Temporal Structure Learning for Clustering Massive Data Streams in Real-Time*

Michael Hahsler † Margaret H. Dunham ‡

Abstract

This paper describes one of the first attempts to model the temporal structure of massive data streams in real-time using data stream clustering. Recently, many data stream clustering algorithms have been developed which efficiently find a partition of the data points in a data stream. However, these algorithms disregard the information represented by the temporal order of the data points in the stream which for many applications is an important part of the data stream. In this paper we propose a new framework called Temporal Relationships Among Clusters for Data Streams (TRACDS) which allows us to learn the temporal structure while clustering a data stream. We identify, organize and describe the clustering operations which are used by state-of-the-art data stream clustering algorithms. Then we show that by defining a set of new operations to transform Markov Chains with states representing clusters dynamically, we can efficiently capture temporal ordering information. This framework allows us to preserve temporal relationships among clusters for any state-of-the-art data stream clustering algorithm with only minimal overhead.

To investigate the usefulness of TRACDS, we evaluate the improvement of TRACDS over pure data stream clustering for anomaly detection using several synthetic and real-world data sets. The experiments show that TRACDS is able to considerably improve the results even if we introduce a high rate of incorrect time stamps which is typical for real-world data streams.

Keywords: Data stream, clustering, temporal structure, Markov chain

1 Problem Specification

Algorithms for clustering data streams [4, 5, 8, 13, 23, 24, 26–28] have focused on many characteristics of stream data (e.g., limited storage but potentially unbounded size of data stream, single pass over the data, near real-time processing, concept drift), but the fact that data arrives in a temporal order, which is perhaps one of the most important aspects of the data stream, is typically not directly used.

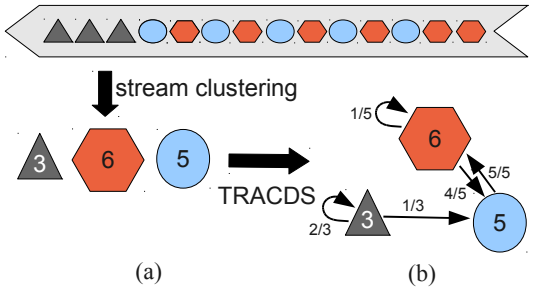


Figure 1: Stream Clustering: (a) Partitioning of a data stream using standard (data stream) clustering neglects the temporal aspect of the data. (b) With TRACDS temporal relationships between clusters are learned dynamically as an evolving Markov chain (transitions between clusters are represented by arcs).

Figure 1(a) illustrates what happens during data stream clustering. The data stream is represented by the ordered sequence of shapes in the upper half of the figure. Different shapes represent events that are similar enough to be put in the same cluster. In the left lower half, the larger symbols represent the clusters annotated with the number of events assigned to each cluster. Since the data volume produced by a data streams is typically unbounded, it is infeasible to store each event assigned to a cluster. Rather, cluster-wide summaries called cluster features or synopses that include descriptive statistics for a cluster (mean, variance, etc.) are stored. During this summarization, any timestamp available for events is either lost or treated as if it were any other attribute. For example, when in the stream in Figure 1 a Hexagon event occurs, the next event is a Circle event $4/5 = 80\%$ of the time (ignoring last event); a Circle is followed by a Hexagon 100% of the time; and a Triangle always follows a Triangle. Although the temporal order of events is one of the salient concepts of stream data, this ordering relationship of the clusters is lost completely after clustering.

One of the major applications using data stream

*To appear in the SIAM Conference on Data Mining 2011 (SDM11)

†Southern Methodist University, mhahsler@lyle.smu.edu

‡Southern Methodist University, mhd@lyle.smu.edu

clustering is rare event (anomaly) detection. When during cluster formation temporal order information is discarded, we might sacrifice important indicators to detect/predict rare event (e.g., intrusion in a computer network, flooding, heat waves, hurricanes, or eruptions of volcanoes based on climate data). For example, for intrusion detection in a large computer network we may observe a user change from behavior type A to behavior type B, both represented by clusters labeled as non-suspicious behavior. However, the transition from A to B might be extremely unusual and itself indicate an intrusion event. This can only be detected if the temporal structure of the data is preserved in the clustering.

We argue that temporal and ordering aspects should be considered as an integral part when performing clustering events in data streams for many applications. However, we are not aware of research that combines clustering with preserving the temporal structure in the data stream that meets the requirements for data stream processing (single-pass over the data, only store synopses for clusters, etc.).

In this paper we present the TRACDS framework which provides a transparent way to add temporal ordering information in the form of a dynamically changing Markov Chain to data stream clustering algorithms (see Figure 1(b)). The framework generalizes the ideas by Dunham et al developed for the Extensible Markov Model (EMM) [11] for data stream clustering. In this paper we identify a set of clustering operations which is sufficient to describe state-of-the-art data stream clustering algorithms and we develop a complete set of TRACDS operations to efficiently manipulate Markov Chains to learn temporal information for clustering. This clean separation between clustering and TRACDS operations makes the framework directly applicable to any state-of-the-art data stream clustering algorithm. We will show that the framework can be implemented to add only minimal overhead to the data stream clustering algorithm and thus it is suitable to handle massive data streams.

While data stream clustering and Markov chains are well know techniques, this paper provides two novel contributions.

1. This paper is one of the first to attempt to model the temporal structure of massive data streams in real-time.

2. We combine data stream clustering and Markov chains which are dynamically changed by a set of operations in a new and efficient way.

This paper is organized as follows. We start with related work in Section 2. In Section 3 we formally introduce the TRACDS framework. In Section 4 we use several synthetic and real-world data sets to analyze the improvement of TRACDS over pure clustering for anomaly detection. Finally, Section 5 discusses directions for future work.

2 Related Work

2.1 Data Stream Clustering. Clustering, the grouping of similar objects, has been around since the beginning of human existence. It was not until the 1800s, however, that formal algorithms were developed. Excellent surveys of clustering techniques have been published that summarize these developments (e.g., [14]). Traditional clustering is often viewed as a partitioning or segmentation of a static data set where the order of observation has no relevance. Dynamic clustering techniques were developed to incrementally cluster data and take into account the temporal nature of data by specifically looking at how the clusters change as data arrives [15].

