To appear in Data & Knowledge Engineering

Online Clustering of Parallel Data Streams

Jürgen Beringer and Eyke Hüllermeier

Fakultät für Informatik
Otto-von-Guericke-Universität
Magdeburg, Germany
{*juergen.beringer,eyke.huellermeier*}*@iti.cs.uni-magdeburg.de*

Abstract

In recent years, the management and processing of so-called data streams has become a topic of active research in several fields of computer science such as, e.g., distributed systems, database systems, and data mining. A data stream can roughly be thought of as a transient, continuously increasing sequence of time-stamped data. In this paper, we consider the problem of clustering parallel streams of real-valued data, that is to say, continuously evolving time series. In other words, we are interested in grouping data streams the evolution over time of which is similar in a specific sense. In order to maintain an up-to-date clustering structure, it is necessary to analyze the incoming data in an online manner, tolerating not more than a constant time delay. For this purpose, we develop an efficient online version of the classical K-means clustering algorithm. Our method's efficiency is mainly due to a scalable online transformation of the original data which allows for a fast computation of approximate distances between streams.

Key words: data mining, clustering, data streams, fuzzy sets

1 Introduction

In recent years, so-called *data streams* have attracted considerable attention in different fields of computer science such as, e.g., database systems, data mining, or distributed systems. As the notion suggests, a data stream can roughly be thought of as an ordered sequence of data items, where the input arrives more or less continuously as time progresses [19,15,7]. There are various applications in which streams of this type are produced such as, e.g., network monitoring, telecommunication systems, customer click streams, stock markets, or any type of multi-sensor system.

A data stream system may constantly produce huge amounts of data. To illustrate, imagine a multi-sensor system with 10,000 sensors each of which sends a measurement every second of time. Regarding aspects of data storage, management and processing, the continuous arrival of data items in multiple, rapid, time-varying, and potentially unbounded streams raises new challenges and research problems. Indeed, it is usually not feasible to simply store the arriving data in a traditional database management system in order to perform operations on that data later on. Rather, stream data must generally be processed in an online manner in order to guarantee that results are up-to-date and that queries can be answered with small time delay. The development of corresponding *stream processing systems* is a topic of active research [4].

In this paper, we consider the problem of clustering data streams. Clustering is one of the most important and frequently used data analysis techniques. It refers to the grouping of objects into homogeneous classes or groups and is commonly seen as a tool for discovering structure in data. In our context, the goal is to maintain classes of data streams such that streams within one class are similar to each other in a sense to be specified below. Roughly speaking, we assume a large number of evolving data streams to be given, and we are looking for groups of data streams that evolve similarly over time. Our focus is on time-series data streams, which means that individual data items are real numbers that can be thought of as a kind of measurement. There are numerous applications for this type of data analysis such as e.g. clustering of stock rates.

Apart from its practical relevance, this problem is also interesting from a methodological point of view. Especially, the aspect of efficiency plays an important role: Firstly, data streams are complex, extremely high-dimensional objects making the computation of similarity measures costly. Secondly, clustering algorithms for data streams should be adaptive in the sense that up-to-date clusters are offered at any time, taking new data items into consideration as soon as they arrive. In this paper, we develop techniques for clustering data streams that meet these requirements. More specifically, we develop an efficient online version of the classical K-means clustering algorithm. The efficiency of our approach is mainly due to a scalable online transformation of the original data which allows for a fast computation of approximate distances between streams.

The remainder of the paper is organized as follows: Section 2 provides some background information, both on data streams and on clustering. The maintenance and adequate preprocessing of data streams is addressed in Section 3. Section 4 is given to the clustering of data streams and introduces an online version of the K-means algorithm. A fuzzy extension of this approach is motivated and outlined in Section 5. In Section 6, some implementation issues are discussed. Finally, experimental results are presented in Section 7.

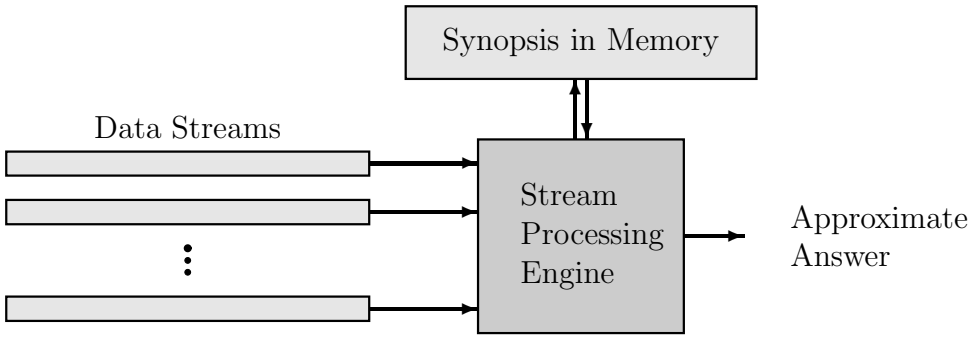

Fig. 1. Basic structure of a data stream model.

2 Background

2.1 The Data Stream Model

The *data stream model* assumes that input data are not available for random access from disk or memory, such as relations in standard relational databases, but rather arrive in the form of one or more continuous data streams. The stream model differs from the standard relational model in the following ways [1]:

- The elements of a stream arrive incrementally in an "online" manner. That is, the stream is "active" in the sense that the incoming items trigger operations on the data rather than being sent on request.
- The order in which elements of a stream arrive are not under the control of the system.
- Data streams are potentially of unbounded size.
- Data stream elements that have been processed are either discarded or archived. They cannot be retrieved easily unless being stored in memory, which is typically small relative to the size of the stream. (Stored/condensed information about past data is often referred to as a *synopsis*, see Fig. 1).
- Due to limited resources (memory) and strict time constraints, the computation of exact results will usually not be possible. Therefore, the processing of stream data does commonly produce *approximate* results [5].

2.2 Clustering

Clustering refers to the process of grouping a collection of objects into classes or "clusters" such that objects within the same class are *similar* in a certain sense, and objects from different classes are dissimilar. In addition, the goal is sometimes to arrange the clusters into a natural hierarchy (hierarchical

clustering). Also, cluster analysis can be used as a form of descriptive statistics, showing whether or not the data consists of a set of distinct subgroups.

Clustering algorithms proceed from given information about the similarity between objects e.g. in the form of a *proximity matrix*. Usually, objects are described in terms of a set of measurements from which similarity degrees between pairs of objects are derived, using a kind of similarity or distance measure. There are basically three types of clustering algorithms (see e.g. [23]): *Mixture modeling* assumes an underlying probabilistic model, namely that the data were generated by a probability density function, which is a mixture of component density functions. *Combinatorial algorithms* do not assume such a model. Instead, they proceed from an objective function to be maximized and approach the problem of clustering as one of combinatorial optimization. The third type of algorithms, so called *mode-seekers*, are somewhat similar to mixture models. However, they take a non-parametric perspective and try to estimate modes of the component density functions directly. Clusters are then formed by looking at the closeness of the objects to these modes that serve as cluster centers.

One of the most popular clustering methods, the so-called K-means algorithm [22], belongs to the latter class. This algorithm starts by guessing K cluster centers and then iterates the following steps until convergence is achieved:

- clusters are built by assigning each element to the closest cluster center;
- each cluster center is replaced by the mean of the elements belonging to that cluster.

K-means usually assumes that objects are described in terms of quantitative attributes, i.e. that an object is a vector $x \in \Re^n$. Dissimilarity between objects is defined by the Euclidean distance, and the above procedure actually implements an iterative descent method that seeks to minimize the variance measure ("within cluster" point scatter)

$$\sum_{k=1}^{K} \sum_{x_i, x_j \in C_k} \|x_i - x_j\|^2, \tag{1}$$

where C_k is the k-th cluster. In each iteration, the criterion (1) is indeed improved, which means that convergence is assured. Still, it is not guaranteed that the global minimum will be found, i.e., the final result may represent a suboptimal local minimum of (1).

