Available online at www.sciencedirect.com
ScienceDirect
Information Systems 33 (2008) 240–260
www.elsevier.com/locate/infosys

Continuous subspace clustering in streaming time series ☆

Maria Kontaki*, Apostolos N. Papadopoulos, Yannis Manolopoulos

Department of Informatics, Aristotle University, 54124 Thessaloniki, Greece

Received 9 January 2007; accepted 11 September 2007
Recommended by F. Carino Jr.

Abstract

Performing data mining tasks in streaming data is considered a challenging research direction, due to the continuous data evolution. In this work, we focus on the problem of clustering streaming time series, based on the sliding window paradigm. More specifically, we use the concept of subspace α-clusters. A subspace α-cluster consists of a set of streams, whose value difference is less than α in a consecutive number of time instances (dimensions). The clusters can be continuously and incrementally updated as the streaming time series evolve with time. The proposed technique is based on a careful examination of pair-wise stream similarities for a subset of dimensions and then it is generalized for more streams per cluster. Additionally, we extend our technique in order to find maximal pClusters in consecutive dimensions that have been used in previously proposed clustering methods. Performance evaluation results, based on real-life and synthetic data sets, show that the proposed method is more efficient than existing techniques. Moreover, it is shown that the proposed pruning criteria are very important for search space reduction, and that the cost of incremental cluster monitoring is more computationally efficient that the re-clustering process.

Keywords: Continuous processing; Subspace clustering; Streaming time series; Sliding window

1. Introduction

The study of query processing and data mining techniques for data stream processing has recently attracted the interest of the research community [1–3], due to the fact that many applications manage data that change very frequently with respect to time. Examples of such emerging applications are network monitoring, financial data analysis, sensor networks to name a few. The most important property of data streams is that new values are continuously arrive, and therefore efficient storage and processing techniques are required to cope with (usually) high update rates.

Due to the highly dynamic nature of data streams, random access is prohibitive. Therefore, each data stream is possible to be read only once (or a limited number of times). This feature poses additional difficulties for query processing, since the data can only be accessed in arrival order. Moreover, additional methods are required for data mining tasks, such as clustering and association rule discovery, to cope with the data evolution.

A streaming time series s is a sequence of real values $s[1]$, $s[2], \ldots$, where new values are

☆ Research supported by the PENED 2003 program, funded by the General Secretariat for Research and Technology, Ministry of Development, Greece.
*Corresponding author. Tel.: +30 23 10991924; fax: +30 2310991913.
E-mail address: kontaki@csd.auth.gr (M. Kontaki).

0306-4379/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.is.2007.09.001

continuously appended as time progresses. For example, a temperature sensor which monitors the environmental temperature every five minutes, produces a streaming time series of temperature values. As another example, consider a car equipped with a GPS device and a communication module, which transmits its position to a server every 10 min. A streaming time series of two-dimensional points (the x and y coordinates of its position) is produced. Note, that in a streaming time series, data values are ordered with respect to arrival time. New values are appended at the end of the series.

1.1. Motivation

Clustering is considered an important data mining task and significant results have been reported for several types of data [4,5]. The challenge in a set of streaming time series is to update the clustering information as time progresses, avoiding the re-clustering process which is a computationally intensive procedure. It is desirable to use incremental clustering algorithms to enable continuous clustering.

Given a set of streaming time series, clustering can be applied to all available values within a specified length, which is known as the *sliding window* model. The size of the sliding window defines the dimensionality of each streaming time series. For example, a sliding window of size 256 means that each time series is considered as a 256-dimensional vector. Each dimension corresponds to a time instance. Searching for clusters in a large number of dimensions may result to failure, because as the size of the sliding window increases the probability that two streams will belong to the same cluster decreases. In many cases, although two or more streams do not belong to the same cluster for the whole sliding window, they do so by considering a subset of dimensions.

Fig. 1 illustrates three streaming time series A, B and C with a sliding window of size 17. We assume that two streams belong to the same cluster if the difference of the values in the corresponding dimensions is less than or equal to 2. By inspecting Fig. 1, it is evident that these streams cannot belong to the same cluster, since the difference of values in several dimensions is more than 2. For example, the value difference of A and B in the second dimension is $7-4=3$. However, by considering subsets of dimensions, streams A and B belong to the same cluster for the dimension intervals $[d_3, d_6]$, which contains d_3, d_4, d_5, d_6 and $[d_9, d_{17}]$, which contains d_9, d_{10}, d_{11}, d_{12}, d_{13}, d_{14}, d_{15}, d_{16}, d_{17}. It is evident that the value difference of streams A and B in each of these dimensions is less than or equal to 2.

The basic requirements for the generation of subspace clusters is that each cluster should contain a sufficient number of streams, in a sufficient number of consecutive dimensions. The generated subspace clusters contribute to the discovery of useful knowledge, since they reveal a high degree of similarity among streams participating in the same cluster.

The usefulness of the proposed clustering scheme is twofold. First, it can be used as a subspace clustering tool in several application domains such as:

- *Sensor monitoring*. Sensors that measure similar values in the same period of time, designate that

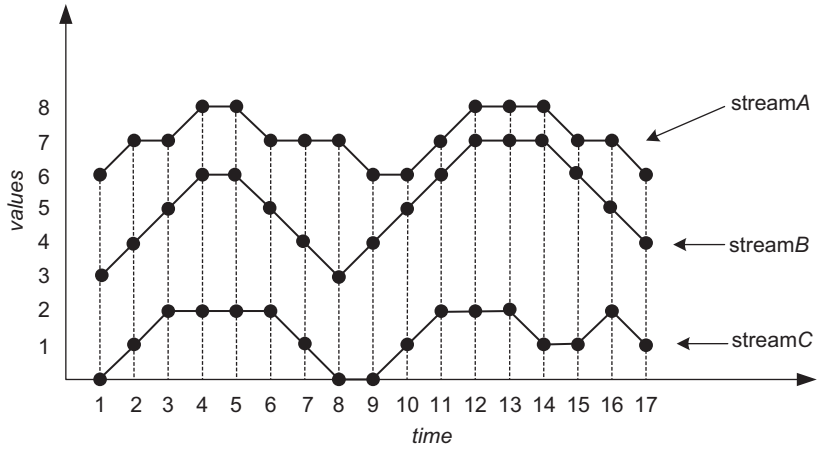

Fig. 1. Example of subspace clustering.

the environmental conditions in the corresponding locations are similar (e.g., temperature, wind speed, seismic behavior).
- *Traffic analysis*. A telephone company can cluster its subscribers in order to group them according to the cost of their calls. Special prices can be offered to a number of groups, at specific time intervals, to increase the company profit.

Second, it serves as a base to calculate pClusters in consecutive dimensions incrementally. pClusters [6,7] have been used as a method to find subspace clusters in any subset of the dimensions in static data. However, in streaming data, a major property is that the data are time ordered. So, in this paper, we deal with pClusters on consecutive dimensions.

1.2. Contribution

Although the literature is reach in methods and techniques for clustering, subspace clustering and continuous query processing, to the best of our knowledge, there is no work on continuous subspace clustering when the data evolve with time. Therefore, we present a methodology to attack the problem and we study effective algorithms toward efficient subspace α-cluster generation for a set of streaming time series. Toward this direction, we propose a method to update the clusters when new stream values become available, avoiding the process of re-clustering. The generated α-clusters are defined only on consecutive dimensions.

Moreover, we investigate the incremental computation of pClusters in consecutive dimensions, which are based on the use of the pScore measure proposed in [6,7]. It is shown that the method to produce subspace α-clusters is easily adapted to produce pClusters as well. The most efficient algorithm for generating pClusters in any subset of the available dimensions is MaPle [6]. However, MaPle generates pClusters only in static time series. The proposed method can be used to generate pClusters incrementally. In addition, the proposed method can also be used to generate pClusters in static time series efficiently, outperforming MaPle significantly. However, the proposed method can find pClusters only in consecutive dimensions contrary to MaPle that can find pClusters in any subset of the dimensions.

In summary, the contributions of this work have as follows:

(i) the study of the subspace clustering problem in streaming time series,
(ii) the study of continuous subspace clustering taking into account the time series evolution,
(iii) the incremental computation of pClusters in consecutive dimensions,
(iv) the performance evaluation of the proposed method based on real-life and synthetic data sets, and
(v) the performance comparison between the proposed method and MaPle, the state-of-the-art subspace clustering algorithm to generate pClusters.

The rest of the work is organized as follows. In Section 2 we discuss the appropriate related work. Section 3 studies in detail the proposed method for continuous clustering of streaming time series. Section 4 illustrates the incremental computation of pClusters by using the proposed algorithm. Section 5 presents performance evaluation results based on real-life and synthetic data sets, whereas Section 6 concludes the work.

2. Related work

Clustering is an important research direction with significant research contributions [4,5]. In [8] and [9] it has been demonstrated that similarity search and clustering respectively is *meaningless* for spaces that are embedded in a very large number of dimensions. This observation has lead a significant number of researchers to study alternative clustering methodologies. One research direction that has been followed, is subspace clustering.

In [10], the authors have studied the problem of subspace clustering in a high-dimensional space and they have proposed CLIQUE, which is a grid-based bottom-up algorithm to discover density-based clusters. The space is divided in equal-sized cells, and the density of each cell is computed as the fraction of the number of the objects in the cell over the total number of objects. CLIQUE determines dense cells and merges them to create clusters in a high-dimensional space. In [11], the concept of entropy is used to determine a dense cell, whereas in [12] a density-connected method has been proposed based on DBSCAN [13]. A different approach has been followed in [14], where the Fascicle method has

been proposed as a scheme to cluster and compress data. The aforementioned methods apply to static data sets, and their adaptation to the streaming case is not obvious.

In [15–17] the authors have proposed top-down algorithms for subspace cluster discovery. The basic drawback of these methods is the usage of parameter k, which is the number of subspace clusters that each method should report. In many real applications, this value is not known a priori.