With the advent of the streaming data concept, many clustering researchers began to adapt clustering techniques to streaming data. The salient features of data streams are that data continues to arrive and that it is impractical to keep all of the data. Data stream management techniques look at various strategies (such as time windows and snapshots) to handle unbounded streams. Clustering techniques must be incremental and usually consider some sort of dynamic updating of the clusters. Barbará [6] identified the requirements for clustering of data streams to include: compactness; "fast, incremental processing" and identification of outliers.

An early algorithm called *STREAM* [13] partitions the data stream into segments, clusters each segment individually by solving the k-medians problem and then iteratively reclusters the resulting centers to obtain a final clustering. Since data streams can potentially grow unbounded, data stream clustering algorithms do not store all data points but rather use a summary or synopsis consisting of aggregate statistics to store information about each cluster. The idea was introduced in the non-data stream clustering algorithm *BIRCH* [30] where the summaries were called cluster feature vectors. The data stream clustering algorithm *CluStream* [4] introduces micro-clusters represented by summary statistics. Micro-clusters are handled by the algorithm online. However, to create a final clustering, micro-clusters have to be merged based on their summary statistics resulting in a simpler clustering. As this reclustering is performed offline, it can be done with any regular clustering algorithm.

An important feature of data streams is that their

structure may change over time. Most of the following clustering algorithms handle change by using an exponential fading model to reduce the weight of older data points. An exponential fading is used since it can easily be applied directly to most summary statistics. *DenStream* [8] maintains micro-clusters in real time and uses a variant of the density-based *GDBSCAN* [21] to produce a final clustering for users. *HPStream* [5] finds clusters that are well defined in different subsets of the dimensions of the data. *WSTREAM* [23] uses a kernel density estimation to find rectangular windows to represent clusters. The windows can move, contract, expand and be merged over time. *E-STREAM* [27] adds cluster splitting by maintaining histograms for each cluster and dimension. While the popular density-based clustering method *OPTICS* [18] is not suitable for data streams, a variant called *OpticsStream* [24] can be used to visualize the clustering structure and structure changes in data streams. Recently, the density-based data stream clustering algorithms *D-Stream* [26] and *MR-Stream* [28] were developed. *D-Stream* uses an online component to map each data point into a predefined grid and then uses an offline component to cluster the grid based on density. *MR-Stream* facilitates the discovery of clusters at multiple resolutions by using a grid of cells that can dynamically be sub-divided into more cells using a tree data structure. Recently, data stream clustering algorithms for massive data [2] and uncertain data [3] have also been introduced.

All these approaches center on finding clusters of data points given some notion of similarity but neglect the temporal ordering structure of the data which might be crucial to understanding the underlying data.

2.2 Incorporating Temporal Order. There have been several research efforts to incorporate temporal order information into clustering. We briefly review some of these concepts prior to introducing what we propose for TRACDS. The term *temporal clustering* is generally used to mean applying clustering to time series data [29]. Typically either several time series are clustered to find sets of similar series or subsequences are clustered to find similar parts in time series [16]. Since we are interested in clustering individual data points while preserving the temporal order, neither of these approaches is applicable. The temporal structure between data points in time series is typically modeled by auto-regressive models (e.g., ARIMA) which is more difficult for multivariate data [25] and typically does not honor the restrictions for data streams (e.g., single pass over the data). We are not aware of research which combines auto-regressive models with clustering massive multivariate data streams while preserving temporal structure efficiently.

Evolutionary clustering [9] considers the problem of clustering data over time with the goal to trade off the two potentially conflicting criteria of representing the current data as faithfully as possible by the clustering while preventing dramatic shifts in the clustering from one timestep to the next. Similarly, the *MONIC* framework [22] uses the term *cluster transitions* to refer to the fact that a cluster may change over time (e.g., change its density, move or merge with another cluster). While these approaches deal with the fact that there is a need to detect and evaluate these changes, they still largely ignore the order information inherent in the data stream.

C-TREND [1] captures the temporal ordering concept among clusters with transitions between clusters obtained based on predefined temporal partitions. It tries to tackle a problem similar to what we address in this paper, however it suffers from many restrictions. Especially, C-TREND is not suited for data streams as a fixed number of partitions must be created before classical clustering algorithms which need all data are used. Also only transitions between partitions in time are identified and all transitions between clusters within each partition are ignored.

In the following we introduce TRACDS, a framework which is able to efficiently capture temporal order information for data stream clustering.

3 Introduction to TRACDS

Although TRACDS does not implement data stream clustering itself, we have to introduce the concepts before moving on to our framework. Clustering is typically thought of in terms of partitioning a data set consisting of observations into several (typically k, a predefined number) groups of similar observations where observations in different groups are less similar. Formally a clustering can be defined as:

DEFINITION 3.1. (CLUSTERING) *A clustering ζ is a partitioning of a data set $\mathbf{D}$ into k subsets $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ called clusters such that*

(1) $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$ *for all* $i \neq j$,

(2) $\bigcup_{i=1}^{k} \mathcal{C}_i \subseteq \mathbf{D}$, *and*

(3) the value of a specified cost function $f_c(\zeta)$ is minimized (typically by a heuristic).

The requirement that clusters do not share data points means that we deal with crisp and not soft partitions where a data point could be assigned to several partitions typically with varying degree of membership.

In the above definition we do not require that all data points in $\mathbf{D}$ are assigned to a cluster. Some data points might be labeled as outliers and stay unassigned.

Data stream clustering algorithms take into account that for many applications data is arriving continuously and that the number of clusters may not be known in advance. To deal with the fact that data streams may produce more data than is practical to store, data stream clustering algorithms work with synopses for clusters (sometimes called clustering features or CFs) instead of keeping all the data points.

Definition 3.2. (Data Stream Clustering) *At each point in time t a data stream clustering ζ_t is a partitioning, as defined in Definition 3.1, of $\mathbf{D}_t$, the data seen thus far, into k components. However, instead of all data points assigned to clusters $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ only synopses $\vec{c}_1, \vec{c}_2, \ldots, \vec{c}_k$ are stored and k is allowed to change over time. The synopses $\vec{c}_i$ with $i = 1, 2, \ldots, k$ contain summary information about the size, distribution and location of the data points in $\mathcal{C}_i$.*

Still, data stream clustering algorithms only partition the data and temporal aspects (e.g., order or timestamps) are not preserved in the clustering (with the exception that old data might be removed or weighted to be able to reflect concept drift). We propose to model the temporal structure between clusters as an evolving *Markov Chain (MC)* which at each point in time represents a regular time-homogeneous MC, but which is updated using a set of well defined operations when new data is available. In the following we will restrict the discussion to first order evolving MCs. First order MCs work well as an approximation for many applications, however, as for regular MCs, it is possible to extend the idea to higher order models [17].