2.3 Related Work

Stream data mining [12] is a topic of active research, and several adaptations of standard statistical and data analysis methods to data streams or related models have been developed recently (e.g. [8,33]). Likewise, several online data mining methods have been proposed (e.g. [25,9,30,6,17,14]). In particular, the problem of clustering in connection with data streams has been considered in [20,11,29]. In these works, however, the problem is to cluster the elements of one individual data stream, which is clearly different from our problem, where the objects to be clustered are the streams themselves rather than single data items thereof. To the best of our knowledge, the problem in this form has not been addressed in the literature so far.

There is a bunch of work on time series data mining in general and on clustering time series in particular [26]. Even though time series data mining is of course related to stream data mining, one should not overlook important differences between these fields. Particularly, time series are still static objects that can be analyzed offline, whereas the focus in the context of data streams is on dynamic adaptation and online data mining.

3 Preprocessing and Maintaining Data Streams

The first question in connection with the clustering of (active) data streams concerns the concept of distance or, alternatively, similarity between streams. What does similarity of two streams mean, and why should they, hence, fall into one cluster?

Here, we are first of all interested in the qualitative, time-dependent evolution of a data stream. That is to say, two streams are considered similar if their evolution over time shows similar characteristics. As an example consider two stock rates both of which continuously increased between 9:00 a.m. and 10:30 a.m. but then started to decrease until 11:30 a.m.

To capture this type of similarity, we shall simply derive the Euclidean distance between the normalization of two streams (a more precise definition follows below). This measure satisfies our demands since it is closely related to the (statistical) *correlation* between these streams. In fact, there is a simple linear relationship between the correlation of normalized time series (with mean 0 and variance 1) and their (squared) Euclidean distance. There are of course other reasonable measures of similarity for data streams or, more specifically, time series [21], but Euclidean distance has desirable properties and is commonly used in applications.

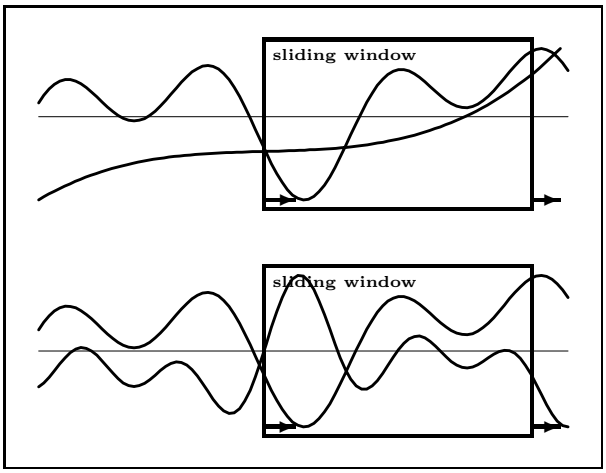

Fig. 2. Data streams are compared within a sliding window of fixed size. In the example above, the behavior of the two streams is obviously quite different. In the example below, the two streams are similar to some extent.

3.1 Data Streams and Sliding Windows

The above example (clustering of stock rates) already suggests that one will usually not be interested in the entire data streams, which are potentially of unbounded length. Instead, it is reasonable to assume that recent observations are more important than past data. Therefore, one often concentrates on a *time window*, that is a subsequence of a complete data stream. The most common type of window is a so-called *sliding window* that is of fixed length and comprises the w most recent observations (cf. Fig.2). A more general approach to taking the relevancy of observations into account is that of *weighing*. Here, the idea is to associate a weight in the form of a real number to each observation such that more recent observations receive higher weights.

When considering data streams in a sliding window of length w, a stream (resp. the relevant part thereof) can formally be written as a w-dimensional vector $X = (x_0, x_1, \ldots, x_{w-1})$, where a single observation x_i is simply a real number. As shown in Fig. 3, we further partition a window into m blocks (basic windows) of size v, which means that $w = m \cdot v$ (Table 1 provides a summary of notation): [1]

$$X = (\underbrace{x_0, x_1, \ldots, x_{v-1}}_{B_1} \mid \underbrace{x_v, x_{v+1}, \ldots, x_{2v-1}}_{B_2} \mid \ldots \mid \underbrace{x_{(m-1)v}, x_{(m-1)v+1}, \ldots, x_{w-1}}_{B_m})$$

Data streams will then be updated in a "block-wise" manner each time v new items have been observed. This approach gains efficiency since the number of necessary updates is reduced by a factor of v. On the other hand, we tolerate the fact that the clustering structure is not always up-to-date. However, since

[1] Typical values as used in our experiments later on are $w = 8192$ and $v = 1024$.

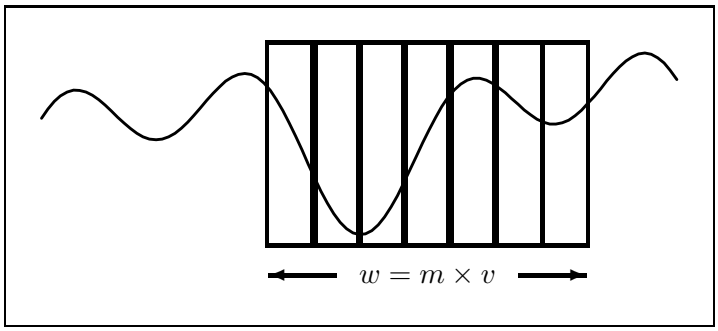

Fig. 3. A window of length w is divided into m blocks of size v.

symbol	meaning
X	data stream
X^n	normalized data stream
x_i	single observation
x_{ij}	$(j+1)$-th element of block B_i
w	window length
v	length of a block
m	number of blocks in a window
c	weighing constant
V	weight vector
$\bar{x}$	mean value of a stream
s	standard deviation of a stream

Table 1
Notation

the delay is at most one block size, this disadvantage is limited at least for small enough blocks. Apart from that, one should note that a small number of observations can change a stream but slightly, hence the clustering structure in the "data stream space" will usually not change abruptly.

We assume data items to arrive synchronously, which means that all streams will be updated simultaneously. An update of the stream X, in this connection also referred to as X^{old}, is then accomplished by the following shift operation:

$$\begin{aligned} X^{old}: &\quad B_1 \mid B_2 \mid B_3 \mid \ldots \mid B_{m-1} \mid B_m \\ X^{new}: &\qquad\quad B_2 \mid B_3 \mid \ldots \mid B_{m-1} \mid B_m \mid B_{m+1} \end{aligned} \tag{2}$$

where B_{m+1} denotes the *entering* block.

Finally, we allow for an exponential weighing of observations (within a win-

dow). The weight attached to observation $x_\imath$ is defined by $c^{w-\imath-1}$, where $0 < c \leq 1$ is a constant. We denote by V the weight vector $(c^{w-1}, c^{w-2} \dots c^0)$ and by $V \odot X$ the coordinate-wise product of V and a stream X:

$$V \odot X \stackrel{\text{df}}{=} (c^{w-1}x_0, c^{w-2}x_1 \dots c^0 x_{w-1}).$$

3.2 Normalization

Since we are interested in the *relative* behavior of a data stream, the original streams have to be normalized in a first step. By normalization one usually means a linear transformation of the original data such that the transformed data has mean 0 and standard deviation 1. The corresponding transformation simply consists of subtracting the original mean and dividing the result by the standard deviation. Thus, we replace each value $x_\imath$ of a stream X by its normalization

$$x_\imath^n \stackrel{\text{df}}{=} \frac{x_\imath - \bar{x}}{s}. \tag{3}$$

Considering the weighing of data streams, $\bar{x}$ and s become the weighted average and standard deviation, respectively:

$$\begin{aligned}
\bar{x} &= \frac{1-c}{1-c^w} \cdot \sum_{\imath=0}^{w-1} x_\imath \cdot c^{w-\imath-1}, \\
s^2 &= \frac{1-c}{1-c^w} \cdot \sum_{\imath=0}^{w-1} (x_\imath - \bar{x})^2 \cdot c^{w-\imath-1} \\
&= \frac{1-c}{1-c^w} \cdot \sum_{\imath=0}^{w-1} (x_\imath)^2 \cdot c^{w-\imath-1} - (\bar{x})^2.
\end{aligned}$$