Several research contributions have used δ-clusters for subspace clustering [6,7,17–19]. However, the concept of δ-clusters is treated differently. In [18], δ-biclusters have been proposed to find subspace clusters in a set of genes and conditions of DNA microarrays. In [7], the pScore metric has been proposed to measure the coherence of a cluster. The method determines object and attribute pair-wise clusters and utilizes a prefix-tree to generate clusters in a high-dimensional space. The same metric has been used in [6] to find pair-wise clusters, along with a depth-first-search algorithm to prune redundant non-maximal clusters. In [19], it has been shown that the above methods do not scale well in large data sets, and therefore the authors have proposed SeqClus. SeqClus is based on the Counting Tree data structure, that provides a compact summary of the dense patterns in a data set. Using the occurrences of a dense pattern, SeqClus generates subspace clusters. The above methods operate on static data sets. It is not straightforward to apply these methods for the streaming case, since they either rely on: (i) access methods that should be updated continuously to cope with the arrival rate of new stream values, or (ii) algorithms that cannot adapt easily to the incremental case.

Recently, the problem of data stream clustering has attracted the interest of the research community [20–24]. The majority of these contributions apply the k-median clustering technique. The fundamental characteristic of the proposed methods is that they attack the problem of incremental clustering considering one data stream only. However, this characteristic is quite restrictive, taking into account that modern applications require the management of a large number of data streams.

Due to the dynamic nature of the data streams, continuous and incremental algorithms are necessary to process streaming time series. Recent research work in continuous query processing includes [20,21,23,24]. These research contributions study various aspects of continuous query processing issues, taking into consideration that the update rate may be high.

To the best of the authors' knowledge, this is the first attempt to solve the incremental subspace clustering problem in streaming time series.

3. Incremental clustering

For illustration purposes, stream values in the last W dimensions are represented by a matrix, in which the rows represent the streams and the columns represent the last W time instances. Table 1 summarizes the basic symbols and the corresponding definitions that are used throughout the study. We begin our exploration with a number of basic definitions that are used for the rest of the work.

Definition 1 (*simple α-cluster*). A *simple α-cluster* contains a number of streams with pair-wise distances at most α in a single dimension. There is no restriction applied to the number of streams contained in each cluster.

The jth simple α-cluster in the ith dimension is represented as $c_{i,j}$. The previous definition does not take into consideration possible restrictions applied to the number of streams in each cluster and the number of consecutive dimensions. By forcing each cluster to contain at least *minRows* streams and at least *minCols* dimensions we have:

Definition 2 (*subspace α-cluster*). A *subspace α-cluster* contains at least *minRows* streams, for which

Table 1
Basic symbols used throughout the study

Symbol	Description
s, s_i	a streaming time series
$s[i]$	the value of s in the ith dimension
N	the number of streams
W	the size of the sliding window
C_i	a maximal subspace α-cluster
$c_{i,j}$	the jth simple α-cluster of the ith dimension
c, c'	simple α-clusters
m	number of streams in a cluster
G, G_i	a group of candidate α-clusters
minRows	minimum number of streams contained in a subspace α-cluster
minCols	minimum number of consecutive dimensions contained in a subspace α-cluster
α	maximum distance between any two streams for a given dimension

the maximum value difference is at most α in at least *minCols* consecutive dimensions.

In the example illustrated in Fig. 1, assuming that $minRows = 2$, $minCols = 3$ and $\alpha = 2$, we have two generated subspace α-clusters containing streams A and B, defined by dimension ranges $[d_3, d_6]$ and $[d_9, d_{17}]$. However, assuming that $minCols = 5$, we have only one subspace α-cluster defined by the dimension range $[d_9, d_{17}]$. Moreover, assuming that $minRows = 3$ there is no subspace α-cluster, since we can not define a subspace α-cluster containing at least three streams.

A subspace α-cluster C is represented as a pair $(S, [d_i, d_j])$, where S is a set of streams and $[d_i, d_j]$ is an interval of $j - i + 1$ consecutive dimensions (time instances), where $i \leqslant j$. Evidently, the cardinality of S must be at least *minRows*, whereas the number of consecutive dimensions must be at least *minCols*. We assume that the streams contained in S are represented by their corresponding IDs. Furthermore, we assume that stream IDs are stored in S in a non-decreasing order.

Definition 3 (*maximal subspace α-cluster*). A subspace α-cluster $(S, [d_i, d_j])$ is *maximal*, if (a) we cannot find another α-cluster $(S, [d_k, d_l])$ such that $k \leqslant i$ and $l \geqslant j$ and (b) we cannot find another α-cluster $(T, [d_i, d_j])$ such that $S \subset T$.

Subspace α-clusters have a very convenient *closure property*, as it is demonstrated by the following Proposition.

Proposition 1 (*closure property*). *Let $C = (S, [d_i, d_j])$ be a subspace α-cluster, not necessarily maximal. Then every cluster $C' = (S', [d_k, d_l])$ such that $S' \subset S$, $k \geqslant i$, $l \leqslant j$, $|S'| \geqslant minRows$ and $l - k + 1 \geqslant minCols$, is also a subspace α-cluster.*

Proof. First, we show that if $C = (S, [d_i, d_j])$ is a subspace α-cluster, then $C' = (S', [d_i, d_j])$ is also a subspace α-cluster for $S' \subset S$. Assume that C' is not a subspace α-cluster. Since $|S'| \geqslant minRows$ and $j - i + 1 \geqslant minCols$, the only reason for the violation is the existence of at least two streams $s_1 \in S'$ and $s_2 \in S'$ such that their value difference in at least one dimension is more than α. However, since $s_1 \in S$ and $s_2 \in S$, we conclude that C is not a subspace α-cluster, which contradicts our initial assumption.

Next, we show that if $C = (S, [d_i, d_j])$ is a subspace α-cluster, then $C' = (S, [d_k, d_l])$ is also a subspace α-cluster for $k \geqslant i$, $l \leqslant j$. Again, assume that C' is not a subspace α-cluster. This means that there exist two streams $s_1 \in S$, $s_2 \in S$ and a dimension d_x, $k \leqslant x \leqslant l$ such that the value difference of the streams is more than α. However, since the dimension d_x is contained in C we conclude that C is not a subspace α-cluster, which contradicts our initial assumption. □

The power of the closure property lies in the fact that it is not necessary to discover all possible subspace α-clusters, but only a subset of them. This property is very similar to the Apriori principle [25] which has been used for association rule discovery.

We are ready now to proceed with the detailed description of the proposed methodology, which attacks the following problem:

Given a set of streaming time series, a maximum value difference α, a sliding window size W and two integer numbers *minRows* and *minCols*, determine all maximal subspace α-clusters continuously, where each cluster contains at least *minRows* streams, and the value difference is less than or equal to α, in at least *minCols* consecutive dimensions.

The proposed methodology comprises the following phases: (i) the initialization phase, which determines an initial set of maximal subspace α-clusters, and (ii) a series of update phases which incrementally maintain the clusters when new stream values become available.

3.1. Cluster initialization (CI)

The purpose of the cluster initialization phase (CI) is to determine an initial set of maximal subspace α-clusters, based on the last W values of each streaming time series. Each such cluster must contain at least *minRows* streams and at least *minCols* consecutive time instances.

The CI process comprises a series of steps. In the first step, each time instance (dimension) is inspected separately to determine simple α-clusters (which are defined in one dimension only). In the next step, the algorithm generates all clusters containing $m = 2$ streams in the maximum possible number of dimensions. In each subsequent step the algorithm tries to increase the number of streams contained in each cluster ($m = m + 1$), until the generation of all possible maximal subspace α-clusters, according to the values of α, *minRows* and *minCols*. Clusters that contain less than *minCols* dimensions are discarded permanently in each step of the algorithm, since they cannot contribute to the final answer.

We will illustrate the CI process by means of an example, which is depicted in Figs. 2–4. Assume that there are $N = 5$ streaming time series with a sliding window of size $W = 4$. Moreover, let $\alpha = 2$, $minRows = 4$ and $minCols = 3$. Fig. 2(a) shows the value of each stream in every dimension, Fig. 2(b) shows subsets of values that satisfy the α constraint, whereas Fig. 2(c) shows the generated simple α-clusters for $\alpha = 2$.

To determine the simple α-clusters for each dimension we proceed as follows. The values in each dimension are sorted in a non-decreasing order. The produced sorted sequence S is processed by means of two pointers p_{left} and p_{right}. Initially, p_{left} and p_{right} are placed on the first element of the sorted sequence. The pointer p_{right} is incremented until it reaches an element where $|S[p_{left}] - S[p_{right}]| > \alpha$. If this happens, then all elements $S[p_{left}]$, $S[p_{left}+1],\ldots,S[p_{right}-1]$ form a cluster in the corresponding dimension. Then, the pointer p_{left} is increased by one, and the same process is applied until p_{right} reaches the end of the sorted sequence. If two clusters end at the same element, the one containing the minimum number of elements is discarded.

Following the generation of the initial set of simple α-clusters, the next step considers pairs of streams and determines if there are any simple α-clusters with two streams ($m = 2$). Fig. 3(a) depicts the generated clusters for each dimension, whereas Fig. 3(b) shows all possible 2-level clusters that are generated. Each 2-level cluster is formed by combining two streams that have common simple α-clusters, in each one of at least *minCols* consecutive dimensions. The common simple α-clusters are illustrated in the fourth column of Fig. 3(b). The candidate 2-level clusters are separated in four different groups, as it is indicated by the dashed lines in Fig. 3(b). All candidate clusters in each group must share $m - 1$ streams and can differ in only the last one. Each group is treated separately, and therefore, we begin with the first group which is composed of candidate clusters containing stream s_1. Some of these clusters will be rejected, whereas the others will be used to form candidate 3-level clusters.