A (first order) discrete parameter Markov Chain [19] is a special case of a Markov Process in discrete time and with a discrete state space. It is characterized by a sequence of random variables $\{X_t\} = \langle X_1, X_2, X_3, \ldots \rangle$ with t being the time index. All random variables share the same domain $\text{dom}(X_t) = S = \{s_1, s_2, \ldots, s_k\}$, a set called the state space. The *Markov property* states that for a first order model the next state is only dependent on the current state. Formally,

$$P(X_{t+1} = s_{t+1} \mid X_t = s_t, \ldots, X_1 = s_1) = P(X_{t+1} = s_{t+1} \mid X_t = s_t),$$

where $s_1, \ldots, s_t, s_{t+1} \in S$.

For simplicity we use for transition probabilities the notation $a_{ij} = P(X_{t+1} = s_j \mid X_t = s_i)$, $i, j = 1, 2, \ldots, k$, where it is appropriate. Time-homogeneous MC can be represented as a $k \times k$ transition matrix $\mathbf{A} = (a_{ij})$ containing the transition probabilities from each state to all other states. Another representation is as a graph with the states as vertices and the arcs labeled with transition probabilities. Transition probabilities can be easily estimated from the observed transition counts c_{ij} (transitions from state s_i to s_j) using the maximum likelihood method by $a_{ij} = c_{ij}/n_i$ where $n_i = \sum_{j=1}^{k} c_{ij}$.

MCs are very useful to keep track of temporal information using the Markov property as a relaxation. With a MC it is easy to predict the probability of future states or predict missing values based on the temporal structure of the data. It is also easy to calculate the probability of a new sequence of length l given a MC as the product of transition probabilities:

$$P(X_l = s_l, X_{l-1} = s_{l-1}, \ldots, X_1 = s_1) = P(X_1 = s_1) \prod_{i=1}^{l-1} P(X_{i+1} = s_{i+1} \mid X_i = s_i)$$

Data streams typically contain dimensions with continuous data and/or have discrete dimensions with a large number of domain values [2]. In addition, the data may continue to arrive resulting in a large number of observations. For the MC we use in TRACDS, data points have to be mapped onto a manageable number of states. This mapping is already done by the used data stream clustering algorithm. The clustering algorithm assigns each point to a cluster (or micro-cluster) which is represented by a state in the MC.

Definition 3.3. (TRACDS) *TRACDS is defined at each point in time t as a tuple $\mathcal{T} = (S, \mathbf{C}, s_c)$ (we omit subscript t for readability), where S is the set of k states, $\mathbf{C}$ is the $k \times k$ transition count matrix specifying the MC over the states in S and $s_c \in S$ keeps track of the current state, i.e., the state to which the last observation was assigned to. Given a data stream clustering ζ, TRACDS has the following properties:*

(1) At each point in time t there is a one-to-one correspondence between the clusters in ζ and the states S.

(2) The transition probability a_{ij} estimated by the transition counts in $\mathbf{C}$ represents the probability that given a data point in cluster i, the next data point in the data stream will belong to cluster j with $i, j = 1, 2, \ldots, k$.

(3) $\mathcal{T}$ is created online in parallel to the data stream clustering ζ.

(4) An empty clustering with no data points is represented by an empty $\mathbf{C}$ with $S = \emptyset$ and we define s_c to be ϵ to indicate that there is no current state.

In order to satisfy the properties in Definition 3.3 $\mathcal{T}$ has to evolve over time to reflect all changes to the clustering. Note that since k in the clustering can change over time also the number of states in $\mathcal{T}$ has to adapt. In the following we will identify all operations that data stream clustering algorithms perform and present a set of well defined operations, called TRACDS operations, which are used to update $\mathcal{T}$ accordingly.

3.1 Data Stream Clustering Operations. The *MONIC* framework [22] deals with the evolution of clusters over time. It is not suitable for data streams as it is not online and does not support relationships between clusters at a given point in time. However, it presents a technique which is independent of the used clustering algorithm and in that sense is similar to TRACDS. MONIC identifies so-called external and internal cluster transitions reflecting the change from a clustering at time point t to a later clustering at $t+1$ (e.g., cluster survives, cluster is absorbed, cluster moves). Although these cluster transitions reflect the changes of clusters between two points in time, they are a good starting point to identify typical building blocks (we call clustering operations) of data stream clustering algorithms. Such building blocks are, for example, adding a new incoming data point to an existing cluster or creating a new cluster. Formally a clustering operation is defined as:

Definition 3.4. (Clustering Operation) *A clustering operation is defined as a function*

$$\zeta_{t+1} = q(\zeta_t, x),$$

which is used by the data stream clustering algorithm to update the clustering given additional information x (a new data point, the index of the cluster to be deleted, etc.).

Any data stream clustering algorithm can be described as a sequence of such clustering operations that are triggered by the specifics of the clustering algorithm itself. We identify the necessary clustering operations for state-of-the-art data stream clustering algorithms in Table 1. In the following we will define these operations. Some clustering operations are triggered by the data stream clustering algorithm when a new data point is available for the data stream. The two typical operations are:

- $q_{\text{assign}}(\zeta, x)$: *Assign the new data point x to an existing cluster.* The clustering algorithm uses the cluster summaries in ζ to find the appropriate cluster i and then updates $\vec{c}_i$.
- $q_{\text{create}}(\zeta, x)$: *Create a new cluster.* At some point (e.g., if assigning a new data point to an existing cluster is not appropriate) a new empty cluster represented by $\vec{c}_{k+1}$ is added to ζ and k is incremented by one.

