
CluStream	0.64	0.61	0.59	0.64	0.62

The results are presented in Tables 3 to 6. The best results are set in bold. If the results of two algorithms were very close then we used the Student's t-test to determine if the difference is significant (the scores of the 10 runs are approximately normally distributed). If the difference between the top algorithm is not significant at a 1% significance level, then we set both in bold.

For the Cassini data, DBSTREAM finds often a perfect clustering at only 14 MCs while D-Stream needs many more MCs to produce comparable results. CluStream (with k-means) reclustering does not perform well since the clusters are not strictly convex. The mixture of three Gaussians is a hard problem since the randomly placed clusters often overlap significantly and 50% of the data points are uniformly distributed noise. DBSTREAM performs similarly to CluStream and DenStream while D-Stream performs poorly. Finally, on DS3 and DS4, DBSTREAM performs in most

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2522412, IEEE Transactions on Knowledge and Data Engineering

cases superior to all competitors.

The experiments show that DBSTREAM consistently performs equally well or better than other reclustering strategies with fewer and therefore larger MCs. A reason is that larger MCs mean that the intersection areas between MCs are also larger and potentially can contain more points which improves the quality of the estimates of the shared density. It is interesting that clustering quality (in terms of the adjusted Rand index) of DBSTREAM with a very small number of MCs is comparable to the clustering quality achieved by other reclustering methods with many more MCs. This is an important finding since less and larger MCs means faster execution and the lower memory requirements for the MCs can be used to offset the addition memory needed to maintain the shared density graph.

6.2 Clustering with Concept Drift

Next, we investigate clustering performance over several data stream. For evaluation, we use the horizon-based prequential approach introduced in the literature [6] for clustering evolving data streams. Here the current clustering model is evaluated with the next 1000 points in the horizon and then these points are used to update the model. Recent detailed analysis of prequential error estimation for classification can be found in [34], [35]. We compare DBSTREAM again to D-Stream, DenStream and CluStream. Note that the number of clusters varies over time for some of the datasets. This needs to be considered when comparing to CluStream, which uses a fixed number of clusters and thus is at a disadvantage in this situation.

Figure 4 shows the results over the first 10,000 points from a stream from the Cassini data set. DBSTREAM's shared density approach learns the structure quickly while CluStream's k-means reclustering cannot cluster the concave structure of the data properly. DenStream often tends to place single or few MCs in its own cluster, resulting in spikes of very low quality. D-Stream is slower in adapting to the structure and produces results inferior to DBSTREAM.

Figures 5 show the results on a stream created with MOA's Random Radial Base Function (RBF) Generator with Events. The events are cluster splitting/merging and deletion/creation. We use the default settings with 10% noise, start with 5 clusters and allow one event every 10,000 data points. We use for DenStream the ϵ parameter as suggested in the original paper. Since the number of clusters changes over time, and CluStream needs a fixed number, we set k to 5, the initial number of clusters, accepting the fact that sometimes this will be incorrect. CluStream does not perform well because of this fixed number of macro-clusters and the noise in the data while DenStream, D-Stream and DBSTREAM perform better.

Next, we use a data stream consisting of 2 million readings from the 54 sensors deployed in the Intel Berkeley Research lab measuring humidity, temperature, light and voltage over a period of more than one month. The results in Figure 6 show that all clustering algorithms detect daily fluctuations, and DBSTREAM produces the best results.

Finally, we use the Forest Cover Type data, which contains 581,012 instances of cartographic variables (we use the 10 numeric variables). The ground truth groups the instances into 7 different forest cover types. Although this data is not a data stream, we use it here in a streaming fashion. Figure 7 shows the results. The data set is hard to cluster with many clusters in the ground truth heavily overlapping with each other. For some part of the data the adjusted Rand index for all algorithms even becomes negative, indication that structure found in the cartographic variables does not correspond with the ground truth. DBSTREAM is again the top performer with on average a higher average adjusted Rand index than DenStream and CluStream.

In general, DBSTREAM performs very well in terms of average corrected Rand index, high average purity and a small within cluster sum of squares (WSS). Only in one instance D-Stream produces a better purity result. CluStream produces twice a slightly lower WSS. This can be explained by the fact that CluStream uses k-means reclustering which directly tries to minimize WSS.