Proposition 2 (*cluster pruning criterion*). *If the number of candidate m-level clusters contained in a group is less than* $minRows - m + 1$ *then all the clusters in this group can be safely discarded from further consideration.*

Proof. Consider that we have a group G consisting of x m-level clusters, with $x < minRows - m + 1$. If the first m-level cluster of the group is combined with the rest $x - 1$ m-level clusters of the same group, then a new group is formed containing $x - 1$ $(m + 1)$-level clusters. If the second m-level cluster of G is combined with all $x - 2$ m-level clusters of G, then another group is formed containing $x - 2$ $(m + 1)$-level clusters. Therefore, in the $(m + 1)$th level, the maximum number of $(m + 1)$-level clusters

a

streams	d_1	d_2	d_3	d_4
s_1	7.4	4.6	6.7	5.5
s_2	6.4	3.7	8.2	8.0
s_3	8.1	3.9	8.6	5.6
s_4	5.2	6.0	5.5	5.8
s_5	8.0	3.2	7.8	8.3

b

dimension	clustered stream values
d_1	{5.2, 6.4}, {6.4, 7.4, 8.0, 8.1}
d_2	{3.2, 3.7, 3.9, 4.6}, {4.6, 6.0}
d_3	{5.5, 6.7}, {6.7, 7.8, 8.2, 8.6}
d_4	{5.5, 5.6, 5.8}, {8.0, 8.3}

c

dimension	simple α-clusters
d_1	$c_{1,1}=\{s_2, s_4\}$, $c_{1,2}=\{s_1, s_2, s_3, s_5\}$
d_2	$c_{2,1}=\{s_1, s_2, s_3, s_5\}$, $c_{2,2}=\{s_1, s_4\}$
d_3	$c_{3,1}=\{s_1, s_4\}$, $c_{3,2}=\{s_1, s_2, s_3, s_5\}$
d_4	$c_{4,1}=\{s_1, s_3, s_4\}$, $c_{4,2}=\{s_2, s_5\}$

Fig. 2. Cluster initialization (a) stream values, (b) stream values clustered per dinension and (c) all simple α-cluster.

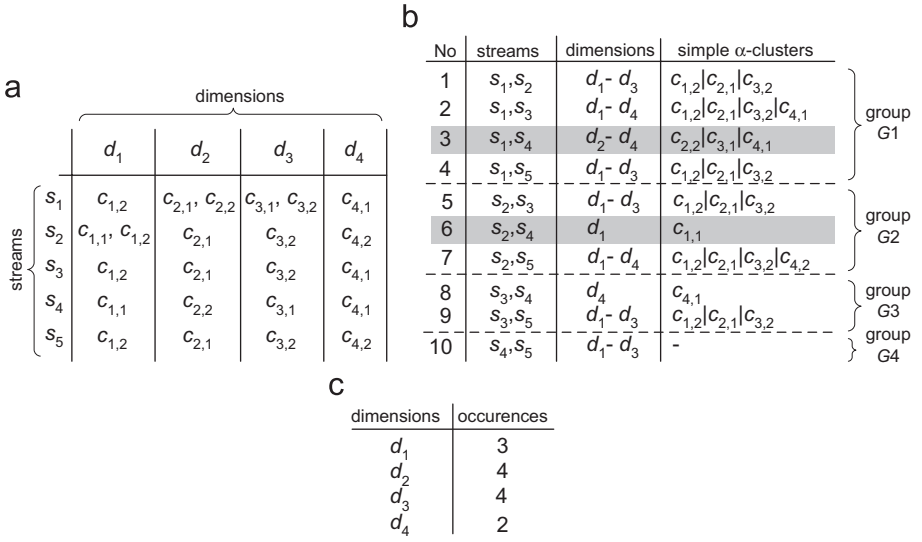

Fig. 3. Cluster initialization (continued) (a) simple α-cluster, (b) candidate 2-level α-cluster ($m = 2$) and (c) dimension occurences for group G1.

contained in a group will be at most $x - 1$. In the *minRows*-level, a group will have at most $x - minRows + m$ clusters, and $x < minRows - m + 1 \Rightarrow x - minRows + m < 1$. This means that we cannot have a subspace α-cluster, and thus, G can be discarded. □

Evidently, all candidate clusters in the first group survive the cluster pruning criterion. At a first glance, it seems that all four clusters qualify, since each pair of streams contain at least three dimensions. However, with a more careful look we can see that dimension d_4 must be rejected. The following proposition explains.

Proposition 3 (*dimension pruning criterion*). *If each candidate α-cluster in a group G contains exactly m streams and the number of occurrences of a dimension in G is less than $minRows - m + 1$, then this dimension cannot contribute to the generation of subspace α-clusters.*

Proof. Consider a group G of m-clusters. Assume that the occurrences of a dimension d_i in G is $o_m < minRows - m + 1$. Then, in the next level, the $m + 1$-level clusters of any group will be at most $x - 1$ (see proof of Proposition 2) and the number of occurrences of dimension d_i in G will be at most $o_{m+1} = o_m - 1$. The proof is similar to that of Proposition 2 and it is omitted. In the *minRows*-level, the number of occurrences of the d_i will be at most $o_{minRows} = o_m - minRows + m$. Therefore, $o_{minRows} < minRows - m + 1 - minRows + m \Rightarrow o_{minRows} < 1$, and this means that dimension d_i can not participate in a subspace α-cluster. □

If the application of the dimension pruning criterion affects an existing cluster, either the cluster will be rejected, if the number of dimensions is less than *minCols*, or will shrink (dimensionality shrinkage), if the number of dimensions is at least *minCols*. Applying the dimension pruning criterion to our case, it is evident that dimension d_4 has only two occurrences, (see Fig. 3(c)) and therefore must be rejected from further consideration. This means that cluster no. 3 contains the streams $\{s_1, s_2\}$ and the dimensions d_2 and d_3. However, since $minCols = 3$ this cluster is rejected from further consideration (shaded row of Fig. 3(b)).

In the next step, the method tries to merge the clusters that survived the previous step, toward the generation of clusters containing $m + 1$ streams. Therefore we try to combine the clusters no. 1 with no. 2, no. 1 with no. 4 and no. 2 with no. 4 (recall that no. 3 has been rejected). These combinations are depicted in tabular form in Fig. 4(a). The clusters are categorized in two different groups. Each group should contain clusters that share all stream IDs, except the last. For example, candidate clusters no. 1 and no. 2 are contained in the first group since they differ in the last stream only, and they have two streams in common s_1 and s_2. Again, at a first glance all three candidate clusters

a

No	streams	dimensions	simple α-clusters
1	s_1, s_2, s_3	d_1- d_3	$c_{1,2}\|c_{2,1}\|c_{3,2}$
2	s_1, s_2, s_5	d_1- d_3	$c_{1,2}\|c_{2,1}\|c_{3,2}$
3	s_1, s_3, s_5	d_1- d_3	$c_{1,2}\|c_{2,1}\|c_{3,2}$

b

No	streams	dimensions	simple α-clusters
1	s_1, s_2, s_3, s_5	d_1- d_3	$c_{1,2}\|c_{2,1}\|c_{3,2}$

Fig. 4. Cluster initialization (continued) (a) candidate 3-level α-clusters ($m = 3$) and (b) candidate 4-level α-clusters ($m = 4$).

of Fig. 4(a) qualify. However, cluster no. 3 can be safely rejected, as it is suggested by the cluster pruning criterion (Proposition 2). This is illustrated by the shaded row in Fig. 4(a).

By inspecting clusters no. 1 and no. 2 in the first group it is evident that both clusters survive both pruning criteria. Therefore, these two clusters can be combined toward the generation of a single 4-level cluster, which is illustrated in Fig. 4(b). Recall, that $minRows = 4$ and $minCols = 3$. Therefore, this cluster is recorded as an answer, since it contains four streams and these streams form a subspace α-cluster in three dimensions.

Let us now return to check the second group of clusters depicted in Fig. 3(b). The candidate cluster no. 6 would never be created by the algorithm, since it does not satisfy the *minCols* restriction. It is shown here only for demonstration purposes. This means that there are now only two candidate clusters in this group. According to the first pruning criterion these clusters should be discarded without any further consideration.

Up to this point, we have checked all candidate clusters of streams s_1 and s_2. Is it necessary to check the clusters for streams s_3, s_4 and s_5? The answer is negative, since each group does not survive the cluster pruning criterion. In fact, since there are three remaining streams it is impossible to generate a 4-level cluster ($minRows = 4$), as it is illustrated by the following Proposition.

Proposition 4 (*stream pruning criterion*). *If the number of remaining streams is less than minRows then all groups of candidate clusters generated by these streams can be safely discarded since it is impossible to give subspace α-clusters.*

Proof. Assume that there are x remaining streams for processing where $x < minRows$. When $m = 2$, the number of clusters that can be formed in a group is at most $x - 1$ (see Proposition 2). After $minRows - 2$ levels, where $m = minRows$, the number of clusters in a group will be at most $x - 1 - minRows + 2 = x - minRows + 1 < 1$, which means that $x < minRows$ streams cannot form subspace α-clusters. □

The CI algorithm terminates at this point and reports as an answer the cluster illustrated in Fig. 4(b). Recall that each group of candidate clusters has been studied separately. Is it possible to miss a possible cluster? The answer is negative as it is suggested by the following Proposition.

Proposition 5 (*correctness of CI algorithm*). *By treating each group of candidate clusters separately, it is impossible to miss a maximal subspace α-cluster.*

Proof. Assume that we have two candidate α-clusters $C_1 = (S_1, [d_i, d_j])$ and $C_2 = (S_2, [d_i, d_j])$, where $S_1 = \{s_1, \ldots, s_j, s_k, s_{j+1}, \ldots, s_{j+i}\}$ and $S_2 = \{s_1, \ldots, s_j, s_l, s_{j+1}, \ldots, s_{j+i}\}$. These clusters differ only in one stream, which is not the last, and therefore belong to different groups. Each one contains exactly $m = j + i + 1$ streams.

Let $C_3 = (S_3, [d_i, d_j])$ be an α-cluster, that can be generated from the combination of C_1 and C_2, where $S_3 = \{s_1, \ldots, s_j, s_k, s_l, s_{j+1}, \ldots, s_{j+i}\}$. Then, due to the closure property, the following clusters also exist: $C_4 = (S_4, [d_i, d_j])$ and $C_5 = (S_5, [d_i, d_j])$, where $S_4 = s_1, \ldots, s_j, s_k, s_l, s_{j+1}, \ldots, s_{j+i-2}, s_{j+i-1}$ and $S_5 = s_1, \ldots, s_j, s_k, s_l, s_{j+1}, \ldots, s_{j+i-2}, s_{j+i}$. Note that C_4 and C_5 belong to the same group, since they differ in the last stream only. This means, that C_3 will be generated by combining C_4 and C_5, and therefore, the combination of C_1 and C_2 is not required. □

By the aid of Proposition 5, algorithm CI computes all maximal subspace α-clusters, by considering only candidate α-clusters which belong to the same group. This way, it is impossible to discover the same cluster more than once and therefore, less computational effort is required. The outline of the CI algorithm is depicted in Fig. 5.

