Hierarchical Clustering of Time-Series Data Streams

Article *in* IEEE Transactions on Knowledge and Data Engineering · June 2008

DOI: 10.1109/TKDE.2007.190727 · Source: IEEE Xplore

Hierarchical Clustering of Time-Series Data Streams

Pedro Pereira Rodrigues, João Gama, and João Pedro Pedroso

Abstract—This paper presents and analyzes an incremental system for clustering streaming time series. The Online Divisive-Agglomerative Clustering (ODAC) system continuously maintains a tree-like hierarchy of clusters that evolves with data, using a top-down strategy. The splitting criterion is a correlation-based dissimilarity measure among time series, splitting each node by the farthest pair of streams. The system also uses a merge operator that reaggregates a previously split node in order to react to changes in the correlation structure between time series. The split and merge operators are triggered in response to changes in the diameters of existing clusters, assuming that in stationary environments, expanding the structure leads to a decrease in the diameters of the clusters. The system is designed to process thousands of data streams that flow at a high rate. The main features of the system include update time and memory consumption that do not depend on the number of examples in the stream. Moreover, the time and memory required to process an example decreases whenever the cluster structure expands. Experimental results on artificial and real data assess the processing qualities of the system, suggesting a competitive performance on clustering streaming time series, exploring also its ability to deal with concept drift.

Index Terms—Data stream analysis, clustering streaming time series, incremental hierarchical clustering, change detection.

1 Introduction

In recent days, information has grown in importance as the Web spread out, with a constant increase in communication capabilities, creating a global network interaction of both data and processes. The traditional setting for data analysis had turned gathering data into one of the most difficult tasks in data mining applications. Nowadays, we face the opposite situation. In fact, not rarely is the amount of data available from a given source (for example, sensor networks) so high that traditional batch systems, which are usually based on memory storage and multiple readings of the same data, simply cannot be used. In recent real-world applications, data flows continuously from a *data stream* at high speed, producing examples over time. Traditional models cannot adapt to the high-speed arrival of new examples [1]. This way, new algorithms have been developed, which aim at processing data in *real time*. These algorithms should be capable of processing each example in constant time and memory while consistently supplying a compact data description at each given moment [2]. In this context, quicker responses are usually requested. We need to continuously maintain a decision model at any time, which should reflect the behavior of the most recent data.

- *P.P. Rodrigues is with LIAAD-INESC Porto L.A. and the Faculty of Sciences, University of Porto, Rua de Ceuta, 118-6, 4050-190 Porto, Portugal. E-mail: pprodrigues@fc.up.pt.*
- *J. Gama is with LIAAD-INESC Porto L.A. and the Faculty of Economics, University of Porto, Rua de Ceuta, 118-6, 4050-190 Porto, Portugal. E-mail: jgama@fep.up.pt.*
- *J.P. Pedroso is with UESP-INESC Porto L.A., Faculty of Sciences, University of Porto, Rua do Campo Alegre, 1021/1055, 4169-007 Porto, Portugal. E-mail: jpp@fc.up.pt.*

Manuscript received 21 July 2006; revised 20 May 2007; accepted 6 Nov. 2007; published online 20 Nov. 2007.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0350-0706.
Digital Object Identifier no. 10.1109/TKDE.2007.190727.

Time-series data is perhaps the most common kind of data explored by data miners [3]. Clustering is probably the most frequently used data mining algorithm [4], used in exploratory data analysis. Data clustering techniques that work in real time must allow the update of the clustering definition based only on the current model and on new examples. Aside from the resource restrictions, one could be interested in the analysis of the clustering structure and clusters' evolution over time. Systems should be able to refine the cluster structure whenever more information is available and to take into account that the structure can change over time. Among different techniques known in the literature, hierarchical models are more versatile, as they do not require an a priori definition of the number of clusters. From these, divisive methods seem to be the most appropriate to apply to an online procedure, building the clustering structure by using a top-down strategy [5].