6.3 Scalability Results

A major concern with scalability is that Newton's number increases dramatically with the dimensionality d and, therefore, even for a moderately high dimensionality (e.g., $\mathcal{K}_{17} > 5000$) many operations will take close to $O(k'^2)$ instead of $O(k')$ time. In order to analyze scalability, we use mixture of Gaussians data similar to the data set used above and the well known KDD Cup'99 data set.

We create mixture of Gaussians data sets with three clusters in d-dimensional space, where d is ranging from 2 to 50. Since we are interested in the average number of edges of the shared density graph and noise would introduce many MCs without any edge, we add no noise to the data for the following experiment. We always use 10,000 data points for clustering, repeat the experiment for each value of d ten times and report the average. To make the results better comparable, we tune the clustering algorithm by choosing r to produce about 100–150 MCs. Therefore we expect the maximum for the average edges per MC in the shared density to be between 100 and 150 for high-dimensional data. Figure 9 shows that the average number of edges in the shared density graph grows with the dimensionality of the data. However, it is interesting to note that the number is significantly less than expected given the worst case number obtained via Newton's number or k'. After a dimensionality of 25 the increase in the number of edges starts to flatten out at a very low level. This can be explained by the fact that only the MCs representing a cluster in the data are packed together and the MCs on the surface of each cluster have significantly less neighbors (only towards the inside of the cluster). Therefore, clusters with larger surface area reduce the average number of edges in the shared density graph. This effect becomes more pronounces in higher dimensions since the surface area increases exponentially with d and this offsets the exponential increase in possible neighbors.

Next, we look at the cost of maintaining the shared density graph for a larger evolving data stream. The KDD Cup'99 data set contains network traffic data with more than 4 million records and we use the 34 numerical features for clustering. First, we standardize the values by subtracting the feature mean and dividing by the feature standard deviation. We use a radius of $r = 1$ for clustering. Since we are interested in the memory cost, we set α and w_{noise}

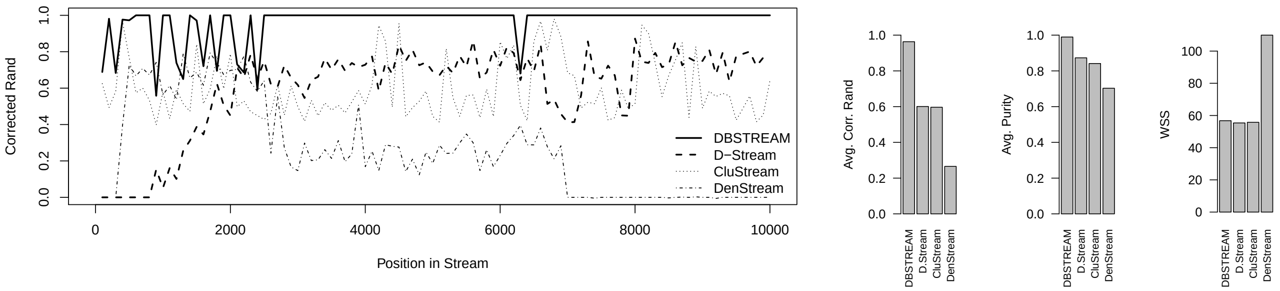


Fig. 4. Learning the structure of the Cassini data set over the first 10,000 data points.

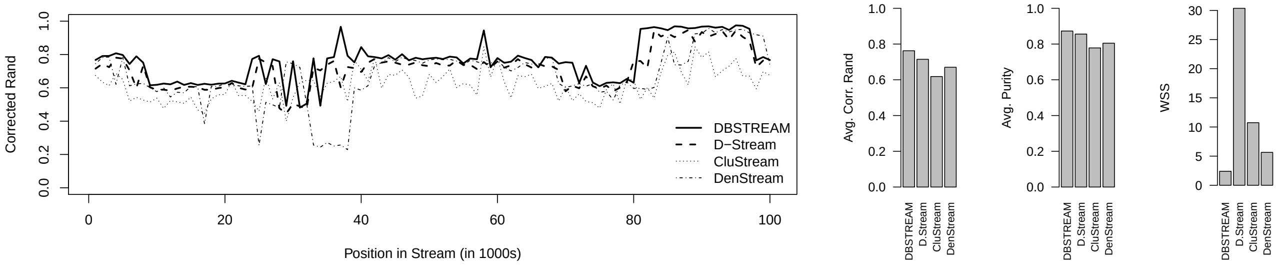


Fig. 5. Clustering quality on MOA's Random RBF Generator (100,000 data points).