3.2. Cluster maintenance (CM)

The purpose of the cluster maintenance phase (CM) is to keep the clustering information up to

Algorithm CI (S,α, $minRows$, $minCols$, W)
Input
S: set of streams,
α: maximum value difference for a dimension in a cluster,
$minRows$: minimum number of streams per cluster,
$minCols$: minimum number of dimensions per cluster,
W: sliding window size
Output
A: set of maximal subspace α-clusters

```
1.  for i=1 to W
2.      compute all simple α-clusters for dimension d_i;
3.  end for
4.  for i=1 to N − minRows + 1
5.      set m = 2;
6.      generate m-level candidate α-clusters for stream i;
7.      apply cluster pruning;
8.      apply dimension pruning;
9.      while there exist m-level candidates do
10.         generate m + 1-level candidate α-clusters that
            contain minCols or more dimensions;
11.         increase m;
12.         if m ≥ minRows and
13.         C is maximal subspace α-cluster then
14.             update A;
15.         end if
16.         apply cluster pruning;
17.         apply dimension pruning;
18.     end while
19. end for
20. report A;
```

Fig. 5. Outline of CI algorithm.

date, taking into consideration the new stream values. This phase is executed when new values for the streams become available for the next time instance. We distinguish two different cases, which are handled by different algorithms:

(1) all streams update their values in the next time instance and
(2) only one stream updates its value in the next time instance.

3.2.1. Multiple updates per dimension (CM-UPALL algorithm)

In this case, all streams update their values in each time instance.[1] Since the processing is based on the sliding window paradigm, the left-most dimension should be discarded and a new one should be included. An example is illustrated in Fig. 6(a), where stream values in dimension d_1 should be rejected, whereas stream values in dimension d_5 should be taken into consideration to update the clustering information. This requires the deletion of all simple α-clusters of dimension d_1 and the determination of all simple α-clusters for dimension d_5. These clusters are illustrated in Fig. 6(b).

The CM algorithm CM-UPALL, which is depicted in Fig. 7, operates in two steps.

(1) Initially, existing maximal subspace α-clusters are checked, since some of them may be rejected due to the deletion of dimension d_1. Moreover, some of the existing clusters may be expanded by the inclusion of the newly created dimension d_5.

[1]This does not necessarily mean that the new values will be different than the previous ones.

a

streams	deleted d_1	d_2	d_3	d_4	new d_5
s_1	7.4	4.6	6.7	5.5	8.2
s_2	6.4	3.7	8.2	8.0	6.1
s_3	8.1	3.9	8.6	5.6	8.3
s_4	5.2	6.0	5.5	5.8	5.5
s_5	8.0	3.2	7.8	8.3	5.0

b

streams	deleted d_1	d_2	d_3	d_4	new d_5
s_1	$c_{1,2}$	$c_{2,1}$, $c_{2,2}$	$c_{3,1}$, $c_{3,2}$	$c_{4,1}$	$c_{5,2}$
s_2	$c_{1,1}$, $c_{1,2}$	$c_{2,1}$	$c_{3,2}$	$c_{4,2}$	$c_{5,1}$
s_3	$c_{1,2}$	$c_{2,1}$	$c_{3,2}$	$c_{4,1}$	$c_{5,2}$
s_4	$c_{1,1}$	$c_{2,2}$	$c_{3,1}$	$c_{4,1}$	$c_{5,1}$
s_5	$c_{1,2}$	$c_{2,1}$	$c_{3,2}$	$c_{4,2}$	$c_{5,1}$

Fig. 6. Simple α-clusters after the arrival of d_5 (a) stream values and (b) simple α-clusters.

Algorithm CM-UPALL (S, α, $minRows$, $minCols$, W)
Input
S: set of streams,
α: maximum value difference for a dimension in a cluster,
$minRows$: minimum number of streams per cluster,
$minCols$: minimum number of dimensions per cluster,
W: sliding window size
Output
A: set of maximal subspace α-clusters

1. delete all the simple α-clusters of the first dimension;
2. find all the simple α-clusters for the new dimension;
3. update and expand the existing maximal subspace α-clusters;
4. delete the clusters that have less than $minCols$ dimensions;
5. **for** i=1 to $N - minRows + 1$
6. set $m = 2$;
7. generate m-level candidate α-clusters of stream i only for the last $minCols$ dimensions;
8. apply cluster pruning;
9. **while** there exist m-level candidates **do**
10. generate $m + 1$-level candidate α-clusters that contain $minCols$ dimensions;
11. increase m;
12. **if** $m \geq minRows$ **and**
13. C is maximal subspace α-cluster **then**
14. update A;
15. **end if**
16. apply cluster pruning;
17. **end while**
18. **end for**
19. report A;

Fig. 7. Outline of CM-UPALL algorithm.

(2) Next, the algorithm searches for new maximal subspace α-clusters that may be generated due to the arrival of the new dimension d_5.

Initially, each cluster containing d_4 as its rightmost dimension is checked for possible expansion by adding dimension d_5. If the cluster can be expanded, then it is included in the answer. Next, the dimension d_1 is deleted from all clusters that contain it. If by deleting dimension d_1 a cluster is left with less than *minCols* dimensions, then it is deleted. Finally, the other clusters that are not

affected by the deletion of dimension d_1 and the inclusion of dimension d_5 are considered part of the new answer.

In order to search for new clusters that may have been formed due to the inclusion of dimension d_5, the algorithm inspects only the last *minCols* dimensions. The reason for this is given in the following proposition.

Proposition 6 (*correctness of CM-UPALL algorithm*). *Let d_{new} be the newly created dimension. To search for new clusters it is sufficient to study the last minCols dimensions* (*i.e.*, $d_{new-minCols+1}$, $d_{new-minCols+2}, \ldots, d_{new}$).

Proof. Assume that we take into account the last $minCols + k$ dimensions, where $k \geqslant 1$ and $minCols + k \leqslant W$. If there exists a cluster C in the last $minCols + k$ dimensions, then this means that the cluster was present also in the $minCols + k - 1$ dimensions (due to the closure property), and therefore the cluster is not new. Consequently, it is safe to search for new clusters only in the last *minCols* dimensions. □

Due to Proposition 6, the method considers only the last *minCols* dimensions. Therefore, if a dimension does not participate in a cluster, the cluster is rejected because it violates the *minCols* restriction. This means that dimension pruning is not necessary, and only cluster pruning needs to be applied.

3.2.2. One update per dimension (CM-UPONE algorithm)

In this case, only one stream updates its value in each time instance. As in the previous case, the method first updates the simple α-clusters and after that, performs changes to the existing maximal subspace α-clusters, if needed. The outline of the CM-UPONE algorithm is illustrated in Fig. 8.

Let s be the stream which updates its value. To update the simple α-clusters the algorithm performs the following steps, for each dimension:

(1) First, the stream s is deleted from the simple α-clusters that belongs to, using the value that the corresponding dimension has.
(2) Second, the stream s is inserted in existing simple α-clusters or it is inserted in a new one, using the new value of the dimension.

Each simple α-cluster c_{ij} contains the minimum and the maximum value of its streams. The simple α-clusters are ordered, i.e., the minimum (maximum) of a cluster is always greater than the minimum (maximum) of the previous one. Thus, the method determines the first cluster that contains the old value. The method scans the remaining simple a-clusters, until a cluster that does not contain the stream s is found. Since the clusters are ordered, the remaining clusters cannot contain s.

To insert s, the method again scans the available simple α-clusters, for each dimension. Stream s can be inserted in a cluster, that its bounds contain, or can be extended to contain, its new value. Assume, that the new value of s is v and the minimum and maximum values of a simple α-cluster c is min_c and max_c respectively. If $v > min_c + \alpha$, then s cannot be inserted in c. The method continues to scan subsequent clusters, until a cluster c' is found such that $v < max_{c'} - \alpha$. Stream s is inserted in all simple α-clusters between c and c' and their bounds are updated if necessary. If c and c' are consecutive, a new cluster should be created to insert s. The new cluster is placed between c and c'.

The next step is to update the maximal subspace α-clusters. All clusters containing the updated stream are deleted. The remaining clusters are not affected. To find new maximal subspace α-clusters, the method computes 2-level candidate α-clusters by combining the updated stream with every other. In the subsequent steps, the algorithm tries to increase the number of streams contained in each cluster.

3.3. Performance issues

Here we study some performance issues regarding the proposed method by discussing the necessary auxiliary data structures required to guarantee efficient cluster monitoring. We investigate each phase of the algorithm separately.

The CI phase requires the determination of all simple α-clusters for each dimension. The dimensionality of the streaming time series is determined by the parameter W which defines the size of the sliding window. To determine the simple α-clusters for any dimension, we require that the values in the corresponding dimension are sorted in non-decreasing order. Alternatively, we can utilize a heap data structure to store the values of each dimension. In any case, the required complexity is $O(W \cdot N \cdot \log N)$. The preprocessing of each dimension can be performed when new values become available.

Algorithm CM-UPONE (s_i, S, α, $minRows$, $minCols$, W)
Input
s_i: the updated stream s_i,
S: set of streams,
α: maximum value difference for a dimension in a cluster,
$minRows$: minimum number of streams per cluster,
$minCols$: minimum number of dimensions per cluster,
W: sliding window size
Output
A: set of maximal subspace α-clusters

```
1.  for i=1 to W
2.      update all simple α-clusters for dimension d_i;
3.  end for
4.  delete existing maximal subspace α-clusters which contain s_i;
5.  set m = 2;
6.  generate m-level candidate α-clusters for stream s_i;
7.  apply cluster pruning;
8.  apply dimension pruning;
9.  while there exist m-level candidates do
10.     generate m + 1-level candidate α-clusters that
        contain minCols or more dimensions;
11.     increase m;
12.     if m ≥ minRows and
13.     C is maximal subspace α-cluster then
14.         update A;
15.     end if
16.     apply cluster pruning;
17.     apply dimension pruning;
18. end while
19. report A;
```

Fig. 8. Outline of CM-UPONE algorithm.