As suggested above, $\bar{x}$ and s^2 are updated in a block-wise manner. Let X be a stream and denote by $x_{\imath\jmath}$ the $(\jmath+1)$-th element of the $\imath$-th block $B_\imath$. Particularly, the exiting block leaving the current window ("to the left") is given by the first block $B_1 = (x_{10}, x_{11}, \dots, x_{1,v-1})$. Moreover, the new block entering the window ("from the right") is $B_{m+1} = (x_{m+1,0}, x_{m+1,1}, \dots, x_{m+1,v-1})$. We maintain the following quantities for the stream X:

$$Q_1 \stackrel{\text{df}}{=} \sum_{\imath=0}^{w-1} x_\imath \cdot c^{w-\imath-1}, \quad Q_2 \stackrel{\text{df}}{=} \sum_{\imath=0}^{w-1} (x_\imath)^2 \cdot c^{w-\imath-1}.$$

Likewise, we maintain for each block B_k the variables

$$Q_1^k \stackrel{\text{df}}{=} \sum_{\imath=0}^{v-1} x_{k\imath} \cdot c^{v-\imath-1}, \quad Q_2^k \stackrel{\text{df}}{=} \sum_{\imath=0}^{v-1} (x_{k\imath})^2 \cdot c^{v-\imath-1}.$$

An update via shifting the current window by one block is then accomplished by setting

$$\begin{aligned} Q_1 &\leftarrow Q_1 \cdot c^v - Q_1^1 \cdot c^w + Q_1^{m+1}, \\ Q_2 &\leftarrow Q_2 \cdot c^v - Q_1^2 \cdot c^w + Q_2^{m+1}. \end{aligned}$$

3.3 Discrete Fourier Transform

Another preprocessing step replaces the original data by its Discrete Fourier Transform (DFT). As will be explained in more detail in section 3.5, this provides a suitable basis for an efficient approximation of the distance between data streams and, moreover, allows for the elimination of noise. The DFT of a sequence $X = (x_0, \ldots, x_{w-1})$ is defined by the DFT coefficients

$$\mathrm{DFT}_f(X) \stackrel{\mathrm{df}}{=} \frac{1}{\sqrt{w}} \sum_{\jmath=0}^{w-1} x_\jmath \cdot \exp\left(\frac{-i2\pi f \jmath}{w}\right), \qquad f = 0, 1, \ldots, w-1,$$

where $i = \sqrt{-1}$ is the imaginary unit.

Denote by X^n the normalized stream X defined through values (3). Moreover, denote by V the weight vector $(c^{w-1}, \ldots, c^0)$ and recall that $V \odot X^n$ is the coordinate-wise product of V and X^n, i.e. the sequence of values

$$c^{w-\imath-1} \cdot \frac{x_\imath - \bar{x}}{s}.$$

Since the DFT is a linear transformation, which means that

$$\mathrm{DFT}(\alpha X + \beta Y) = \alpha\,\mathrm{DFT}(X) + \beta\,\mathrm{DFT}(Y)$$

for all $\alpha, \beta \geq 0$ and sequences X, Y, the DFT of $V \odot X^n$ is given by

$$\begin{aligned} \mathrm{DFT}(V \odot X^n) &= \mathrm{DFT}\left(V \odot \frac{X - \bar{x}}{s}\right) \\ &= \mathrm{DFT}\left(\frac{V \odot X - V \odot \bar{x}}{s}\right) \\ &= \frac{\mathrm{DFT}(V \odot X) - \mathrm{DFT}(V) \odot \bar{x}}{s}. \end{aligned}$$

Since $\mathrm{DFT}(V)$ can be computed in a preprocessing step, an incremental derivation is only necessary for $\mathrm{DFT}(V \odot X)$.

3.4 Computation of DFT Coefficients

Recall that the exiting and entering blocks are $B_1 = (x_{10}, x_{11}, \ldots, x_{1,v-1})$ and $B_{m+1} = (x_{m+1,0}, x_{m+1,1}, \ldots, x_{m+1,v-1})$, respectively. Denote by $X^{old} = B_1|B_2|\ldots|B_m$ the current and by $X^{new} = B_2|B_3|\ldots|B_{m+1}$ the new data stream. Without taking weights into account, the DFT coefficients are updated as follows [33]:

$$\begin{aligned}\mathrm{DFT}_f \leftarrow\ & e^{\frac{i2\pi fv}{w}} \cdot \mathrm{DFT}_f \\ & + \frac{1}{\sqrt{w}} \left(\sum_{\jmath=0}^{v-1} e^{\frac{i2\pi f(v-\jmath)}{w}} x_{m+1,\jmath} - \sum_{\jmath=0}^{v-1} e^{\frac{i2\pi f(v-\jmath)}{w}} x_{1,\jmath} \right).\end{aligned}$$

In connection with our weighing scheme, the weight of each element of the stream must be adapted as well. More specifically, the weight of each element $x_{\imath\jmath}$ of X^{old} is multiplied by c^v, and the weights of the new elements $x_{m+1,\jmath}$, coming from block B_{m+1}, are given by $c^{v-\jmath-1}$, $0 \le \jmath \le v-1$. Noting that $\mathrm{DFT}_f(c^v \cdot X^{old}) = c^v\,\mathrm{DFT}_f(X^{old})$ due to the linearity of the DFT, the DFT coefficients are now modified as follows:

$$\begin{aligned}\mathrm{DFT}_f \leftarrow\ & e^{\frac{i2\pi fv}{w}} \cdot c^v\,\mathrm{DFT}_f \\ & + \frac{1}{\sqrt{w}} \left(\sum_{\jmath=0}^{v-1} e^{\frac{i2\pi f(v-\jmath)}{w}} c^{v-\jmath-1} x_{m+1,\jmath} - \sum_{\jmath=0}^{v-1} e^{\frac{i2\pi f(v-\jmath)}{w}} c^{w+v-\jmath-1} x_{1,\jmath} \right).\end{aligned}$$

Using the values

$$\beta_k^f \stackrel{\mathrm{df}}{=} \sum_{\jmath=0}^{v-1} e^{\frac{i2\pi f(v-\jmath)}{w}} c^{v-\jmath-1} x_{k,\jmath}, \tag{4}$$

the above update rule simply becomes

$$\mathrm{DFT}_f \leftarrow e^{\frac{i2\pi fv}{w}} \cdot c^v\,\mathrm{DFT}_f + \frac{1}{\sqrt{w}} \left(\beta_{m+1}^f - c^w \beta_1^f \right).$$

As can be seen, the processing of one block basically comes down to maintaining the Q- and β-coefficients. The time complexity of the above update procedure is therefore $O(nvu)$. Moreover, the procedure needs space $O(nmu + nv)$: For each stream, the β-coefficients have to be stored for each block plus the complete last block.

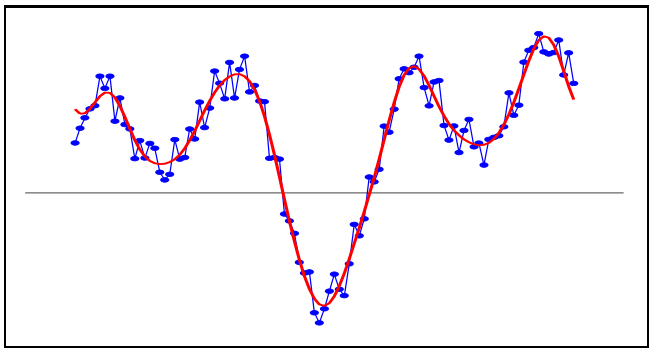

Fig. 4. Original (noisy) signal and DFT-filtered curve.

3.5 Distance Approximation and Smoothing

We now turn to the problem of computing the Euclidean distance

$$\|X - Y\| = \left(\sum_{i=0}^{w-1} (x_i - y_i)^2 \right)^{1/2}$$

between two streams X and Y (resp. the distance $\|V \odot X^n - V \odot Y^n\|$ between their normalized and weighted versions) in an efficient way. A useful property of the DFT is the fact that it preserves Euclidean distance, i.e.

$$\|X - Y\| = \| \operatorname{DFT}(X) - \operatorname{DFT}(Y) \| . \tag{5}$$

Furthermore, the most important information is contained in the first DFT coefficients. In fact, using only these coefficients within the inverse transformation (which recovers the original signal X from its transform $\operatorname{DFT}(X)$) comes down to implementing a low-pass filter and, hence, to using DFT as a smoothing technique (see Fig. 4 for an illustration).