Several operations can be triggered by other events. For example by a clean-up process which is scheduled at regular intervals or by the clustering algorithm when it runs out of memory. These include:

- $q_{\text{remove}}(\zeta, x)$: *Remove a cluster.* Here x is i, the index of the cluster to be removed. In this case the associated summary $\vec{c}_i$ is removed from ζ and k is decremented by one.
- $q_{\text{merge}}(\zeta, x)$: *Merge two clusters.* Here input x contains i and j, the indices of two clusters to be merged. First a new merged cluster is created by combining the two summaries $\vec{c}_i$ and $\vec{c}_j$ in an appropriate way. Then the two merged clusters are removed.
- $q_{\text{fade}}(\zeta, x)$: *Fade the cluster structure.* This adapts the cluster structure over time by reducing the influence of older data points. The input x is empty in this operation as it is a clustering wide function. Since only a summary and not the original data points are available, fading has to be done on this summary information by updating each summary $\vec{c}_i$ for $i = 1, 2, \ldots, k$. Typically (see [5, 8, 23, 24]) a decay function $f(t) = 2^{-\lambda t}$ is used to specify the weight of data points added t timesteps in the past. This fading can be done iteratively on summary statistics if they exhibit the properties of additivity and temporal multiplicity defined in [5].
- $q_{\text{split}}(\zeta, x)$: *Split a cluster.* Given i, an input cluster index, two new clusters, j and l with appropriate summaries, $\vec{c}_j$ and $\vec{c}_l$, are created. Subsequently i is removed.

Note that the actions of merging and fading imply that the clustering summaries themselves should be additive so that they can be combined during the merge operation. This was a salient feature of the clustering features proposed in BIRCH. If not additive, then some known function must exist to combine them. Also notice that a reclustering operation, which is used in many stream clustering algorithms, is accomplished by a series of merges. The split operation is currently only supported by E-Stream [27] to split a large cluster up into several smaller and denser clusters.

The exact procedure of how and why operations like deleting and merging are executed is defined by the

Operation	*CluStream*	*HPStream*	*DenStream*	*WSTREAM*	*OpticsStream*	*E-Stream*	*D-Stream*	*MR-Stream*
q_{assign}	x	x	x	x	x	x	x	x
q_{create}	x	x	x	x	x	x	x	x
q_{remove}	x	x	x	x	x	x	x	x
q_{merge}	offline		offline	x	offline	x	offline	offline
q_{fade}		x	x	x	x	x	x	x
q_{split}						x		

Table 1: Clustering operations used by data stream clustering algorithms (marked with x). *Offline* in row q_{merge} indicates that merging is only used as an offline reclustering step.

used data stream clustering algorithm. For TRACDS it is only important that we can define operations to keep $\mathcal{T}$ consistent with the clustering.

3.2 TRACDS Operations. As indicated earlier, TRACDS operations are triggered by the stream clustering operations stated above. Each clustering operation triggers a unique TRACDS operation which updates $\mathcal{T}$. As x is used to indicate the input to the stream clustering operations, we use y to indicate the input to the TRACDS operations. y is uniquely determined by the clustering operation.

Definition 3.5. (TRACDS Operation) *A TRACDS operation is a function*

$$\mathcal{T}_{t+1} = r(\mathcal{T}_t, y),$$

which is used to update the TRACDS data structure using information y provided by a clustering operation (defined above).

In the following we will define for each clustering operation q a unique TRACDS operation r. We use the same subscript to identify which TRACDS operation corresponds to which clustering operation.

- $r_{\text{assign}}(\mathcal{T}, y)$: *Record a state transition.* y identifies s_i, the state corresponding to the cluster the new data point was assigned to in ζ. If the current state is known ($s_c \neq \epsilon$), update $\mathbf{C}$ by setting $c_{s_c,s_i} \leftarrow c_{s_c,s_i} + 1$. Finally, we set the current state to the new state $s_c \leftarrow s_i$.

- $r_{\text{create}}(\mathcal{T}, y)$: *Create a new state.* y is empty in this case. To represent the new cluster, we have to add a state s_{k+1} to $\mathcal{T}$ by $S \leftarrow S \cup \{s_{k+1}\}$. We enlarge $\mathbf{C}$ by a row and a column for this state, and finally, we set the current state to the newly added state $s_c \leftarrow s_{k+1}$.

- $r_{\text{remove}}(\mathcal{T}, y)$: *Remove state.* To remove state s_i, identified in y, let $S \leftarrow S \setminus \{s_i\}$ and remove the row i and column i in the transition count matrix $\mathbf{C}$. This deletes the state. If s_c is s_i, then set the current state to the no state $s_c \leftarrow \epsilon$.

- $r_{\text{merge}}(\mathcal{T}, y)$: *Merge two states.* To merge two states s_i and s_j, input in y, into a new state s_m, the following steps are performed:

  1. Create new state s_m (see r_{create} above).
  2. Create the outgoing and incoming arcs for s_m by for all $l \in \{1, 2, \ldots, k\}$ let $c_{ml} \leftarrow c_{il} + c_{jl}$ and $c_{lm} \leftarrow c_{li} + c_{lj}$.
  3. Delete the old states s_i and s_j (see r_{remove} above).
  4. If s_c is either s_i or s_j, then set the current state to the new state $s_c \leftarrow s_m$.

- $r_{\text{fade}}(\mathcal{T}, y)$: *Fade the transition probabilities which represent the cluster structure.* y is empty in this case. The fading strategy used on the cluster synopses by the data stream clustering algorithm must also be used on the transition count matrix $\mathbf{C}$ resulting in a fading effect consistent with the clustering. Cluster algorithms typically use exponential decay $f(t) = 2^{-\lambda t}$ which is multiplicative and using repeatedly

  $$\mathbf{C}_{t+1} = 2^{-\lambda}\, \mathbf{C}_t$$

  results in the desired compounded fading effect.

- $r_{\text{split}}(\mathcal{T}, y)$: *Split states.* y contains s_i, where i is the index of the cluster to be split. As with fading, the splitting strategy used must be consistent with the one implemented by the clustering algorithm. Since only synopses information is available for the clustering as well as for the transition counts only some heuristic can be used here (e.g., assign the transition counts proportionally to the number of observations assigned to each new cluster).

The following example illustrates the use of some of the TRACDS clustering operations.