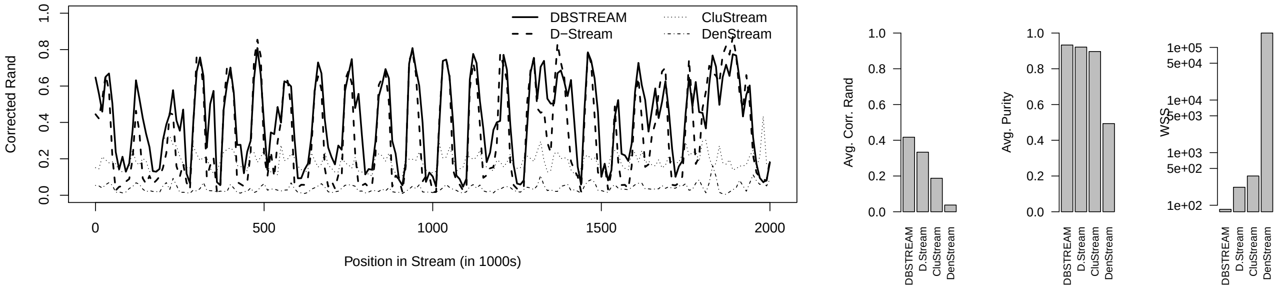


Fig. 6. Sensor data (2 million data points).

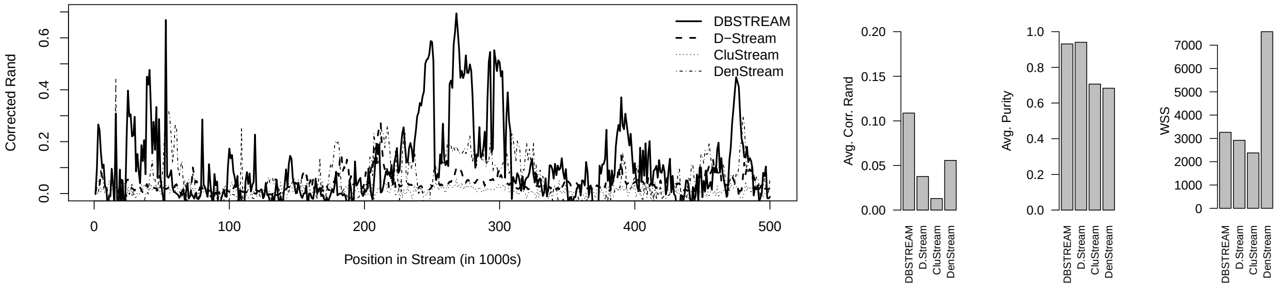


Fig. 7. Clustering quality on the Forest Cover Type data set.