Therefore, a reasonable idea is to approximate the distance (5) by using only the first $u \ll w$ rather than all of the DFT coefficients and, hence, to store the values (4) only for $f = 0, \ldots, u-1$. More specifically, since the middle DFT coefficients are usually close to 0, the value

$$\left(2 \sum_{f=1}^{u-1} (\operatorname{DFT}_f(X) - \operatorname{DFT}_f(Y))(\overline{\operatorname{DFT}_f(X) - \operatorname{DFT}_f(Y)}) \right)^{1/2}$$

is a good approximation to (5). Here, we have used that $\operatorname{DFT}_{w-f+1} = \overline{\operatorname{DFT}}_f$, where $\overline{\operatorname{DFT}}_f$ is the complex conjugate of DFT_f. Moreover, the first coefficient DFT_0 can be dropped, as for real-valued sequences the first DFT coefficient is given by the mean of that sequence, which vanishes in our case (recall that we normalize streams in a first step).

The above approximation has two advantages. First, by filtering noise we cap-

```
1. Initialize K cluster centers at random
2. Repeat
3.     Assign each stream to the closest center
4.     Replace each center by the center of its associated cluster
5.     If a new block is complete:
6.         Update the streams and pairwise distances
7.         Update the optimal cluster number K
```

Fig. 5. Incremental version of the K-means algorithm for clustering data streams.

ture only those properties of a stream that are important for its characteristic time-dependent behavior. Second, the computation of the distance between two streams becomes much more efficient due to the related dimensionality reduction.

4 Clustering Data Streams

The previous section has presented an efficient method for computing the (approximate) pair-wise Euclidean distances between data streams in an incremental way. On the basis of these distances, it is principally possible to apply any clustering method. In this section, we propose an incremental version of the well-known K-means algorithm. K-means appears especially suitable in our context, as it is an iterative procedure that can be extended to the online setting in a natural way.

Our incremental K-means method works as shown in Fig. 5: The standard K-means algorithm is run on the current data streams. As soon as a new block is available for all streams, the current streams are updated by the shift operation (2). Standard K-means is then simply continued. In other words, the clustering structure of the current streams is taken as an initialization for the clustering structure of the new streams. This initialization will usually be good or even optimal since the new streams will differ from the current ones but slightly (see Fig. 6 for an illustration).

An important additional feature that distinguishes our algorithm from standard K-means is an incremental adaptation of the cluster number K. Needless to say, such an adaptation is very important in the context of our application where the clustering structure can change over time. Choosing the right number K is a question of practical importance in standard K-means also, and a number of (heuristic) strategies has been proposed. A common approach is to look at the cluster dissimilarity (the sum of distances between objects and their associated cluster centers) for a set of candidate values $K \in \{1, 2, \ldots, K_{max}\}$. The cluster dissimilarity is obviously a decreasing function of K, and one ex-

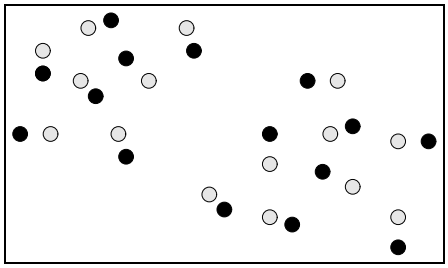

Fig. 6. A slight change of the data streams, here indicated as points, will usually not change the clustering structure drastically. In this example, the clusters before (light points) and after the modification (dark points) are even identical.

pects that this function will show a kind of irregularity at K^*, the optimal number of clusters: The benefit of increasing the cluster number K will usually be large if $K < K^*$ but will be comparatively small otherwise. This intuition has been formalized, e.g. by the recently proposed *gap statistic* [31].

Unfortunately, the above strategy is not practicable in our case as it requires the consideration of too large a number of candidate values. Our adaptation process rather works as follows: In each iteration phase, one test is made in order to check whether the clustering structure can be improved by increasing or decreasing K^*, the current (hopefully optimal) cluster number. By iteration phase we mean the phase between the entry of new blocks, i.e. while the streams to be clustered remain unchanged.

We restrict ourselves to adaptations of K^* by ± 1, which is again justified by the fact that the clustering structure will usually not change abruptly. In order to evaluate a clustering structure, we make use of a standard quality measures, especially the well-known *separation index* [32]. Good results have also been obtained with a measure that has recently been proposed in [28].

Let $Q(K)$ denote the above quality measure for the cluster number K resp. the clustering obtained for this number. The optimal cluster number is then updated as follows:

$$K^* \leftarrow \arg\max\{Q(K^*-1), Q(K^*), Q(K^*+1)\}.$$

Intuitively, going from K^* to K^*-1 means that one of the current clusters has disappeared, e.g. since the streams in this cluster have become very similar to the streams in a neighbored cluster. Thus, $Q(K^*-1)$ is derived as follows: One of the current candidate clusters is tentatively removed, which means that each of its elements is re-assigned to the closest cluster (center) among the remaining ones (note that different elements might be assigned to different clusters). The quality of the clustering structure thus obtained in then computed. This is repeated K times, i.e., each of the current clusters is removed by way of trial. The best clustering structure is then chosen, i.e., $Q(K^*-1)$

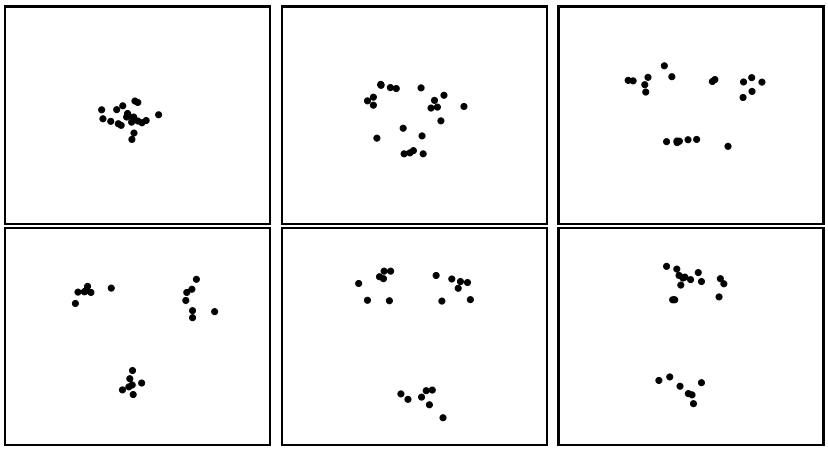

Fig. 7. Snapshots of a dynamic clustering structure at different time points (upper left to lower right).

is defined by the quality of the best structure.

Going from K^* to $K^* + 1$ assumes that an additional cluster has emerged, e.g. since a homogeneous cluster of streams has separated into two groups. To create this cluster we complement the existing K^* centers by one center that is defined by a randomly chosen object (stream). The probability of a stream to be selected is reasonably defined as an increasing function of the stream's distance from its cluster center. In order to compute $Q(K^* + 1)$, we try out a fixed number of randomly chosen objects and select the one that gives the best clustering.

In order to process one block, our online clustering procedure needs time $O(nuK^2)$; the complexity is quadratic in K since, in connection with the adaptation of the number of clusters, each of the current clusters is tentatively removed. The space complexity of the procedure is $O(nuK)$, since the distances have to be maintained for each stream and cluster.

5 Fuzzy Clustering

Fuzzy (K-means) clustering is a generalization of standard (K-means) clustering that has proved to be useful in many practical applications. In standard clustering, each object is assigned to one cluster in an unequivocal way. As opposed to this, in fuzzy clustering an object x may belong to different clusters at the same time, and the degree to which it belongs to the $\imath$-th cluster is expressed in terms of a *membership degree* $\mu_\imath(x)$. Consequently, the boundary of single clusters and the transition between different clusters are usually "smooth" rather than abrupt.

This aspect is of special importance in the context of our application. In fact,

since data streams are evolving objects in a very high-dimensional space, the clustering structure can change over time, and it usually does so in a smooth way. To illustrate, suppose that the objects to be clustered are continuously moving points in a two-dimensional space.[2] Fig. 7 shows snapshots of a dynamic clustering structure at different time points. At the beginning, there is only one big cluster. However, this cluster begins to divide itself into three small clusters, two of which are then again combined into one cluster. Thus, there are time points where the structure definitely consists of one (first picture), two (sixth picture), and three (fourth picture) clusters. In-between, however, there are intermediate states, for which it is not always possible to determine the number of clusters or to assign an object to one cluster in an unequivocal way.