Table 2: Sequence of operations for Example 1

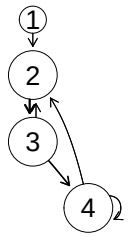
Cluster assignment	TRACDS operation	Manipulation of $\mathbf{C}$	s_c
	initial	$\mathbf{C}$ is 0×0	ϵ
1	$r_{new\ cluster}$	expand $\mathbf{C}$ to 1×1	
	$r_{assign\ point}$	no manipulation	1
2	$r_{new\ cluster}$	expand $\mathbf{C}$ to 2×2	
	$r_{assign\ point}$	$c_{1,2} \leftarrow c_{1,2} + 1$	2
3	$r_{new\ cluster}$	expand $\mathbf{C}$ to 3×3	
	$r_{assign\ point}$	$c_{2,3} \leftarrow c_{2,3} + 1$	3
2	$r_{assign\ point}$	$c_{3,2} \leftarrow c_{3,2} + 1$	2
3	$r_{assign\ point}$	$c_{2,3} \leftarrow c_{2,3} + 1$	3
4	$r_{new\ cluster}$	expand $\mathbf{C}$ to 4×4	
	$r_{assign\ point}$	$c_{3,4} \leftarrow c_{3,4} + 1$	4
4	$r_{assign\ point}$	$c_{4,4} \leftarrow c_{4,4} + 1$	4
2	$r_{assign\ point}$	$c_{4,2} \leftarrow c_{4,2} + 1$	2
3	$r_{assign\ point}$	$c_{2,3} \leftarrow c_{2,3} + 1$	3
4	$r_{assign\ point}$	$c_{3,4} \leftarrow c_{3,4} + 1$	4

EXAMPLE 1. (CREATE A TRACDS) *A data stream clustering algorithm starts with an empty clustering ζ and we also start with an empty TRACDS data structure $\mathcal{T}$ with no states $S = \emptyset$ and the starting state s_c set to ϵ indicating that there is no state yet. We assume that the clustering algorithm assigns ten incoming data points the clusters 1, 2, 3, 2, 3, 4, 4, 2, 3, 4. This sequence of assignments triggers the 14 TRACDS operations shown in Table 2. The table shows the assumed cluster assignments by the clustering algorithm, the executed TRACDS operations and manipulations to $\mathbf{C}$ and s_c. We have 10 operations to assign points to clusters and 4 operations to create the 4 needed clusters/states. Creating new clusters/states increases the size of $\mathbf{C}$ and adding a data point increases the counts inside the matrix. The transition count matrix $\mathbf{C}$ resulting from the 14 operations is shown in Figure 2(a). A graph representation of the transition count matrix is shown in Figure 2(b). The nodes are labeled with the cluster/state labels 1–4 and larger nodes represent clusters with more observations. The arcs represent transitions with heavier arcs indicating a higher transition count.*

3.3 Implementation and Complexity of TRACDS. TRACDS operations can be implemented separately from the clustering operations. We only need a very light-weight interface through which we can observe what operations the clustering algorithm executes plus minimal additional information (e.g., the cluster index for a new data point). Adding such an interface to existing clustering algorithms is straight forward.

	1	2	3	4
1	0	1	0	0
2	0	0	3	0
3	0	1	0	2
4	0	1	0	1

(a)



(b)

Figure 2: (a) Transition count matrix $\mathbf{C}$ and (b) graph representation of the Markov Chain for Example 1

The space and time complexity of maintaining TRACDS depends on the data structure used to store the transition count matrix $\mathbf{C}$. A suitable data structure is a two-dimensional array with $k' \geq k$ columns/rows where we allow columns and the corresponding rows to be marked as currently unused. On such a data structure the most often used operation of assigning a new data point can be done in constant time. Deleting clusters, creating new clusters and merging clusters takes $O(k)$. We only have to reorganize the data structure with $O(k^2)$ if no more unused rows/columns are available. Note that most data stream clustering algorithms make sure that k does not increase unbounded which reduces or might even avoid the need for the more expensive reorganization operation. Fading is also a more expensive operation since we have to go through all k^2 cells. However, a strategy similar to the one used in [28] can be used to significantly reduce the burden by using timestamps when the last fading was performed on a count and then only do compounded fading when a transition is updated.

Space requirements are $O(k'^2)$ where $k' \geq k$ is the chosen size for $\mathbf{C}$. However, since for certain types of data $\mathbf{C}$ might be very sparsely populated it is possible to use a list-based data structure to reduce the space requirements at the expense of the time complexity of the operations.

The computational needs of TRACDS directly depend on the number and type of clustering operations executed by the used data stream clustering algorithm. Our experiments show that the time spent on TRACDS operations is negligible compared to the clustering operations (see next section).

4 Evaluation

In this section we evaluate the improvement of TRACDS compared to standard stream clustering. Al-

though TRACDS is potentially useful for other applications we focus here on the improvement for anomaly detection. Anomaly detection is an important and well researched stream mining task. It is the basis for many applications like anomaly detection in weather related data and intrusion detection in computer networks. Although many anomaly detection techniques were proposed (see [10] for a current survey), we only consider a data stream clustering based approach here since we are only interested in analyzing if TRACDS can improve over standard clustering.

As the baseline for unsupervised anomaly detection via clustering, we follow the simple approach by Eskin et al [12]. Clustering with a fixed distance threshold around a center is used to get local density estimates and all members assigned to a cluster i are classified as outliers if the density in the cluster is low, i.e., $n_i < \delta_c$ where n_i is the number of points assigned to cluster i and δ_c is a suitable threshold. This approach was found to perform favorably compared to more complicated and computationally expensive approaches using support vector machines and k-Nearest Neighbor [12].

For TRACDS we only use temporal information given by transition probabilities in the MC in $\mathcal{T}$. TRACDS also has access to cluster size which might further improve results, but we ignore it here since we want to concentrate only on the captured temporal order information. We classify each data point as an anomaly if the transition probability from cluster i, the cluster for the previous data point in the stream, to the cluster of the current point j is below a pre-specified threshold, i.e., $a_{ij} < \delta_T$.

As the data stream clustering algorithm we use the threshold nearest neighbor (tNN) algorithm we implemented together with TRACDS in the R-extension package *rEMM*[1] This simple data stream clustering algorithm is very similar to the one used in [12]. It represents clusters as synopses containing the position and the size of each cluster. It assigns a new data point to an existing cluster if it is within a fixed threshold of its center. If a data point cannot be assigned to any existing cluster, a new cluster is created for the data point. After assignment the data point is discarded. Note that the clustering algorithm used here is only of secondary importance. TRACDS only uses the clustering results as input and we evaluate if TRACDS can improve the results of pure clustering. Other data stream clustering algorithms which might produce better results for anomaly detection will also improve the accuracy of TRACDS.