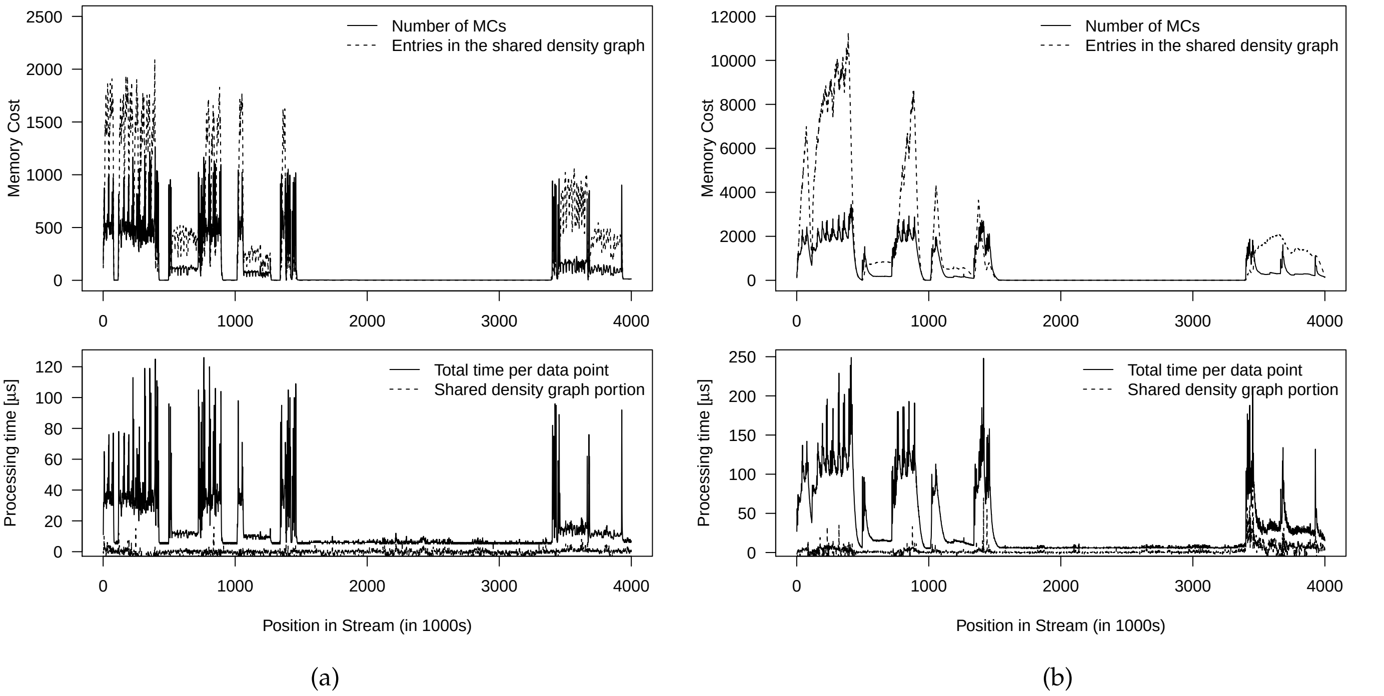


Fig. 8. Memory cost and run time on the KDD Cup'99 data with $r = 1$, $\alpha = 0$, $w_{\text{noise}} = 0$ and for (a) we use $\lambda = 1/1000$ (fast fading) and for (b) $\lambda = 1/10000$ (slow fading). Fading is called every $n = 1000$ points.

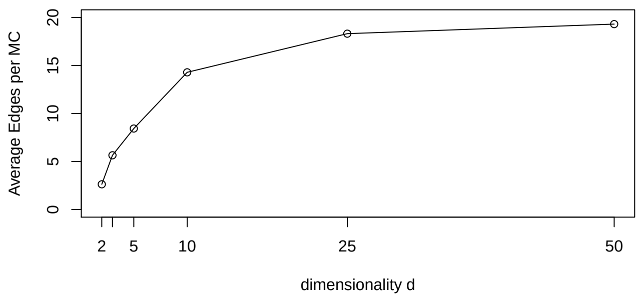


Fig. 9. Average number of edges in the shared density graph for simulated mixture of Gaussians data sets with different dimensionality d.

to zero. Therefore all MCs and entries in the shared density graph will be reported. We report here the results for two settings for λ (slow and fast fading). Figure 8 shows the results. The top plots show the memory cost in terms of the number of MCs and the number of entries in the shared density graph. The bottom graphs show the increase in time needed to maintain the shared density graph relative to the time used by the base-line clustering algorithm. Both settings for λ show similar general behavior with more MCs and connections in the first and third part of the data stream. This reflects the fact that the data only contains data points from a single class between the time instances of roughly 1.5 and 3.5 million. Interesting is that the number of entries in the shared density graph stays very low compared to the worst case given by $k'(k' - 1)$ where k' is the number of MCs. In fact, the experiments show that for the KDD Cup'99 data set the average number of entries per MC in the shared density graph never exceed 3 which is even very low compared to the value of 15–20 obtained in the previous experiment with Gaussians. We can speculate that the data forms lower-dimensional manifolds, which drastically reduces the number of possible neighbors. This is an interesting finding since it means that maintaining the shared density graph is feasible even for high-dimensional data.