The fuzzy variant of K-means clustering seeks to minimize the following objective function [3]:

$$\sum_{i=1}^{n}\sum_{j=1}^{K} \|x_i - c_j\| \, (\mu_{ij})^m, \tag{6}$$

where $\mu_{ij} = \mu_j(x_i)$ is the membership of the i-th object x_i in the j-th cluster, and c_i is the j-th center. In the commonly employed *probabilistic* version of fuzzy K-means, it is assumed that

$$\sum_{j=1}^{K} \mu_{ij} = \sum_{j=1}^{K} \mu_j(x_i) = 1 \tag{7}$$

for all x_i [24]. As the name suggests, the membership degrees might then also be interpreted as probability degrees.

The constant $m > 1$ in (6) is called the *fuzzifier* and controls the overlap ("smoothness") of the clusters (a common choice is $m = 2$). To realize the effect of the fuzzifier, consider first the (non-fuzzy) case $m = 1$. Since each distance $d_{ij} = \|x_i - c_j\|$ is weighted by the membership μ_{ij}, minimizing (6) while satisfying (7) is obviously achieved by assigning a membership of 1 to the cluster (center) having minimal distance d_{ij} and 0 to all other clusters. Now, raising membership degrees to the power $m > 1$ comes down to decreasing the weights of the distances d_{ij}, and the smaller a membership degree μ_{ij}, the stronger the corresponding decrease. Thus, it might then be reasonable to shift some of the unit mass in (7) from high membership degrees to lower ones. Consequently, the higher m is, the more "fuzzy" the clustering structure will become. To give a concrete example, let $d_{i1} = 1/4$ and $d_{i2} = 3/4$. For

[2] We note that, even though these objects might in principle be thought of as data streams over an extremely short window ($w = 2$), the movement of the points in this example is not typical of data streams.

$m = 1$, the optimal solution is to assign x_i to the first cluster, i.e., $\mu_{i1} = 1$ and $\mu_{i2} = 0$. For $m = 2$, the optimal distribution is $\mu_{i1} = 3/4$ and $\mu_{i2} = 1/4$.

Minimizing (6) subject to (7) defines a constrained optimization problem.[3] The clustering algorithm approximates an optimal (or at least locally optimal) solution by means of an iterative scheme that alternates between recomputing the optimal centers according to

$$c_j = \frac{\sum_{i=1}^{n} x_i \, (\mu_{ij})^m}{\sum_{i=1}^{n} (\mu_{ij})^m}$$

and membership degrees according to

$$\mu_{ij} = \left(\sum_{\ell=1}^{K} \left(\frac{\|x_i - c_j\|}{\|x_i - c_\ell\|} \right)^{2/(m-1)} \right)^{-1}.$$

In connection with the adaptive determination of the cluster number K, a validity measure for evaluating a clustering structure is needed. Several such measures have been proposed for the fuzzy case. We obtained the best experimental results with the fuzzy-variant of the separation index, which is defined as

$$\frac{\sum_{i=1}^{n} \sum_{j=1}^{K} \|x_i - c_j\| \, \mu_{ij}^m}{n \cdot \min_{p,q} \|c_p - c_q\|}. \tag{8}$$

As most validity measures do, (8) puts the inter-cluster variability (numerator) in relation to the intra-cluster variability (denominator). In this case, the latter is simply determined by the minimal distance between two cluster centers. Obviously, the smaller the separation index, the better the clustering structure.

In our experiments with fuzzy clustering, we observed the following undesirable effect: If a single data stream changes its characteristic behavior, it usually moves from one cluster to another one. While doing so, the stream itself is often identified as a separate cluster. This is caused by the fact that, roughly speaking, the validity measure (8) prefers small but homogeneous clusters to large but diverse ones.

In order to avoid this problem, we have used two modifications of the standard fuzzy K-means approach. Since these modifications are more or less known, we shall not describe them in detail here but restrict ourselves to outlining the basic ideas: In *possibilistic* K-means clustering, as suggested by Krishnapuram and Keller [27], the constraint (7) is weakened by replacing the equality with an inequality: $\sum_{i=1}^{K} \mu_i(x) \leq 1$. Thus, it is possible that an object (data stream)

[3] As most clustering problems, the standard K-means problem is known to be NP-hard (see e.g. [13]).

Fig. 8. First online visualization tool.

has a low degree of membership in all clusters. For instance, a stream moving from one cluster to an other one might be assigned low membership degrees in both clusters.

The second approach considers a special "noise cluster", to which isolated objects (outliers) can be assigned [10]. We found that the performance of this method could further be improved by punishing very small clusters. This was accomplished as follows: For all clusters having a cardinality of less than 2 in terms of the sum of membership degrees, m, the separation index was divided by a factor $m/2$.

6 Implementation

Our online clustering method has been implemented under XXL (eXtensible and fleXible Library). The latter is a JAVA library for query processing developed and maintained at the Informatics Institute of Marburg University [2], and meanwhile utilized by numerous (international) research groups. Apart from several index structures, this library offers built-in operators for creating, combining and processing queries.

As the name suggests, the basic philosophy of XXL is its flexibility and the possibility to easily extend the library. Currently, a package called PIPES is developed for the handling of active data streams. This package offers abstract classes for the implementation of data sources and operators, which can be combined into so-called *query graphs* in a flexible way.

We have used PIPES for our implementation. Roughly speaking, the process of clustering corresponds to an operator. As input this operator receives (blocks of) data items from the (active) data streams. As output it produces the current cluster number for each stream. This output can be used by any other operator, e.g. by visualization tools that we have developed as well.

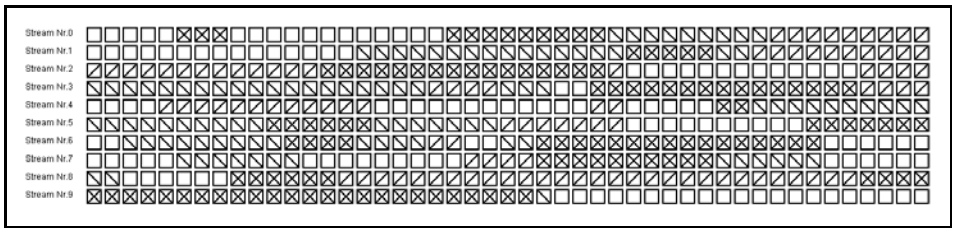

Fig. 9. Second online visualization tool.

Fig. 8 shows an example of an online visualization. Basically, this tool shows a table the rows of which correspond to the data streams. What can be seen here is the top of a list consisting of 230 stock rates whose overall length is 230. Each column stands for one cluster, and a stream's current cluster is indicated by a square. Thus, a square moves horizontally each time the clustering structure changes, making this tool very convenient for visual inspection. Further, each cluster is characterized by means of a *prototypical stream* showing the qualitative behavior of the streams in that cluster. As can be seen, in our example there are currently three types of stock rates showing a similar evolution over the last 16 hours, the size of the sliding window. The first type, for instance, decreased slightly during the first half of that time period and then began to increase until the end of the observed period of time.

A second visualization tool, illustrated in Fig. 9, shows the recent history of the clustering structure, rather than only the current state. Again, each row corresponds to a stream. Now, however, each column corresponds to a time point. Each cluster is identified by a color (here a symbol for better visualization), and these colors are used to identify a stream's cluster at the different time points.

7 Experimental Validation

A convincing experimental validation of our approach is difficult for at least three reasons. Firstly, the evaluation of clustering methods is an intricate problem anyway, since an objectively "correct solution" in the sense of a real clustering structure does usually not exist, at least not in the case of real-world data. [4] Moreover, the performance of a clustering method strongly depends on the choice of the data set (selective superiority problem). In fact, most methods give good results for a particular type of data but otherwise perform poorly. Secondly, since our approach is the first method for clustering complete data streams, there are no alternative methods to compare with. Thirdly, real-world streaming data is currently not available in a form that

[4] Since measuring the quality of results becomes even more intricate in fuzzy clustering, we focused on standard K-means in our experiments.

is suitable for conducting systematic experiments. Therefore, we decided to carry out additional experiments with synthetic data. As an important advantage of synthetic data let us note that it allows for conducting experiments in a *controlled* way and, hence, to answer specific questions concerning the performance of a method and its behavior under particular conditions. Corresponding experiments are presented in Section 7.1. In Section 7.2, a second experimental study using real-world data (stock rates) is presented.