[1] http://CRAN.R-project.org/package=rEMM

4.1 Synthetic data. We first use several synthetic data sets to evaluate our approach and analyze sensitivity to data dimensionality and imperfections in the temporal structure, e.g., caused by data points arriving out-of-order, incorrect time stamps or by a weak temporal structure in the data. To make the experiments in this paper reproducible, we included the used data generator in package *rEMM*. The data generation process creates a data set consisting of k clusters in roughly $[0,1]^d$. For simplicity, the data points for each cluster are drawn from a multivariate normal distribution given a random mean and a random variance/covariance matrix for each cluster. The temporal aspect is modeled by a fixed subsequence through the k clusters of length n. In each step we have a transition probability p_t that the next data point is in the same cluster or in a randomly chosen other cluster, thus we can create slowly or fast changing data. For the complete sequence, the subsequence is repeated l times creating a fixed temporal structure. The data set of size $N = nl$ is generated by drawing a data point from the cluster corresponding to each position in the sequence. To introduce imperfections in the temporal sequence (i.e., incorrect time stamps) we swap two consecutive observations with probability p_s. Note that each swap of two observations influences three transitions and a rather low setting of p_s can distort the temporal structure significantly.

Anomalies are introduced by replacing data points in the data set with probability p_a by randomly chosen data points in $[0,1]^d$. These data points potentially lie far away from clusters and can be easily detected. However, they might also fall within existing clusters and therefore are hard to detect. Since these anomalies violate the temporal structure of the data, TRACDS can be used for detection.

An example of synthetic data generated by the procedure described above and used in the experiments below is shown in Figure 3. The clusters have different shapes, densities and most are not well separated. Most of the anomalies are far away from the clusters but several are close or even within clusters of regular data so it is expected that detecting these anomalies, if possible at all, is a hard task. Figure 4 shows the *Receiver Operator Characteristics (ROC)* curves [20] with false positive rate (FPR) and true positive rate (TPR) for the two anomaly detection approaches. The ROC curve is formed by connecting the FPR/TPR combinations obtained using different values for the two algorithms thresholds, δ_c and δ_T. It can clearly be seen that TRACDS improves the results over simple clustering resulting in a larger area under the ROC curve (AUC).

For the following experiments we create several data

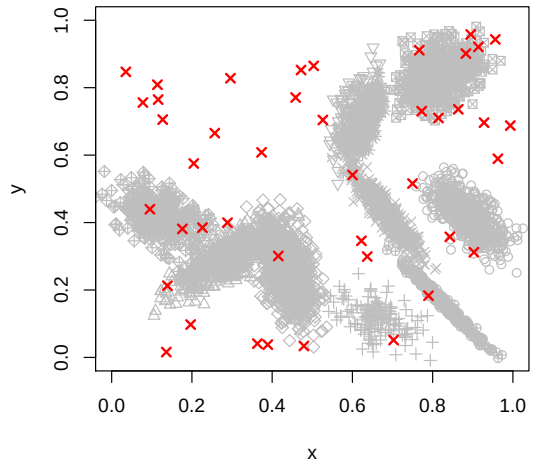


Figure 3: Example of synthetic data ($k = 10$, $d = 2$) used in the first experiment below (first row in Table 3). Anomalies are shown as Xs.

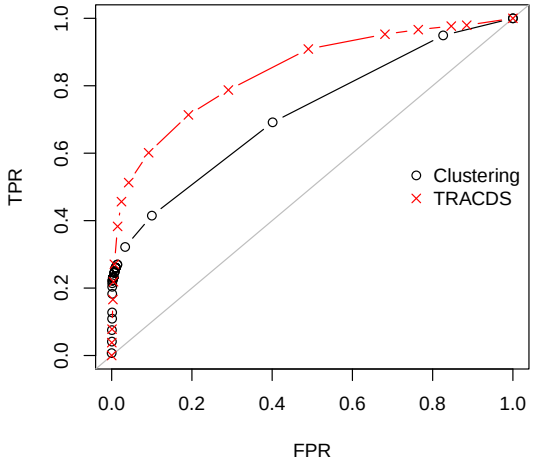


Figure 4: Averaged ROC curves for 10 samples used in the first experiment below (first row in Table 3).

sets with $k = 10$ clusters and an anomaly probability $p_a = 0.01$. The other parameters (n, N, d, p_t and p_s) are varied in the study. We will always create 10 data sets with identical parameters, run clustering and TRACDS on each, average the ROC curves and then report the AUC and the improvement (in percent) of TRACDS over clustering. For the tNN clustering we use Euclidean distance and chose the threshold automatically as the first quartile of the distance distribution in a sample of the data set.

Table 3 summarizes the result for slowly changing data. The probability p_t that a new data point will not belong to the cluster of the previous data point is 50%. For $d = 2$, the AUC for TRACDS is between

Table 3: Slowly changing data ($n = 100$ and $p_t = 0.5$).

d	p_s	N	AUC_{clust}	AUC_{TRACDS}	Improvement
2	0.0	10,000	0.751	0.855	13.8%
2	0.2	10,000	0.753	0.836	11.0%
5	0.0	10,000	0.859	0.936	9.0%
5	0.2	10,000	0.875	0.929	6.1%
10	0.0	10,000	0.904	0.957	5.9%
10	0.2	10,000	0.897	0.942	5.0%

Table 4: Fast changing data ($n = 100$, $p_t = 1$).

d	p_s	N	AUC_{clust}	AUC_{TRACDS}	Improvement
2	0.0	10,000	0.723	0.795	9.9%
2	0.2	10,000	0.744	0.774	3.9%
5	0.0	10,000	0.845	0.890	5.4%
5	0.2	10,000	0.846	0.873	3.2%
10	0.0	10,000	0.877	0.909	3.7%
10	0.2	10,000	0.870	0.881	1.2%

Table 5: No temporal structure ($n = N$ and $p_t = 1$).

d	p_s	N	AUC_{clust}	AUC_{TRACDS}	Improvement
2	0.0	10,000	0.728	0.746	2.4%
5	0.0	10,000	0.816	0.826	1.2%
10	0.0	10,000	0.841	0.845	0.4%

Table 6: Real-world data sets.