Most of the work in the incremental clustering of data streams has been concentrated on example clustering rather than variable clustering. Clustering variables (for example, time series) is a very useful tool for some applications such as sensor networks, social networks, electrical power demand, and stock market. However, the incremental clustering of variables is still not a completely covered issue. The standard approach to cluster variables in a batch scenario uses the *transpose* of the working matrix. In a data stream scenario, this is not possible, because the transpose operator is a blocking operator [2]. For the task of clustering variables in streams, new algorithms are required. To our best knowledge, this is one of the first proposals to achieve this goal.

The main objective of this work is to present an adaptive system to perform a hierarchical clustering of variables, where each variable is a time series, and each new example that is fed to the system is the value of an observation of all time series in a particular moment. The main characteristics of the system are incremental update with new examples, anytime output of the clustering structure, and the ability to detect and react to changes that may occur in it. This paper is a substantial extension of a previous one [6], published as a short paper in a data mining conference, where only a small overview of the system was presented.

In the next section, a review over an incremental clustering analysis for data streams is performed. Section 3 introduces the proposed hierarchical approach for the task of clustering streaming time series, focusing on the decisions of splitting and aggregation. In Section 4, experimental evaluation and real data application results are gathered, supporting the quality of the system. Moreover, scalability and sensitivity tests are presented to assess the robustness of the proposed method. A small discussion over the limitations of the system is presented in Section 5, and we finalize the exposition in Section 6, where concluding remarks and future work are presented.

2 Related Work

Knowledge discovery systems are usually constrained by three limited resources: time, memory, and sample size. In traditional applications, sample size limitation was proved to be dominant, since it often leads to overfitting. Nowadays, time and memory seem to be the bottleneck for machine learning applications, mainly the last one. Current data mining problems approach a new type of data sets, called *data streams*. According to Guha et al. [7], a *data stream* is an ordered sequence of points that can be read only once or a small number of times. For what is relevant in our work, we will consider reading data only once. This data model emerged from several new applications such as sensor data, Web clicks, credit card usage, and multimedia data, which required a different approach for the usual data mining problems. Incremental methods have been developed in various fields of data mining research (for example [1], [8]), which can cope with data stream analysis. Likewise, incremental clustering techniques have also emerged from research in the last years.

Clustering is usually taken as a batch procedure, statically defining the structure of objects. Many clustering techniques emerged from research, but we can group most of them in two major paradigms: *partitional* [9] and *hierarchical* [10] approaches. Nevertheless, *density-based* [11] and *grid-based* [12] systems also present promising research lines. A comprehensive overview on clustering analysis can be found in [9]. Hierarchical algorithms have a major advantage over partitional methods, as they do not require a user-predefined number of target clusters.

We should stress that our goal is to perform clustering on the variable domain (the time series) and not to cluster examples, as most of the previous work. In batch clustering, this is not a challenging issue, as *examples* and *variables* can be easily transposed, whereas in data streams, clustering the standard matrix transposition is not applicable. In fact, most of the work published in clustering of data streams refer to example clustering. This is one of the first works referring to clustering variables in the data streams framework. In the following, we present a concise review of the literature.

One of the first clustering systems to be developed, being both incremental and hierarchical, was the *COBWEB*, a conceptual clustering system that executes a hill-climbing search on the space of hierarchical categorization [13]. This method incrementally incorporates objects in a probabilistic categorization tree, where each node is a probabilistic concept representing a class of objects. The gathering of this information is made by means of the categorization process of the object down the tree, updating counts of sufficient statistics while descending the nodes and executing one of several operations: classifying an object according to an existing cluster, creating a new cluster, combining two clusters, or dividing one cluster into several ones.

2.1 Partitioning Methods

Bradley et al. [14] proposed the *Single-Pass K-Means*, an algorithm that aims at increasing the capabilities of *K-Means* for large data sets. The main idea is to use a buffer where points of the data set are kept in a compressed way. The *STREAM* [15] system can be seen as an extension of [14], which aims at minimizing the sum of the squared differences (as in *K-Means*), keeping as a restriction the use of available memory. STREAM processes data into batches of m points. These points are then stored in a buffer in the main memory. After filling the buffer, STREAM clusters the buffer into k clusters. It then summarizes the points in the buffer by retaining only the k centroids, along with the number of examples in each cluster. STREAM discards all the points, except for the centroids weighted by the number of points assigned to it. The buffer is filled in with new points, and the clustering process is repeated using all points in the buffer. This approach results in a one-pass constant-factor approximation algorithm. The main problem is that STREAM never considers data evolution. The resulting clustering can become dominated by the older outdated data of the stream. However, an interesting aspect of this algorithm is the ability to compress old information, a relevant issue in data stream processing.