The above cost is insignificant compared to the cost required to generate the clusters in each level. Recall that to generate m-level α-clusters, the $(m-1)$-level clusters are required. It can be shown that the total number of possible clusters that can be generated is $2^N - 1$, where N is the number of streaming time series. However, the application of the pruning criteria manages to reduce drastically the number of generated clusters. This effect is demonstrated in Fig. 9, which depicts (1) the total number of clusters in each level, (2) the number of pruned clusters due to cluster pruning, (3) the number of pruned clusters due to dimension pruning and (4) the number of affected clusters by the dimensionality shrinkage. It is evident, that the majority of the candidate α-clusters is discarded. Cluster pruning is more significant when it happens in the first levels, since more clusters are pruned subsequently.

Algorithms CI, CM-UPALL and CM-UPONE require the support of some fundamental operations toward fast lookups of streams, clusters and dimensions. We do not present each operation in detail, since all of them can be efficiently supported by hashing schemes. The required operations are summarized in Table 2.

4. Incremental computation of pClusters

In this section, we study the problem of the incremental computation of pClusters defined by means of the pScore measure. The pScore measure has been proposed in [7] to determine subspace similarities among multidimensional data. More specifically, the pScore measure has been used to effectively determine co-expressed genes in micro-array data [6,7].

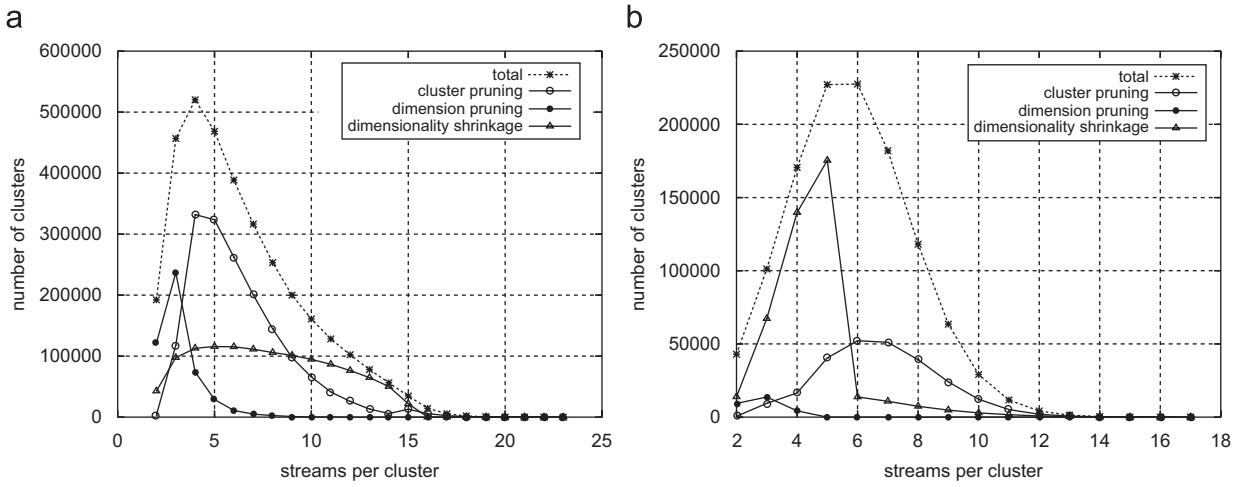

Fig. 9. Pruning power for STOCKS data set (a) $\alpha = 0.2$, $minRows = 15$, $minCols = 3$, $W = 100$ and (b) $\alpha = 0.2$, $minRows = 5$, $minCols = 6$, $W = 100$.

Table 2
Fundamental operations required by CI, CM-UPALL and CM-UPONE algorithms

Algorithm	Operations required
CI	1. find the simple α-clusters that each stream belongs to 2. find the streams that each subspace α-cluster contains 3. find the dimensions that each subspace α-cluster contains 4. find the simple α-clusters that each subspace α-cluster contains
CM-UPALL	1. find the simple α-clusters containing the dimension that is deleted 2. find the subspace α-clusters which start at the deleted dimension 3. find the subspace α-clusters which end at the last dimension 4. all that CI requires
CM-UPONE	1. find the subspace α-clusters which contain a specific stream 2. all that CI requires

Let $r_{i,j}$ be the value of the ith stream (row) on the jth dimension (column). The pScore metric is defined as follows:

$$pScore = |(r_{x,a} - r_{y,a}) - (r_{x,b} - r_{y,b})|.$$

We proceed with the definition of pClusters. Let S be a subset of streams and D be a subset of dimensions. Then, S and D form a δ-pCluster if for any two streams x, $y \in S$ and for any two dimensions a, $b \in D$, $pScore \leqslant \delta$, where $\delta \geqslant 0$. Moreover, to avoid discovering statistically insignificant pClusters, the authors proposed the use of two parameters *minRows* and *minCols*, denoting the minimum number of streams and dimensions respectively, that a pCluster contains.

We can use the proposed algorithms to determine pClusters in consecutive dimensions by applying a simple modification. If we change the definition of simple α-clusters then the resulting output of our algorithms will be α-pClusters. The following proposition explains:

Proposition 7 (*modification of simple α-cluster*). *If we compute the differences between dimensions d_{j-1} and d_j, and use the resulting values for simple α-cluster generation, then the maximal subspace α-clusters that our method produces are identical to the maximal δ-pClusters in a number of consecutive dimensions, where $\delta = \alpha$.*

Proof. Let $D_{i,j}$ be the difference of the stream i in dimensions j and $j-1$, i.e., $D_{i,j} = |r_{i,j} - r_{i,j-1}|$. Assume that we have two streams x and y in two consecutive dimensions a and b. Taking into account the new definition of the simple α-clusters, streams x and y will be in the same simple α-cluster if:

$$|D_{x,b} - D_{y,b}| \leqslant \alpha.$$

These streams will be in the same α-pCluster if the following holds:

$$|(r_{x,a} - r_{y,a}) - (r_{x,b} - r_{y,b})| \leqslant \alpha$$
$$\Rightarrow |(r_{y,b} - r_{y,a}) - (r_{x,b} - r_{x,a})| \leqslant \alpha$$
$$\Rightarrow |D_{y,b} - D_{x,b}| \leqslant \alpha$$
$$\Rightarrow |D_{x,b} - D_{y,b}| \leqslant \alpha. \quad \square$$

The above proposition suggests that if we are interested in determining δ-pClusters in consecutive dimensions, we can use the proposed modified algorithms and set the parameter α equal to the desired δ. The modified algorithms differ from CI, CM-UPALL and CM-UPONE algorithms only in line 2, where the simple α-clusters are computed or updated.

To clarify the above observation we give a simple example. Assume again, that we have two streams x and y in two consecutive dimensions a and b. The values of x is 5 and 17 for a and b dimensions respectively and the values of y is 3 and 16. If $minRows = 2$, $minCols = 2$ and $\delta = 1$, these streams belong to the same pCluster, since pScore $= |(5 - 3)–(17–16)| = 1$. By using the same parameters ($\alpha = 1$) and the first definition of simple α-clusters, these streams do not belong to the same maximal subspace α-cluster, since in the a dimension their difference is 2, so they do not belong in the same simple α-cluster. By using the same parameters ($\alpha = 1$) and the above definition of the simple α-clusters, these streams forms a maximal subspace α-cluster. The differences of stream x is $|5 - 17| = 12$ and of stream y is $|3 - 16| = 13$. Therefore, they belong to the same simple α-cluster. The maximal subspace α-cluster is identical to the δ-pCluster.

To the best of our knowledge, the most efficient algorithm for computing δ-pClusters, based on the pScore metric, is MaPle [6]. However, MaPle can be applied only on static data, whereas our method is applicable in streaming time series data. Moreover, since MaPle generates subspace clusters in any subset of the available dimensions, it has been modified toward generating only clusters in consecutive dimensions. As it is demonstrated in the next section, the proposed algorithm is significantly more efficient than MaPle in all experiments conducted.

5. Performance evaluation

The proposed algorithms and the algorithm MaPle have been implemented in C++ and all experiments have been conducted on a Pentium IV machine at 3.6 GHz, with 1 GBytes RAM, running Windows XP Professional. In the sequel, we present the data sets that have been used in our experiments and the experimental results obtained by the performance study.

5.1. Data sets

The performance evaluation is based on real-life data sets (STOCKS, ECG, TAO, YEAST), as well as on synthetically generated (SYNTHETIC). A short description of the data sets follows:

SYNTHETIC. The SYNTHETIC data set has been produced by means of a data generator. The generator takes several input parameters such as: the number of streams, the sliding window size, the length of each stream, the number of maximal subspace α-clusters, the values for *minRows*, *minCols* and α. The generator computes exactly the desired number of maximal subspace α-clusters with *minRows* streams in *minCols* dimensions. The remaining values are calculated randomly in such a way that the values do not fall in any cluster and no additional clusters are formed.

STOCKS. The STOCKS data set consists of a number of time series denoting the closing prices of stocks and can be obtained from http://www.finance.yahoo.com. Each stock has been subdivided to a number of sub-series of length 200, to obtain a total of 2313 different streaming time series.

ECG. The ECG data set contains electrocardiograms of two-channel recordings and can be obtained from the MIT-BIH Arrhythmia Database (http://www.physionet.org/physiobank/database/mitdb/). Each channel was digitized at 360 samples per second. We chose an electrocardiogram of a 69 years old male, containing 650 000 samples. To form the data set, we picked 30 000 of out of the 650 000 points randomly and each time series is formed from the consecutive 200 values of the selected point. The data set consists of 30 000 different streams.

TAO. The TAO data set (Tropical Atmosphere Ocean) contains wind speed measurements of 65 sites located on the surface of Pacific and Atlantic Ocean. The data set can be obtained from the Pacific Marine Environmental Laboratory (http://www.pmal.noaa.gov/tao). We have used the highest available resolution (e.g., the sampling time interval). About 4000 streams form the data set, and the maximum length of each one is set to 200.

YEAST. The YEAST data set contains the expression levels of 2884 genes under 17 conditions. The YEAST data set can be obtained from http://www.arep.med.harvard.edu/biclustering/yeast.matrix.