Name	N	outliers	d	AUC_{clust}	AUC_{TRACDS}	Impr.
KDD-99	250,000	10	38	0.893	0.972	8.9%
16S	402	43	64	0.516	0.696	34.9%

13.8% larger than for clustering. Introducing temporal errors by swapping data points with a probability of 20% reduces the improvement to 11%. Increasing the dimensionality of the data to $d = 5$ and $d = 10$ results in overall higher AUC for clustering and TRACDS. However, the relative advantage of TRACDS is reduced. This happens since by increasing the dimensionality the volume of the unit hypercube increases exponentially and thus the chance that a random point falls inside a cluster and thus is hard to identify by clustering is reduced.

For Table 4 we changed p_t to one. So it is less likely that two consecutive data points come from the same cluster resulting in faster changing data. This change only affects the temporal structure and thus has little influence on the results from clustering alone. However, it reduces the advantage of TRACDS significantly to 9.9% (3.9% for $p_s = 0.2$) at $d = 2$. Again AUC increases with the dimensionality of the data but the relative advantage of TRACDS is decreasing.

Finally, in Table 5, we set the subsequence length $n = N$ which means that there is no repeating temporal structure in the data. TRACDS performs very similar to clustering. Outlier clusters are still detected by TRACDS since the probability of a transition to these

clusters is smaller than to the regular clusters.

4.2 Real-world data. To investigate if the temporal structure learned by TRACDS is useful for real-world applications we use two data sets. The first dataset is the well known 1999 KDD Cup intrusion detection dataset (KDD-99) obtained from the UCI Machine Learning Repository[2]. It contains network traffic data aggregated at the connection level. We consider the first 250,000 connections in the dataset with roughly 85% normal connections and the remaining connections constitute 10 intrusions of different types (e.g., buffer overflow, perl, smurf, loadmodule, neptune). We use the 38 numeric attributes and set the clustering threshold equal to the median distance between a sample of the data points. Then we used the same procedure as for the synthetic data to create ROC curves. The ROC curves are shown in Figure 5 and the area under the ROC curves is reported in Table 6. TRACDS improves the area by almost 9% indicating that the learned temporal structure preserves important signals for intrusion detection.

The second data set consists of genetic sequences. Such sequences are not temporal data streams, however due to the large volume of data that has to be processed (human DNA contains about 3 billion base pairs), using efficient data stream mining techniques which only need one pass over the data is becoming increasingly important. Here we use 16S rRNA sequences, an important marker for phylogenetic studies, for the two phylogenetic classes Alphaproteobacteria and Mollicutes. The used data is preprocessed by counting the occurrence of triplets of nucleotides (4 distinct bases) in windows of length 100 resulting in data points with 64 counts. The preprocessed data set is included in package *rEMM*. From a set of 30 sequences for each class we randomly choose the data points for 24 sequences for Alphaproteobacteria (normal cases) and then add 3 sequences for Mollicutes (anomalies) at random positions. We cluster the data and compare again the AUC. We repeat this procedure 10 times and report the averages in Figure 6 and Table 6. Finding the anomalies here is a very hard task since many subsequences for both phylogenetic classes are very similar. This results in an AUC for simple clustering close to .5. For some thresholds the clustering approach even produces worse results than picking anomalies randomly. However the sequence information retained by TRACDS improves the results significantly by almost 35%.

Finally, we look at the execution time of TRACDS. We report here the execution time on the KDD-99 data.

[2]http://archive.ics.uci.edu/ml/

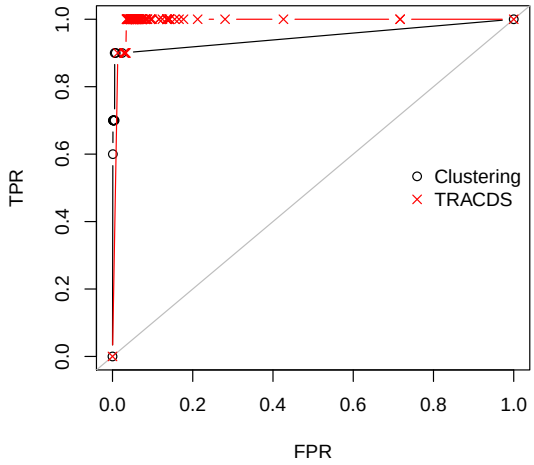


Figure 5: ROC curves for the KDD-99 data set.

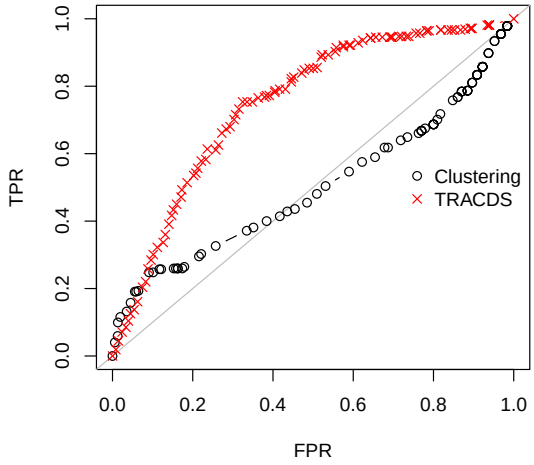


Figure 6: Averaged ROC curves for 10 runs of the 16S rRNA data set.

We run the experiment using our R implementation on an Intel Core 2 (1.5 GHz) machine. Figure 7 shows that the execution time for TRACDS (including the underlying clustering algorithm) increases linearly with the number of data points. Compared with the execution time of just the clustering algorithm, TRACDS only increases the execution time by a negligible amount. As discussed above, the most important operations (insert data point, create new state) take for TRACDS $O(1)$ while the used clustering algorithm has to compute the distances of each new data point to all clusters using $O(kd)$ time, where d is the dimensionality of the data.

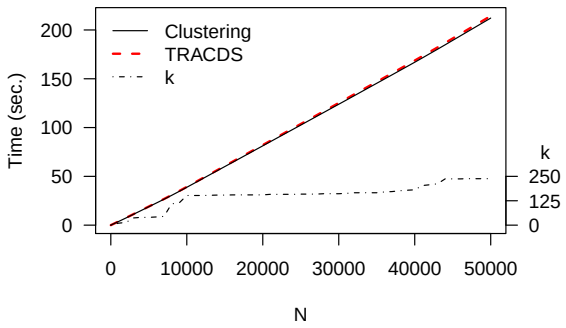


Figure 7: Execution time on KDD-99 data set.