7.1 Synthetic Data

In order to get an impression of the practical performance and applicability of our approach, we have investigated the following two aspects: (a) The efficiency of the method in terms of its runtime. (b) The performance of the method in terms of the quality of clustering structures. In both cases, the aim was to evaluate our online implementation of K-means clustering, including the efficient preprocessing of data streams, rather than the clustering method itself. In fact, K-means is a thoroughly investigated algorithm with known advantages and disadvantages.

Synthetic data was generated in the following way: First, a *prototype* $p(\cdot)$ is generated for each cluster. This prototype is a stochastic process defined by means of a second-order difference equation:

$$\begin{aligned} p(t+\Delta t) &= p(t) + p'(t+\Delta t) \\ p'(t+\Delta t) &= p'(t) + u(t), \end{aligned}$$

$t = 0, \Delta t, 2\Delta t \ldots$. The $u(t)$ are independent random variables, uniformly distributed in an interval $[-a, a]$. Of course, the smaller the constant a is, the smoother the stochastic process $p(\cdot)$ will be. The elements that (should) belong to the cluster are then generated by "distorting" the prototype, both horizontally (by stretching the time axis) and vertically (by adding noise). More precisely, a data stream $x(\cdot)$ is defined by

$$x(t) = p\Big(t + h(t)\Big) + g(t),$$

where $h(\cdot)$ and $g(\cdot)$ are stochastic processes that are generated in the same way as the prototype $p(\cdot)$.[5] Fig. 10 shows a typical prototype together with a distortion $x(\cdot)$.

[5] However, the constant a that determines the smoothness of a process can be different for $p(\cdot), h(\cdot)$, and $g(\cdot)$.

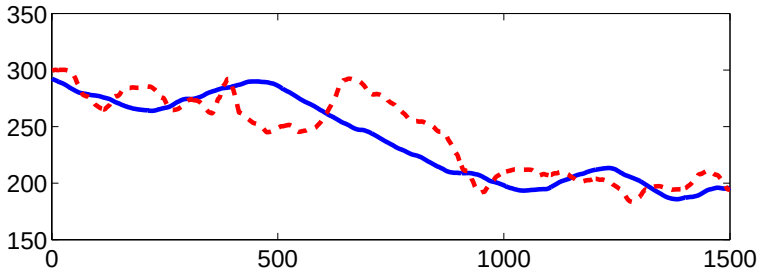

Fig. 10. Example of a prototypical data stream (solid line) and a distorted version (dashed line).

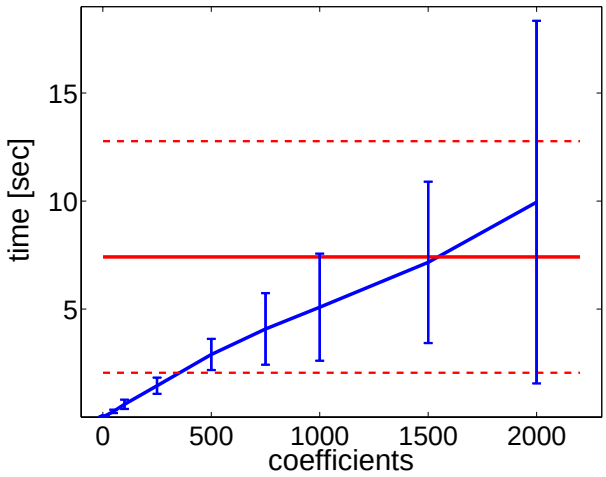

Fig. 11. Mean processing time per block and standard deviation for different numbers of DFT coefficients (increasing curve), average processing time and standard deviation for the original streams (horizontal lines).

7.1.1 Efficiency

We have conducted experiments with 100 data streams, generated as distortions of 6 prototypes. The parameter a was set, respectively, to 0.04, 0.04, and 0.5 for the processes $p(\cdot)$, $h(\cdot)$, and $g(\cdot)$. The window size and block size were set to 8192 and 1024, respectively.

As an efficiency parameter we have measured the time needed in order to process one block, that is to preprocess the data and to update the clustering structure. Fig. 11 shows the mean processing time together with the standard deviation for different numbers of DFT coefficients. Moreover, we have plotted the average processing time and standard deviation for the original streams. As it was to be expected, the time complexity increases as an approximately linear function of the number of DFT coefficients. The critical number of coefficients is around 1500. That is, when using 1500 or more DFT coefficients, the preprocessing of the data will no longer pay off.

As can be seen in Fig. 11, the processing time for one block is ≈ 7 seconds.

Thus, the system can process streams with an arrival rate of ≈ 150 elements per second. When approximating the original streams with 250 DFT coefficients, an arrival rate of ≈ 1000 per second can be handled. As an aside, we note that the memory requirements are not critical in this application, as the number of data streams is still manageable.

7.1.2 Quality

As already noted above, measuring the quality of a clustering structure is an intricate problem. Even in our case, where *artificial clusters* are generated in a controlled way, it is not clear that the clusters thus obtained correspond to the "correct" clustering structure. In fact, if by chance two artificial clusters (resp. their prototypes) become similar enough, they actually define a single *real cluster*. Thus, the relation between the correct number of (real) clusters, K^*, and the number of artificial clusters, K_*, is not an equality but an inequality: $K^* \leq K_*$.[6]

Due to the above reasons, we refrained from computing an absolute measure of the quality of a clustering structure. Rather, we aimed at comparing the clustering structure obtained by our online method, involving the compression of (windowed) data streams, with the structure that would have been obtained by clustering the *original*, uncompressed (windowed) data streams. In other words, we were interested in the quality loss caused by the online preprocessing of the data. Recall that in our approach, distances are computed after having transformed the original data streams from a w-dimensional space to a u-dimensional space, where $u \ll w$. In other words, a clustering structure is derived for *approximations* of the original streams. Therefore, the result might of course differ from the result obtained for the original streams (i.e. for $u = w$), even though intuitively one would expect that a slight perturbation of the original streams will not have a strong effect on the clustering structure.

Here, we face the problem of measuring the dissimilarity or *distance* between two clustering structures. Again, this problem is by no means trivial. A reasonable distance measure could be defined by the proportion of pairs (X, Y) of data streams such that X and Y belong to the same cluster in one of the structures but to different clusters in the other one. A drawback of this measure is the fact that it critically depends on the cluster number. For example, suppose that a single cluster in the first structure is split into two clusters in the second structure. In this case, the above measure will suddenly become very large, even though the two structures might not be regarded as being extremely dissimilar. Moreover, the distance strongly depends on the size of the corresponding cluster, which is also not desirable.

[6] The case where an artificial cluster is divided into two or more real clusters is negligible for large enough windows.

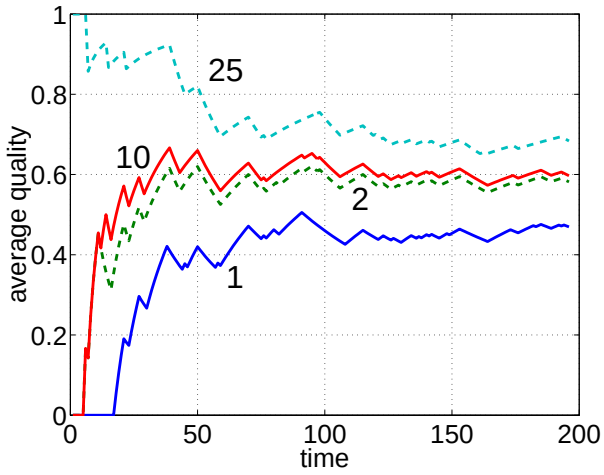

Fig. 12. Average quality of the clustering structure for different numbers of DFT coefficients (1, 2, 10, 25).