5.2. Experimental results

In the sequel, we present the experimental results obtained by (i) the comparison between the proposed algorithms CI, CM-UPALL and CM-UPONE and (ii) the comparison of MaPle and the proposed incremental algorithm, taking into account the CI cost.

5.2.1. Performance of the proposed methods

To evaluate the performance of the methods we used a synthetic data set. The parameter values used (if not otherwise specified) are: the number of streams (N) is 5000, the sliding window (W) is 100, $\alpha = 0.0$, the number of embedded maximal subspace α-clusters is 100, and each one contains 50 streams in 10 dimensions.

In the first experiment, we examine the scalability of the method with respect to the size of the sliding window (W) and the number of streams (N). The corresponding results are depicted in Fig. 10. In this experiment the CI method was applied in each update so the CI cost denotes the cost of reclustering the streaming time series. For the CM-UPALL and CM-UPONE methods, we give only the update costs since the initialization phase is applied only once for both methods. Fig. 10(a) illustrates the scalability of the method with respect to the sliding window size. The time required for CI phase is significantly more than that of both update methods. Moreover, the cost of the CM-UPONE increases with respect to W but the update cost for CM-UPALL is almost steady. This is because the CM-UPONE updates the simple α-clusters of each dimension that contains the updated stream, while the CM-UPALL updates only the simple α-clusters of the new dimension. Fig. 10(b) depicts the scalability of the method with respect to the number of streams. In order to have a similar set up, we generated different synthetic data sets consisting of 1000 to 20 000 streams. In each data set, we embedded 100 maximal subspace α-clusters, but we varied the *minRows* parameter so that the number of values used in the clusters to be proportional to the total number of values. When the number of streams increases significantly, the cost of CM-UPALL is higher than that of CM-UPONE. This happens because: (i) the cost of determining the simple α-clusters of the last dimension increases with the number of streams and (ii) CM-UPALL tries to find new subspace α clusters for all streams, whereas CM-UPONE tries to find new subspace α-clusters only for the updated stream. Again, the cost for CM is much less than that for CI. This suggests that reclustering should be avoided, since the respective computational cost is prohibitive. In the subsequent results, the CI cost is not shown for clarity purposes.

In the second experiment, we study the performance of the methods with respect to parameters *minRows* and *minCols*. Recall, that if all streams are updated, the method examines only the last *minCols* dimensions, whereas if one stream is

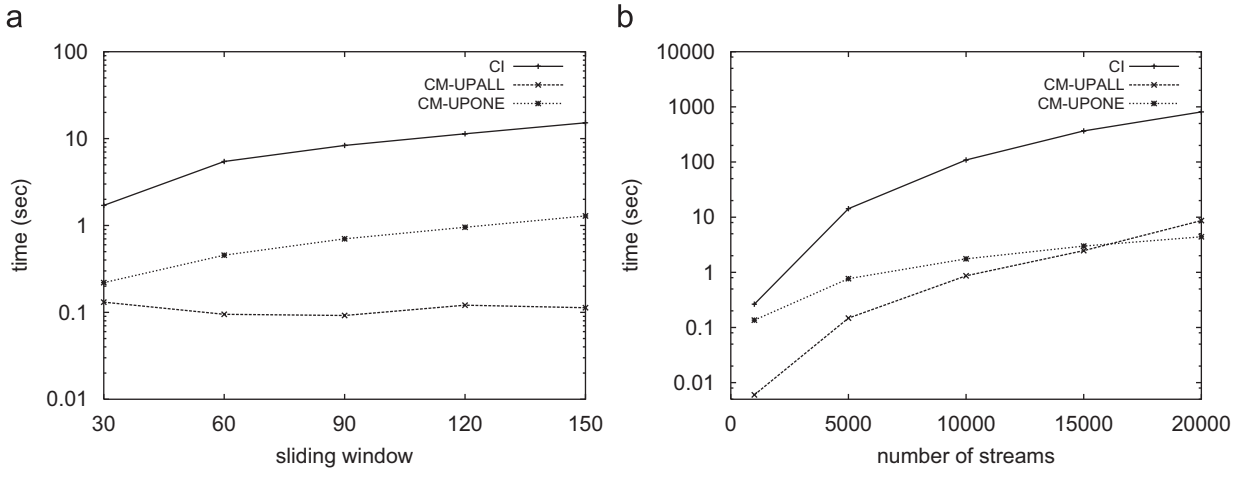

Fig. 10. Response time vs. (a) sliding window size and (b) number of streams for SYNTH data set.

updated, the method searches for new maximal subspace α-clusters only for the updated stream. Fig. 11 illustrates the impact of the parameters to the update algorithms. As expected, the cost of CM-UPALL decreases as *minRows* and *minCols* increase, whereas this is not true for CM-UPONE. This is because the cost of updating the simple α-clusters is more significant than that of finding new subspace α-clusters for a specific stream, so although CM-UPONE determines less clusters as *minRows* and *minCols* increase, the overall cost is dominated by the update of the simple α-clusters. The cost of reclustering (not shown) is significantly higher.

In the next experiment, we study the performance with respect to *minRows* and *minCols* for STOCKS data set. Fig. 12 illustrates the impact of the parameters to the update process. The sliding window size is set to 100, whereas $\alpha = 0.2$. In Fig. 12(a) $minCols = 5$ and in Fig. 12(b) $minRows = 15$. The results are similar with those of the SYNTHETIC data set.

In the sequel, we examine the relationship among the parameters *minRows*, *minCols* and α. Fig. 13 illustrates the results only for CM-UPALL. The sliding window size is set to 100. In Fig. 13(a), it is shown that the cost decreases as the parameters *minRows* and *minCols* increase their values. In Fig. 13(b), it is shown that the cost decreases as *minRows* increases and α decreases. A small α gives a large number of simple α-clusters and therefore, the probability that two streams will belong to the same

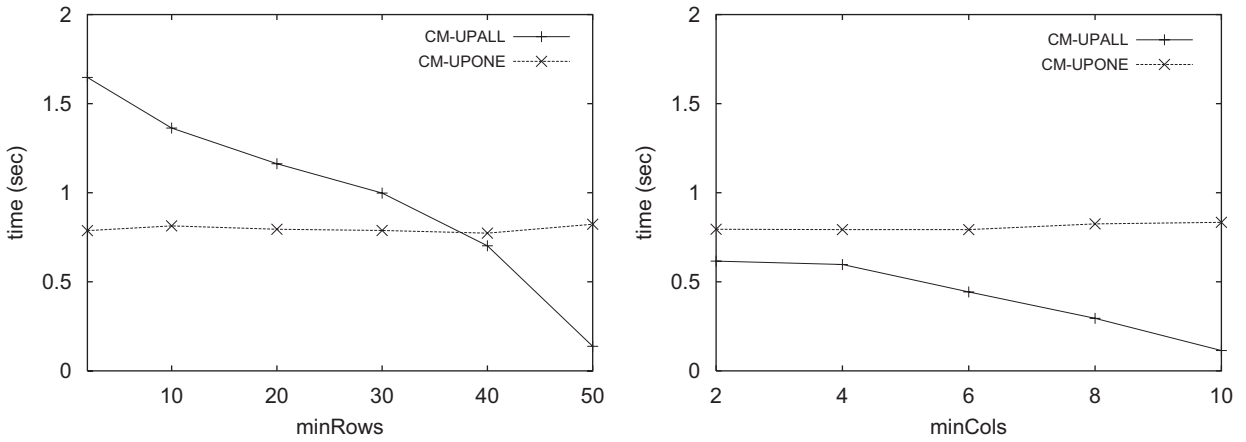

Fig. 11. Response time vs. (a) *minRows* and (b) *minCols* for SYNTH data set.

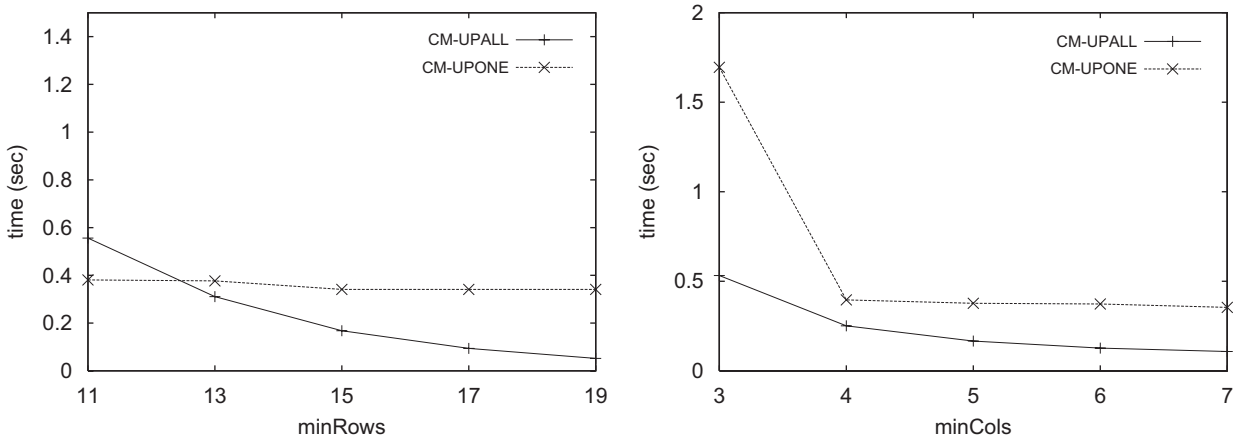

Fig. 12. Response time vs. (a) *minRows* and (b) *minCols* for STOCKS data set.