For run time we report the total processing time per data point of clustering and recording shared density (averaged over 1000 points) in the two bottom graphs of Figure 8. For comparison, we also show the processing time for just the part that records shared density. As expected, the time required for recording shared density follow the number of entries in the shared density graph and peak during the first 1.5 million data points. Compared to the total time needed to cluster a new data point, the shared density graph portion is negligible. This results from the fact, that recording the graph does not incur any additional search cost, after all fixed-radius nearest neighbors are found for the clustering algorithm.

7 Conclusion

In this paper, we have developed the first data stream clustering algorithm which explicitly records the density in the area shared by micro-clusters and uses this information for reclustering. We have introduced the shared density graph together with the algorithms needed to maintain the graph in the online component of a data stream mining algorithm. Although, we showed that the worst-case memory requirements of the shared density graph grow extremely fast with data dimensionality, complexity analysis and experiments reveal that the procedure can be effectively applied to data sets of moderate dimensionality. Experiments also show that shared-density reclustering already performs extremely well when the online data stream clustering component is set to produce a small number of large MCs. Other popular reclustering strategies can only slightly improve over the results of shared density reclustering and need significantly more MCs to achieve comparable results. This is an important advantage since it implies that we can tune the online component to produce less micro-clusters for shared-density reclustering. This improves performance and, in many cases, the saved memory more than offset the memory requirement for the shared density graph.

Acknowledgments

Work by M. Bolaños was supported in part by a Research Experience for Undergraduates (REU) supplement to Grant No. IIS-0948893 by the National Science Foundation.

The authors would like to thank the anonymous reviewers for their many helpful comments which improved this manuscript significantly.

References

[1] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams," in *Proceedings of the ACM Symposium on Foundations of Computer Science*, 12-14 Nov. 2000, pp. 359–366.

[2] C. Aggarwal, *Data Streams: Models and Algorithms*, ser. Advances in Database Systems, Springer, Ed., 2007.

[3] J. Gama, *Knowledge Discovery from Data Streams*, 1st ed. Chapman & Hall/CRC, 2010.

[4] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. P. L. F. d. Carvalho, and J. a. Gama, "Data stream clustering: A survey," *ACM Computing Surveys*, vol. 46, no. 1, pp. 13:1–13:31, Jul. 2013.

[5] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," in *Proceedings of the International Conference on Very Large Data Bases (VLDB '03)*, 2003, pp. 81–92.

[6] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *Proceedings of the 2006 SIAM International Conference on Data Mining*. SIAM, 2006, pp. 328–339.

[7] Y. Chen and L. Tu, "Density-based clustering for real-time stream data," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2007, pp. 133–142.

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

[9] L. Tu and Y. Chen, "Stream data clustering based on grid density and attraction," *ACM Transactions on Knowledge Discovery from Data*, vol. 3, no. 3, pp. 1–27, 2009.

[10] 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 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'1996)*, 1996, pp. 226–231.

[11] A. Hinneburg, E. Hinneburg, and D. A. Keim, "An efficient approach to clustering in large multimedia databases with noise," in *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*. AAAI Press, 1998, pp. 58–65.

[12] L. Ertoz, M. Steinbach, and V. Kumar, "A new shared nearest neighbor clustering algorithm and its applications," in *Workshop on Clustering High Dimensional Data and its Applications at 2nd SIAM International Conference on Data Mining*, 2002.

[13] G. Karypis, E.-H. S. Han, and V. Kumar, "Chameleon: Hierarchical clustering using dynamic modeling," *Computer*, vol. 32, no. 8, pp. 68–75, Aug. 1999. [Online]. Available: http://dx.doi.org/10.1109/2.781637

[14] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams: Theory and practice," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 3, pp. 515–528, 2003.

[15] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for projected clustering of high dimensional data streams," in *Proceedings of the International Conference on Very Large Data Bases (VLDB '04)*, 2004, pp. 852–863.

[16] D. Tasoulis, N. Adams, and D. Hand, "Unsupervised clustering in streaming data," in *IEEE International Workshop on Mining Evolving and Streaming Data. Sixth IEEE International Conference on Data Mining (ICDM 2006)*, Dec. 2006, pp. 638–642.