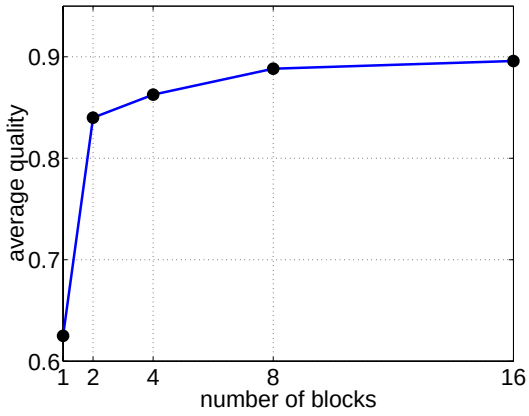

Fig. 13. Average quality of the clustering structure for different numbers of blocks per window.

Due to these difficulties, we decided to employ a rather simple distance measure that only checks whether two structures are completely identical (distance 0) or not (distance 1). The curves in Fig. 12 show the *average quality* for different numbers of DFT coefficients. That is, the value plotted at time point t is the average of the distances at time points $0, \Delta t, 2\Delta t \ldots t$, i.e., the proportion of time points where the online clustering structure is identical to the reference structure (derived offline). As was to be expected, the larger the number of coefficients is, the higher the average quality becomes. In any case, the results are rather satisfying. For example, by using only 25 DFT coefficients, the chance to derive an identical structure is approximately 70%. Again, however, let us point out that the number of coefficients needed to obtain a good approximation to the reference structure will usually depend on the regularity (smoothness) of the streams.

We also investigated the dependence of the clustering quality on the number

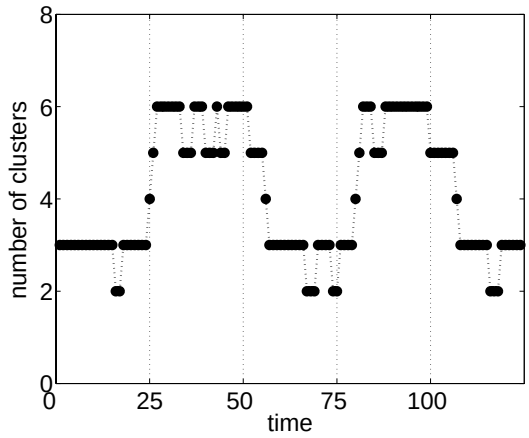

Fig. 14. Changing number of clusters over time.

of blocks per window, using the same quality measure as above (and 50 DFT coefficients). As can be seen in Fig. 13, the quality seems to increase with the number of blocks, an effect which is perhaps not evident in the first instance. It is likely, however, that a more frequent updating of the streams will reduce the approximation error, i.e., the distance between the original stream and its compressed representation, on average. This in turn can explain the positive effect that increasing the number of blocks has on the clustering quality.

Finally, in order to test our scheme for adapting the number of clusters, we varied the number of artificial clusters in the data generating process: Starting with three clusters, the number of clusters was doubled at time points 25 and 75, and again reduced to three at time points 50 and 100. Doubling resp. rejoining a cluster was simply accomplished by temporarily using a completely different data generating process for one half of the streams in that cluster. As can be seen in Fig. 14, the number of clusters is adapted correctly with a relatively small delay. Note that this experiment also supports evidence for our conjecture that the number of real clusters is sometimes smaller than the number of artificial clusters (2 instead of 3 resp. 5 instead of 6 in our example).

7.2 Real-World Data: Stock Rates

In a second experimental study, we applied our online approach to the clustering of stock rates taken from the German HDAX index. During the time period from May 14 to June 1, 2004, the rates of this index were recorded every minute of time. For clustering the 111 data streams thus obtained, a window size of 240 minutes and a block size of 15 minutes were used. The weight parameter was set such that the smallest weight in a block (i.e. the weight of the oldest element) is 0.4.

Fig. 15 shows three exemplary stock rates recorded between 04-05-14, 11:29

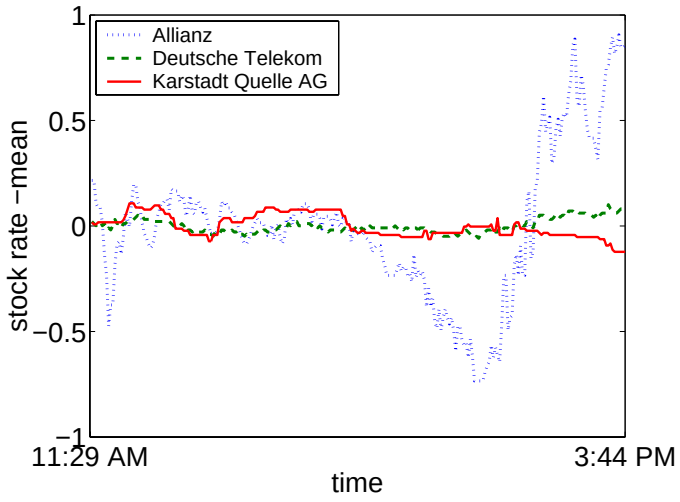

Fig. 15. Three exemplary stock rates (deviation from the mean value) recorded between 04-05-14, 11:29 a.m. and 04-05-14, 3:44 p.m.

a.m. and 04-05-14, 3:44 p.m. To improve the illustration, we didn't plot the rates directly but their deviation from the mean. Even though the stocks of Deutsche Telekom and Karstadt Quelle are closer in absolute value, the qualitative evolution is more similar for Allianz and Deutsche Telekom (dotted and dashed line). In fact, after normalization the distance between the latter is ≈ 5.5 while the distance between the former is ≈ 19.6.

The dynamic grouping of stocks provides valuable information that could be interesting for analysts and stockbrokers. For example, changing trends and tendencies can often be recognized by simply inspecting one of the online visualization tools. In general, however, the evolution of the clustering structure will be the point of departure for further types of data analysis and data mining. That is, the output of our online clustering tool will be the input of other data mining methods.

Just to illustrate this idea, we have used the dynamic clustering structure in order to derive similarity degrees between pairs of stock rates. The similarity between two stocks is simply defined as the proportion of time points for which the two stocks are in the same cluster. In particular, the similarity is 1 (0) if the stocks are always (never) in the same cluster. On the basis of these similarity degrees, we have clustered the stocks, using a standard hierarchical clustering method. A graphical illustration of the result is shown in Fig. 16. The fact that there is no marked structure suggests that our original clustering of data streams is indeed dynamic in the sense that the grouping of stock rates changes in the course of time. This is also confirmed by the statistical distribution of similarity degrees, which is approximately normal with mean ≈ 0.4 and standard deviation ≈ 0.085 (cf. Fig. 17). In other words, extremely similar or dissimilar stocks are rather rare.

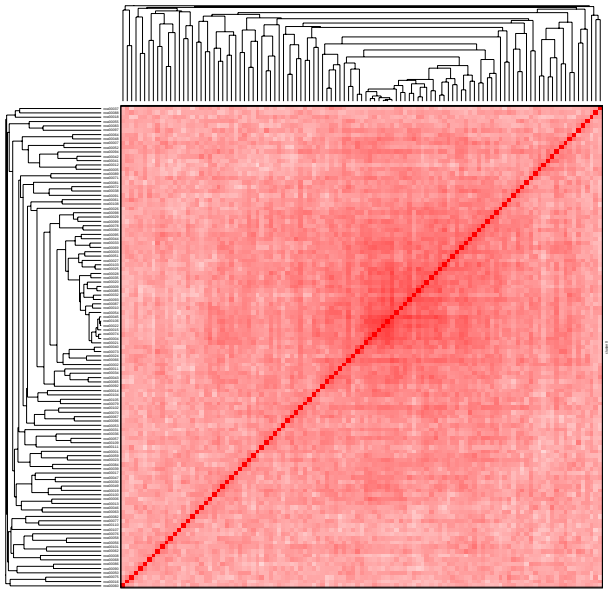

Fig. 16. Visualization of the hierarchical clustering structure (dendogram). Rows and columns correspond to stock rates. The darker a field is, the higher is the corresponding similarity degree.