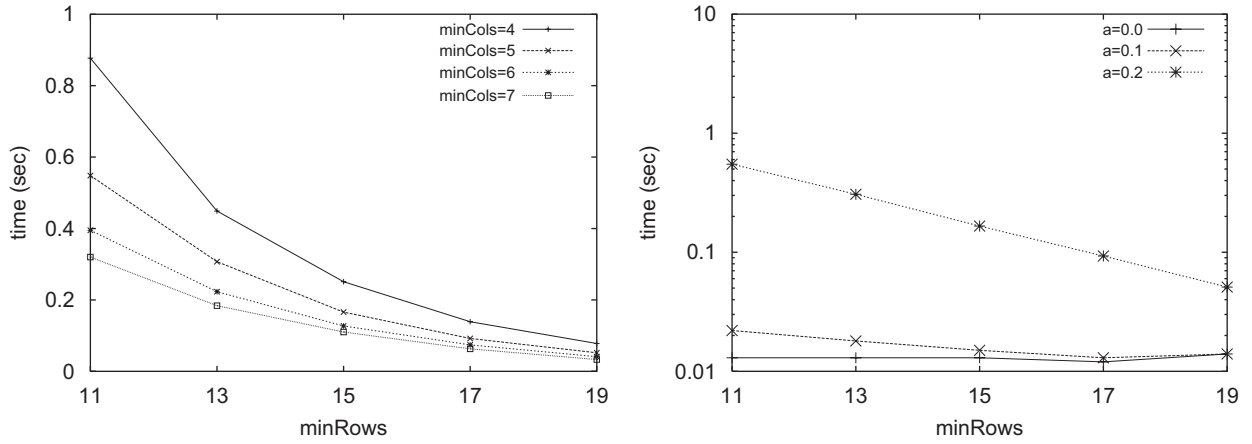

Fig. 13. Response time vs. (a) *minRows* and *minCols* (α fixed at 0.2) and (b) *minRows* and α (*minCols* fixed at 5) for STOCKS data set.

Table 3
Number of clusters and average update time for STOCKS data set

α	*minRows*	*minCols*	Number of clusters and average update time after									
			Initialization		10 updates		30 updates		50 updates		70 updates	
0.1	11	2	494	5.84	489	0.07	438	0.08	384	0.06	449	0.11
0.2	15	4	328	14.61	295	0.26	268	0.29	245	0.21	382	0.80
0.2	20	2	616	63.05	607	0.52	574	1.07	539	0.48	680	1.89
0.3	30	2	287	289.66	278	1.41	243	4.21	251	2.85	389	25.79

Table 4
Number of clusters and average update time for ECG data set

α	*minRows*	*minCols*	Nmber of clusters and average update time after									
			Initialization		5 updates		10 updates		15 updates		20 updates	
0.0	30	3	984	2695.75	980	65.08	984	68.41	979	68.15	986	68.90
0.0	150	2	76	8211.28	75	65.96	72	64.60	71	86.40	69	76.88
1.0	10	9	335	1362.11	321	73.67	314	75.27	310	76.06	311	77.31
1.0	35	5	220	9879.22	209	219.32	203	231.61	201	231.60	194	237.57

simple α-cluster is reduced. Thus, the number of maximal subspace α-clusters decreases and the overall cost is reduced.

In the next experiment, we study the number of maximal subspace α-clusters that our method reports. Some representative results are given in Tables 3 and 4 for the STOCKS and ECG data set respectively. The sliding window size is set to 100. The tables depict the number of clusters, the cost of CI phase, the number of clusters when some update operations have been performed and the average update time of CM-UPALL for these operations. In each update operation all streams are updated. By observing these tables, we can see how the clusters evolve with time. Moreover, it is evident that there are many clusters with respect to proposed α-cluster model in real data. That show the utilization of discovering α clusters in data mining applications.

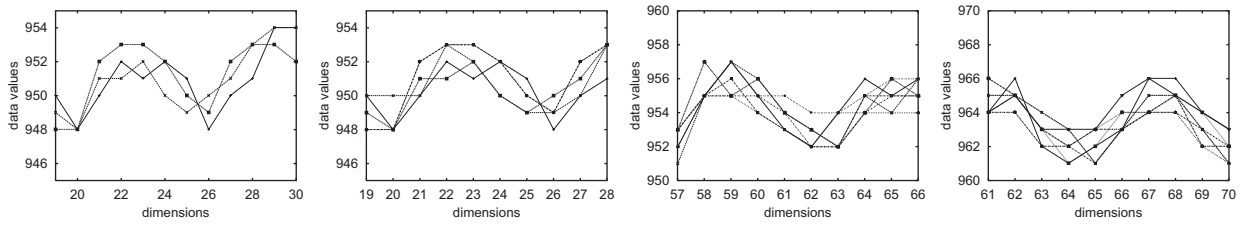

Fig. 14. Examples of maximal subspace α-clusters for ECG data set.

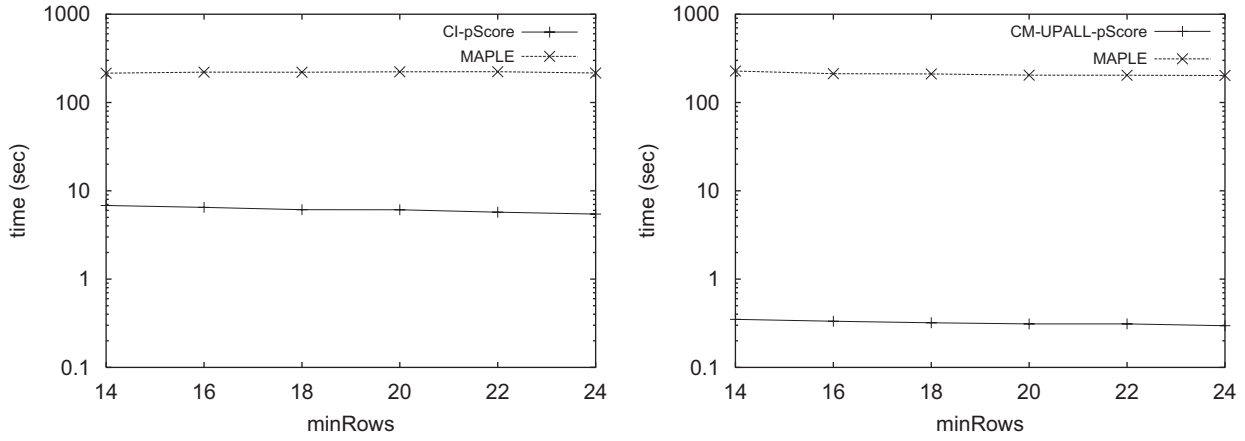

Fig. 15. Response time vs. *minRows* for TAO data set ($minCols = 4$, $W = 30$).

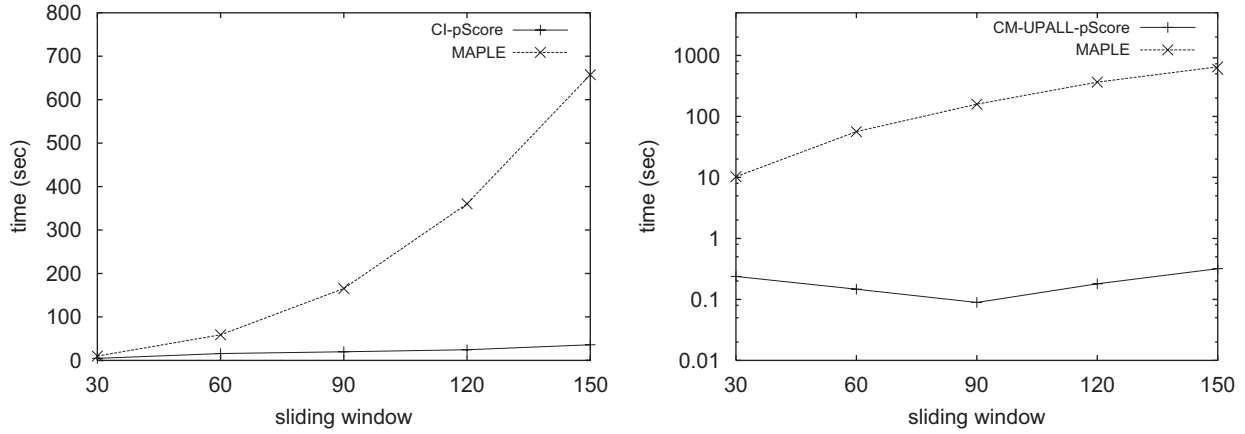

Fig. 16. Response time vs. sliding window for STOCKS data set ($minRows = 25$, $minCols = 5$).

Fig. 14 illustrates some of the clusters identified by the proposed algorithm in the ECG data set. Each figure shows an identified maximal α-cluster ($\alpha = 2$). For example, the first figure shows a maximal α-cluster with 4 streams in 7 consecutive dimensions. It is evident, that there is a high degree of similarity among streams belonging to the same cluster for the specific dimensions.

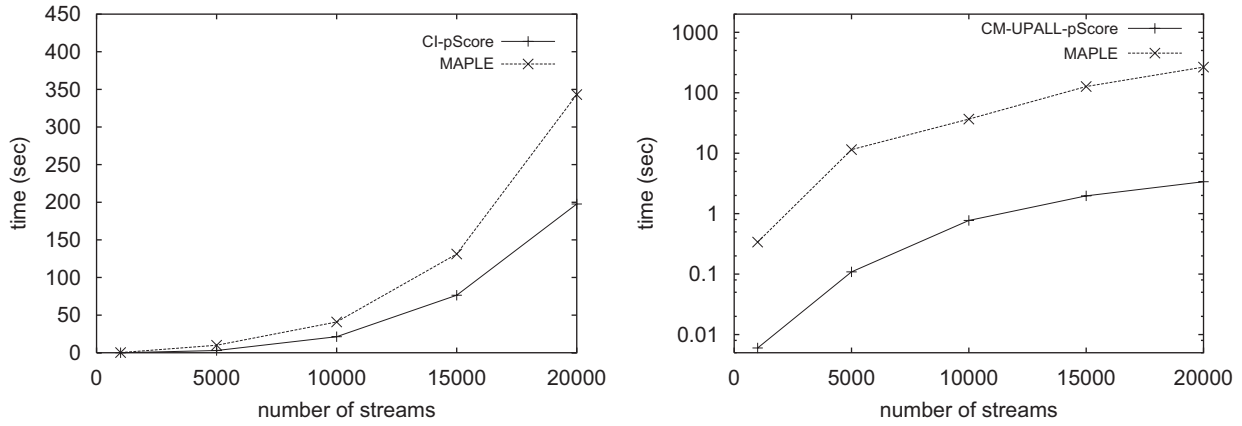

Fig. 17. Response time vs. number of streams for SYNTHETIC data set ($W = 30$, $minCols = 5$).