[17] D. K. Tasoulis, G. Ross, and N. M. Adams, "Visualising the cluster structure of data streams," in *Advances in Intelligent Data Analysis VII*, ser. Lecture Notes in Computer Science. Springer, 2007, pp. 81–92.

[18] K. Udommanetanakit, T. Rakthanmanon, and K. Waiyamai, "E-stream: Evolution-based technique for stream clustering," in *ADMA '07: Proceedings of the 3rd international conference on Advanced Data Mining and Applications*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 605–615.

[19] P. Kranen, I. Assent, C. Baldauf, and T. Seidl, "The clustree: indexing micro-clusters for anytime stream mining," *Knowledge and Information Systems*, vol. 29, no. 2, pp. 249–272, 2011.

[20] A. Amini and T. Y. Wah, "Leaden-stream: A leader density-based clustering algorithm over evolving data stream," *Journal of Computer and Communications*, vol. 1, no. 5, pp. 26–31, 2013.

[21] J. A. Hartigan, *Clustering Algorithms*, 99th ed. New York, NY, USA: John Wiley & Sons, Inc., 1975.

[22] J. L. Bentley, "A survey of techniques for fixed radius near neighbor searching," Tech. Rep., 1975.

[23] ——, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975.

[24] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo, "A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data," in *Data Mining for Security Applications*. Kluwer, 2002.

[25] M. Hahsler and M. H. Dunham, "Temporal structure learning for clustering massive data streams in real-time," in *SIAM Conference on Data Mining (SDM11)*. SIAM, April 2011, pp. 664–675.

[26] C. Isaksson, M. H. Dunham, and M. Hahsler, "Sostream: Self organizing density-based clustering over data stream," in *Machine Learning and Data Mining in Pattern Recognition*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7376, pp. 264–278.

[27] T. Kohonen, "The self-organizing map," *Neurocomputing*, vol. 21, pp. 1–6, 1998.

[28] ——, "Neurocomputing: Foundations of research," J. A. Anderson and E. Rosenfeld, Eds. Cambridge, MA, USA: MIT Press, 1988, ch. Self-organized Formation of Topologically Correct Feature Maps, pp. 509–521.

[29] J. H. Conway, N. J. A. Sloane, and E. Bannai, *Sphere-packings, lattices, and groups*. New York, NY, USA: Springer-Verlag, 1987.

[30] M. Hahsler, M. Bolanos, and J. Forrest, *stream: Infrastructure for Data Stream Mining*, 2015, R package version 1.2-2.

[31] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: massive online analysis," *Journal of Machine Learning Research*, vol. 99, pp. 1601–1604, August 2010.

[32] 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 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 868–876.

[33] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

[34] J. Gama, R. Sebastião, and P. P. Rodrigues, "On evaluating stream learning algorithms," *Machine Learning*, pp. 317–346, 2013.

[35] A. Bifet, G. de Francisci Morales, J. Read, G. Holmes, and B. Pfahringer, "Efficient online evaluation of big data stream classifiers," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '15. ACM, 2015, pp. 59–68.

Michael Hahsler received his MS degree in business administration and his PhD degree in information engineering and management from the Vienna University of Economics and Business, Austria, in 1998 and 2001, respectively. After receiving his post-doctoral lecture qualification in business informatics in 2006, he joined Southern Methodist University as a visiting professor in computer science and engineering. He currently is an assistant professor in Engineering Management, Information, and Systems, and director of the Intelligent Data Analysis Lab at Southern Methodist University. He also serves as an editor of the Journal of Statistical Software. His research interests include data mining and combinatorial optimization, and he is the lead developer and maintainer of several R extension packages for data mining (e.g., arules, dbscan, stream, TSP). He is a member of the IEEE Computer Society.

Matthew Bolaños received his BS degree with distinction in computer science and engineering from Southern Methodist University in 2014. He worked for three years as an undergraduate research assistant on various data mining projects for the Intelligent Data Analysis Lab at Southern Methodist University where he made significant contributions to the stream R package. He received a master's degree from Carnegie Mellon's Human Computer Interaction Institute and works now for Research Now. His research interests include data stream clustering and user experience design.