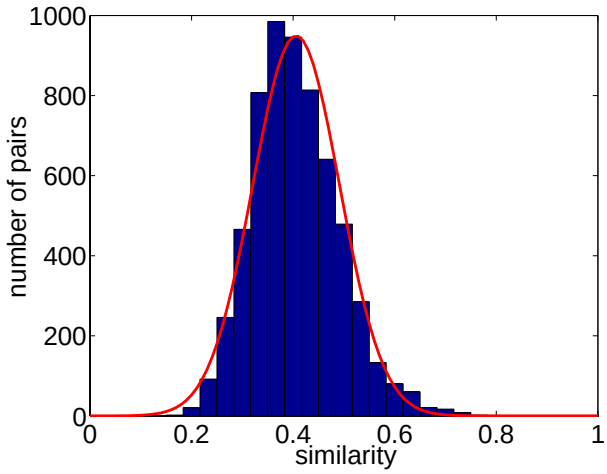

Fig. 17. Distribution of similarity degrees, plotted in the form of a histogram together with the fitted normal distribution (smooth curve).

Some interesting findings have also been obtained by further analyzing the highly similar stock rates. First of all, the small cluster of maximally similar stocks consists of the main index (HDAX) together with a few very large companies such as e.g. VW, Daimler, and BMW. This is hardly astonishing, since these stocks do of course dominate the index. However, there were also pairs of stock rates the high similarity of which was much less obvious and, hence, called for an explanation. This concerns, for example, the stocks of Deutsche Bank and Thyssen Krupp. Interestingly enough, by further investigating this case we found a press release from May 19 (five days af-

ter we started recording the stocks) in which the DEUTSCHE BANK warned against selling THYSSEN KRUPP stocks. Indeed, it turned out that there is a connection between these two stocks (companies), which is a bit involved but makes plausible their similar evolution.

8 Summary

In this paper, we have addressed the problem of clustering data streams in an online manner. To this end, we have developed an adaptable, scalable online version of the K-means algorithm for data streams. A key aspect of our method is an efficient preprocessing step which includes an incremental computation of the distance between data streams, using a DFT approximation of the original data. This way, it becomes possible to cluster thousands of data streams in real-time.

We have also implemented a fuzzy version of online K-means clustering. In fuzzy cluster analysis, an object can belong to more than one cluster, and the degree of membership in each cluster in characterized by means of a number between 0 and 1. This extension of the standard approach appears particularly reasonable in the context of online clustering: If a data stream moves from one cluster to another cluster, it usually does so in a "smooth" rather than an abrupt manner.

In order to investigate the performance and applicability of our approach, we have performed experiments with both synthetic and real-world data. The results of these experiments have shown that our method achieves an extreme gain in efficiency at the cost of an acceptable (often very small) loss in quality. Since we are not aware of other methods for the problem of online clustering of data streams as considered in this paper, it was of course not possible to compare our approach with alternative methods.

Going beyond the relatively simple K-means approach by trying out other clustering methods is a topic of ongoing and future work. In this respect, one might think of extensions of K-means, such as Gath-Geva clustering [16], as well as alternative methods such as e.g. self-organizing maps. Likewise, other techniques might be tested in the preprocessing step of our framework, especially for the online approximation of data streams. For example, several interesting techniques based on wavelet analysis have been proposed recently (e.g. [18]). A first examination of these techniques has shown that they cannot be used directly for our purpose (for example because they maintain approximations of other types of stream summaries, or because they can only deal with finite domains). This does not exclude, however, that they might be adapted in one way or the other.

We conclude the paper by noting that our method does not necessarily produce the end product of a data mining process. Actually, it transforms a set of data streams into a new set of streams the elements of which are cluster memberships. These "cluster streams" can then be analyzed by means of other data mining tools. This point has been nicely illustrated by the experiments with stock rates. Investigating this idea in more detail and developing (online!) methods suitable for analyzing such cluster streams is an important topic of ongoing work.

Acknowledgements: The authors like to express their gratitude for the useful comments and suggestions of three anonymous reviewers.

References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 1–16. ACM, 2002.

[2] J. Bercken, B. Blohsfeld, J. Dittrich, J. Krämer, T. Schäfer, M. Schneider, and B. Seeger. XXL - A library approach to supporting efficient implementations of advanced database queries. In *Proc. of the VLDB*, pages 39–48, 2001.

[3] J.C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithm*. Plenum Press, New York, 1981.

[4] M Cherniack, H Balakrishnan, M Balazinska, D Carney, U Cetintemel, Y Xing, and S Zdonik. Scalable distributed stream processing. In *Proceedings CIDR–03: First Biennial Conference on Innovative Database Systems*, Asilomar, CA, 2003.

[5] J Considine, F Li, G Kollios, and JW Byers. Approximate aggregation techniques for sensor databases. In *ICDE–04: 20th IEEE Int'l Conference on Data Engineering*, 2004.

[6] G Cormode and S Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. In *Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 296–306. ACM Press, 2003.

[7] A. Das, J. Gehrke, and M. Riedewald. Approximate join processing over data streams. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 40–51. ACM Press, 2003.

[8] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *Proc. of the Annual ACM-SIAM Symp. on Discrete Algorithms (SODA 2002)*, 2002.

[9] Mayur Datar and S.Muthukrishnan. Estimating rarity and similarity over data stream windows. In *Algorithms - ESA 2002*, pages 323–334. Springer, 2002.

[10] RN Dave. Characterization and detection of noise in clustering. *Pattern Recognition Letters*, 12(11):657–664, 1991.

[11] P. Domingos and G. Hulten. A general method for scaling up machine learning algorithms and its application to clustering. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 106–113, Williamstown, MA, 2001.

[12] P Domingos and G Hulten. A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics*, 12, 2003.

[13] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay. Clustering large graphs via the singular value decomposition. *Machine Learning*, 56:9–33, 2004.

[14] MM. Gaber, S. Krishnaswamy, and A. Zaslavsky. Cost-efficient mining techniques for data streams. In *Proceedings of the 2nd Workshop on Australasian Information Security, Data Mining and Web Intelligence, and Software Internationalisation*, pages 109–114. Australian Computer Society, Inc., 2004.

[15] M Garofalakis, J Gehrke, and R Rastogi. Querying and mining data streams: you only get one look. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 635–635. ACM Press, 2002.

[16] I. Gath and A. Geva. Unsupervised optimal fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):773–781, 1989.

[17] C. Giannella, J. Han, J. Peia, X. Yan, and PS. Yu. Mining frequent patterns in data streams at multiple time granularities. In H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha, editors, *Next Generation Data Mining*. AAAI/MIT, 2003.

[18] AC. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. *The VLDB Journal*, pages 79–88, 2001.

[19] L Golab and M Tamer. Issues in data stream management. *SIGMOD Rec.*, 32(2):5–14, 2003.

[20] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *IEEE Symposium on Foundations of Computer Science*, pages 359–366, 2000.

[21] D. Gunopulos and G. Das. Time series similarity measures and time series indexing. In *Proc. 2001 ACM SIGMOD International Conference on Management of Data*, Santa Barbara, California, 2001.

[22] JA. Hartigan. *Clustering Algorithms*. Wiley, New York, 1975.

[23] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, 2001.

[24] F. Höppner, F. Klawonn, F. Kruse, and T. Runkler. *Fuzzy Cluster Analysis*. Wiley, Chichester, 1999.

[25] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106. ACM Press, 2001.

[26] E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: A survey and empirical demonstration. In *8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 102–111, Edmonton, Alberta, Canada, July 2002.

[27] R. Krishnapuram and J.M. Keller. A possibilistic approach to clustering. IEEE *Transactions on Fuzzy Systems*, 1(2):98–110, 1993.

[28] I Lerman, J Costa, and H Silva. Validation of very large data sets clustering by means of a nonparametric linear criterion. In *Classification, Clustering and Data Analysis: Recent advances and application, Proc. of IFCS–2002*, pages 147–157. Springer, 2002.

[29] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering, 2002.

[30] S. Papadimitriou, C. Faloutsos, and A. Brockwell. Adaptive, hands-off stream mining. In *29th International Conference on Very Large Data Bases*, 2003.

[31] R. Tibshirami, G. Walther, and T. Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society – Series B*, 63(2):411–423, 2001.

[32] XL Xie and G Beni. A validity measure for fuzzy clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(8):841–847, 1991.

[33] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time, 2002.