5.2.2. Comparison with MaPle

In this section we compare our algorithms with MaPle. For comparison reasons, we modified the MaPle algorithm to find δ-pClusters only in consecutive dimensions. Moreover, we modified the initialization and the update phase as described in Section 4, in order to discover δ-pClusters in consecutive dimensions. The abbreviations for these modified algorithms are CI-pScore and CM-UPALL-pScore.

We compared the performance of the CI-pScore and CM-UPALL-pScore algorithms with that of MaPle, by using TAO, STOCKS and SYNTHETIC data sets. The results are illustrated in Figs. 15–17. In these experiments, we used $\delta = 0$. In order to compare CI-pScore with Maple, we executed each method 10 times and we give the average execution time. In order to compare CM-UPALL-pScore with Maple, we re-apply Maple in each update and we give the average execution time of the updates. The execution time of CM-UPALL-pScore does not include the CI-pScore time, since CI-pScore is applied only once at the beginning. As expected, the cost of CM-UPALL-pScore is significantly less than that of Maple regardless the number of streams, the sliding window and the other parameters. The algorithms were tested extensively. In all cases the results are similar, so we report some representative results. Moreover, our CI-pScore algorithm is more efficient than MaPle in all conducted experiments. This motivates us to compare the CI-pScore algorithm with MaPle in static data. Fig. 18 depicts the result of this comparison on YEAST data set, by using $minRows = 30$, $minCols = 9$.

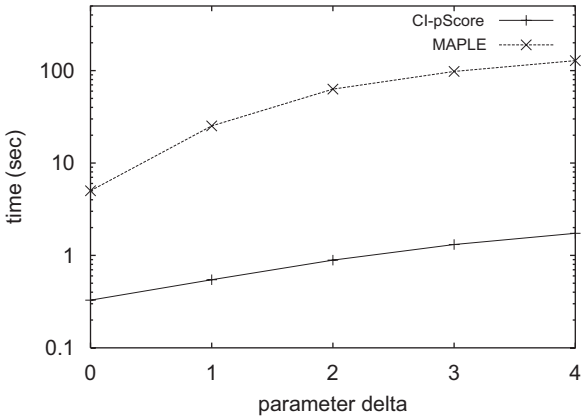

Fig. 18. CI-pScore vs. MaPle (YEAST data set).

In Fig. 19 we give examples of δ-pClusters identified in the STOCKS data set ($\delta = \alpha = 1$). By the comparison of Figs. 14 and 19, useful results are depicted. A subspace α-cluster contains streams that the similarity among them is obvious. Thus, the visualization of the subspace α-clusters, which is very important in order to have effective analysis from experts, is easy. A δ-pCluster on the other hand, can identify shifting or scaling patterns in the data set, as it has been shown in [6]. So even in the case of static data, we can use effectively and efficiently the CI-pScore algorithm if we interest to find δ-pClusters in consecutive dimensions.

In summary, the proposed incremental subspace clustering algorithms are scalable with respect to the number of streams and the sliding window size. Moreover, the cost to update the clusters by using the incremental approach is significantly less than

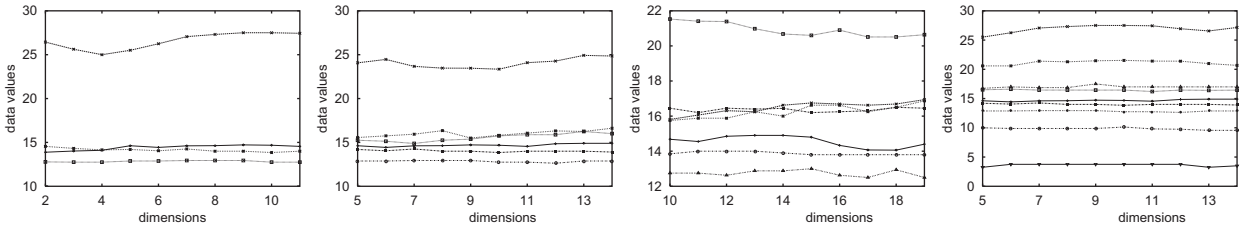

Fig. 19. Examples of δ-pClusters for STOCKS data set.

that of re-applying the initialization phase. Finally, it has been shown that the modified algorithms outperform MaPle significantly, thus they can also be used to generate clusters in static (non-evolving) data sets.

6. Conclusions

We have studied the problem of continuous subspace clustering in streaming time series data. More specifically, a novel method has been proposed toward efficient cluster generation and maintenance. Each cluster is composed of a number of streaming time series, where the pair-wise value difference inside a cluster is at most α, subject to the restrictions that the minimum number of streams is *minRows* and the minimum number of dimensions is *minCols*.

The continuous clustering method comprises a number of different phases: (a) a single initialization phase, which is responsible for the initial cluster generation, and (b) a sequence of maintenance phases, which are used to update the clustering information as time progresses. Each maintenance phase is executed when either new values for all streams are available, or only one new stream value is available in every time instance. It has been demonstrated that by using the proposed pruning criteria (cluster pruning, dimension pruning and stream pruning), significant search space reduction is achieved.

Moreover, we have shown that the proposed method is easily adapted to determine pClusters in consecutive dimensions. A performance comparison with MaPle, the state-of-the-art method for pCluster generation, has shown that the proposed method is more efficient both for time evolving time series and static data.

Appendix A. Supplementary data

Application 1.

Supplementary data associated with this article can be found in the online version, at 10.1016/j.is.2007.09.001.

References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, in: Proceedings of the ACM Symposium on Principles of Database Systems (PODS'02), 2002, pp. 1–16.

[2] S. Babu, J. Widom, Continuous queries over data streams, ACM SIGMOD Rec. 30 (3) (2001) 109–120.

[3] S. Chandrasekaran, M.J. Franklin, Streaming queries over streaming data, in: Proceedings of the International Conference on Very Large Databases (VLDB'02), 2002.

[4] J. Han, M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann, Los Altos, CA, 2000.

[5] M.H. Dunham, Data Mining: Introductory and Advanced Topics, Prentice-Hall, Englewood Cliffs, NJ, 2002.

[6] J. Pei, X. Zhang, M. Cho, H. Wang, P.S. Yu, MaPle: a fast algorithm for maximal pattern-based clustering, in: Proceedings of the IEEE International Conference on Data Mining (ICDM'03), 2003, pp. 259–266.

[7] H. Wang, W. Wang, J. Yang, P.S. Yu, Clustering by pattern similarity in large data sets, in: Proceedings of the ACM International Conference on Management of Data (SIGMOD'02), 2002, pp. 394–405.

[8] K. Beyer, J. Goldstein, R. Ramakrishnan, U. Shaft, When is nearest neighbors meaningful, in: Proceedings of the International Conference on Database Theory (ICDT'99), 1999, pp. 217–235.

[9] J. Lin, E. Keogh, W. Truppel, Clustering of streaming time series is meaningless, in: Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'03), 2003.

[10] R. Agrawal, J. Gehrke, D. Gunopoulos, P. Raghavan, Automatic subspace clustering of high dimensional data for data mining application, in: Proceedings of the ACM International Conference on Management of Data (SIGMOD'98), 1998, pp.94–105.

[11] C. Cheng, A.W. Fu, Y. Zhang, Entropy-based subspace clustering for mining numerical data, in: Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'99), 1999, pp. 84–93.

[12] K. Kailing, H.P. Kriegel, P. Kroger, Density-connected subspace clustering for high-dimensional data, in: Proceedings of SIAM International Conference on Data Mining (SDM'04), 2004, pp. 246–257.

[13] M. Ester, H.P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD'96), 1996, pp. 291–316.

[14] H.V. Jagadish, J. Madar, R.T. Ng, Semantic compression and pattern extraction with fascicles, in: Proceedings of the International Conference on Very Large Databases (VLDB'99), 1999, pp. 186–198.
[15] C.C. Aggarwal, C. Procopiuc, J.L. Wolf, P.S. Yu, J.S. Park, Fast algorithms for projected clustering, in: Proceedings of the ACM International Conference on Management of Data (SIGMOD'99), 1999, pp. 61–72.
[16] C.C. Aggarwal, P.S. Yu, Finding generalized projected clusters in high dimensional spaces, in: Proceedings of the ACM International Conference on Management of Data, 2000, pp. 70–81.
[17] J. Yang, W. Wang, H. Wang, P.S. Yu, δ-clusters: capturing subspace correlation in a large data set, in: Proceedings of the International Conference on Data Engineering (ICDE'02), 2002, pp. 517–528.
[18] Y. Cheng, G.M. Church, Biclustering of expression data, in: Proceedings of the International Conference on Intelligent Systems for Molecular Biology (ISMB'00), 2000, pp. 93–103.
[19] H. Wang, F. Chu, W. Fan, P.S. Yu, J. Pei, A fast algorithm for subspace clustering by pattern similarity, in: Proceedings of the Statistical and Scientific Database Management (SSDBM'04), 2004, pp. 51–60.
[20] C.C. Aggarwal, J. Han, J. Wang, P.S. Yu, A framework for clustering evolving data streams, in: Proceedings of the International Conference on Very Large Databases (VLDB'03), 2003, pp. 81–92.
[21] B. Babcock, M. Datar, R. Motwani, L. O'Callaghan, Maintaining variance and k-medians over data stream windows, in: Proceedings of the ACM Symposium on Principles of Database Systems (PODS'03), 2003, pp. 234–243.
[22] M. Charikar, L. O'Callaghan, R. Panigrahy, Better streaming algorithms for clustering problems, in: Proceedings of the Symposium on the Theory of Computing (STOC'03), 2003, pp. 30–39.
[23] S. Guha, A. Meyerson, N. Mishra, R. Motwani, L. O'Callaghan, Clustering data streams: theory and practice, IEEE Trans. Knowl. Data Eng. 15 (3) (2003) 515–528.
[24] C. Gupta, R. Grossman, GenIc: a single pass generalized incremental algorithm for clustering, in: Proceedings of SIAM International Conference on Data Mining (SDM'04), 2004.
[25] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, A.I. Verkamo, Fast discovery of association rules, Advances in Knowledge Discovery and Data Mining, (1996) 307–328.