5 Conclusion and Future Work

Current work in the temporal clustering field concentrates on the evolution of data streams and how the clustering can follow or detect such changes. In this paper we addressed a completely different problem. We deal with the temporal (or order) relationship between clusters. We presented TRACDS, a framework which can be used to efficiently learn online a model of a massive data stream's temporal structure.

We systematically evaluated TRACDS for anomaly detection on synthetic data with the result that if there is a strong temporal structure in the data, TRACDS can improve accuracy significantly. Experiments on real-world data support these findings and show that the TRACDS framework only minimally increases runtime.

Since this is one of the first papers dealing with modeling the temporal order structure of massive data streams, there are many directions for further research and applications:

- We will study the use of different data stream clustering algorithms as the base for TRACDS. Since the algorithms have different strategies for assigning data points, and for merging, removing and fading clusters, we need to evaluate the impact on performance in terms of runtime and accuracy for different applications and data structure choices.

- We plan to investigate the use of higher order Markov models to learn a more detailed temporal structure. The space complexity for storing the complete transition matrix increases exponentially with the order of the chain. Also difficulties with getting reliable transition probability estimates for higher order models are expected. However, we can deal with these problems. For example, fading transition counts and then removing low counts can help with avoiding the estimation problem. It can be seen as removing noise and the resulting transition matrix can be relatively sparse and thus can be stored in a more compact way. Also we can resort to use lower order transition probabilities for transitions where not enough data is available to reliably estimate the higher order probabilities (see variable order Markov chains [7]).

- It is straight forward to calculate the probability of future states given the current state and a Markov chain. The Markov chain learned by TRACDS can thus be used to predict the cluster a future data point will belong to. An example application is to impute missing values in a stream by identifying the cluster which a data point most likely would have been assigned to given the temporal structure and then to use the cluster's center as the imputed value.

- Since TRACDS learns a temporal model in form of a Markov chain, we can evaluate dissimilarities between data streams by using dissimilarities between the learned models. Applied to genetic sequences, this will lead to new computationally efficient approaches of sequence clustering and sequence classification for high volume genetic sequence data based on TRACDS models.

Acknowledgments

This work is supported in part by the U.S. National Science Foundation under contract number IIS-0948893.

References

[1] G. Adomavicius and J. Bockstedt. C-TREND: Temporal cluster graphs for identifying and visualizing trends in multiattribute transactional data. *IEEE Transactions on Knowledge and Data Engineering*, 20(6):721–735, June 2008.

[2] C. Aggarwal. A framework for clustering massive-domain data streams. In *IEEE 25th International Conference on Data Engineering (ICDE '09)*, pages 102–113, March 29 2009-April 2 2009.

[3] C. Aggarwal and P. Yu. A framework for clustering uncertain data streams. In *IEEE 24th International Conference on Data Engineering (ICDE 2008)*, pages 150–159, 2008.

[4] 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)*, pages 81–92, 2003.

[5] 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 Thirtieth International Conference on Very Large Data Bases (VLDB '04)*, pages 852–863, 2004.

[6] D. Barbará. Requirements for clustering data streams. *SIGKDD Explorations*, 3(2):23–27, 2002.

[7] R. Begleiter, R. El-Yaniv, and G. Yona. On prediction using variable order markov models. *Journal of Artificial Intelligence Research*, 22:385–421, 2004.

[8] 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*, pages 328–339. SIAM, 2006.

[9] D. Chakrabarti, R. Kumar, and A. Tomkins. Evolutionary clustering. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 554–560. ACM, 2006.

[10] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):1–58, 2009.

[11] M. H. Dunham, Y. Meng, and J. Huang. Extensible markov model. In *Proceedings IEEE ICDM Conference*, pages 371–374. IEEE, November 2004.

[12] 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.

[13] 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*, 15(3):515–528, 2003.

[14] A. Jain, M. Murty, and P. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, September 1999.

[15] Y. Kambayashi, T. Hayashi, and S. Yajima. Dynamic clustering procedures for bibliographic data. In *Proceedings of the ACM SIGIR Conference*, pages 90–99, June 1981.

[16] E. Keogh, J. Lin, and W. Truppel. Clustering of time series subsequences is meaningless: Implications for previous and future research. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, page 115. IEEE Computer Society, 2003.

[17] M. Kijima. *Markov Processes for Stochastic Modeling*. Stochastic Modeling Series. Chapman & Hall/CRC, 1997.

[18] H.-P. Kriegel, P. Kröger, and I. Gotlibovich. Incremental OPTICS: Efficient computation of updates in a hierarchical cluster ordering. In *Data Warehousing and Knowledge Discovery*, volume 2737 of *Lecture Notes in Computer Science*, pages 224–233. Springer, 2003.

[19] E. Parzen. *Stochastic Processes*. Society for Industrial Mathematics, 1999.

[20] F. Provost and T. Fawcett. Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. In D. Heckerman, H. Mannila, and D. Pregibon, editors, *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 43–48, Newport Beach, CA, August 1997. AAAI Press.

[21] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu. Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications. *Data Minining and Knowledge Discovery*, 2(2):169–194, 1998.

[22] M. Spiliopoulou, I. Ntoutsi, Y. Theodoridis, and R. Schult. MONIC: Modeling and monitoring cluster transitions. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA*, pages 706–711, 2006.

[23] 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)*, pages 638–642, Dec. 2006.

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

[25] R. S. Tsay, D. Pea, and A. E. Pankratz. Outliers in multivariate time series. *Biometrika*, 87(4):789–804, 2000.

[26] L. Tu and Y. Chen. Stream data clustering based on grid density and attraction. *ACM Transactions on Knowledge Discovery from Data*, 3(3):1–27, 2009.

[27] 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*, pages 605–615. Springer-Verlag, Berlin, Heidelberg, 2007.

[28] 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*, 3(3):1–28, 2009.

[29] T. Warren Liao. Clustering of time series data–a survey. *Pattern Recognition*, 38(11):1857–1874, November 2005.

[30] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 103–114. ACM, 1996.