Recent research developments are directed toward distributed algorithms for a continuous clustering of examples over distributed data streams. Cormode et al. [16] proposed different strategies to achieve this goal, with local and global computations, in order to balance the communication costs. They considered techniques based on the *farthest point* algorithm [17], which gives an approximation for the radius and diameter of clusters, with a guaranteed cost of two times the cost of the optimal clustering. They also present the *parallel guessing* strategy, which gives a slightly worse approximation but requires only a single pass over the data. They conclude that in actual distributed settings, it is frequently preferable to track each site locally and combine the results at the coordinator site.

2.2 Hierarchical Methods

One major achievement in this area of research was the *Balanced Iterative Reducing and Clustering using Hierarchies* (BIRCH) system [18]. The *BIRCH* system builds a hierarchical structure of data, the *CF-tree*, a balanced tree where each node is a tuple (*clustering feature*). A clustering feature contains the sufficient statistics for a given cluster: the number of points, the sum of each of the feature values, and the sum of the squares of each feature value. Each clustering feature corresponds to a cluster, which is being hierarchically organized in a *CF-tree*. Each nonleaf node in the tree aggregates the information gathered in the descendant nodes. This algorithm tries finding the best groups with respect to the available memory while minimizing the amount of input and output. The *CF-tree* grows by aggregation with only one pass over the data, thus having complexity $O(N)$. Another use of the *CF-tree* appears in [19]. More than being an algorithm, the *CluStream* is a complete system composed of two components: one *online* and another *offline*. Structures called *microclusters* are locally kept, having statistical information of data. These structures are defined as a temporal extension of the *clustering feature* vectors presented in [18], which are being kept as images through time, following a pyramidal form. This information is used by the offline component that depends on a variety of user-defined parameters to perform final clustering by an iterative procedure.

The *Clustering Using Representatives* (*CURE*) system [20] performs a hierarchical procedure that assumes an intermediate approach between centroid-based and all-point-based techniques. In this method, each cluster is represented by a constant number of points well distributed within the cluster, which capture the extension and shape of the cluster. This process allows the identification of clusters with arbitrary shapes. The *CURE* system also differs from *BIRCH* in the sense that instead of preaggregating all the points, this system gathers a random sample of the data set, using Chernoff bounds in order to obtain the minimum number of examples.

2.3 Concept Change Detection

Most of the work in machine learning assumes that examples are generated at random according to some stationary probability distribution [21]. There are already several methods in predictive machine learning to deal with changing concepts (for example, [21], [22]). Nevertheless, the notion of a concept drift applied to clustering analysis is not directly derived from the concept drift on the variable domain, as the clustering structure may not be affected by a variable's dynamics. Detecting a concept drift as usually conceived for one time-varying/order-varying variable is not the same as detecting the concept drift on the clustering structure of several time-varying/order-varying variables. These are usually points in the stream of data, where the clustering structure gathered with the previous data is no longer valid, since it no longer represents the new relations of dissimilarity between the streams. In this work, we also try addressing this feature.

3 Online Divisive-Agglomerative Clustering

The task of clustering time series over data streams is not widely studied. In fact, to the best of our knowledge, this is the first proposal of a hierarchical approach to the problem, so we should start by formally introducing it. Afterwards, a complete description of the proposed system is presented, trying to address its main characteristics and innovations.

3.1 Clustering Streaming Time Series

Data streams usually consist of variables producing examples continuously over time. The basic idea behind clustering streaming time series is to find groups of variables that behave similarly through time. Let $X = \langle x_1, x_2, \ldots, x_n \rangle$ be the complete set of n data streams and $X^t = \langle x_1^t, x_2^t, \ldots, x_n^t \rangle$ be the example containing the observations of all streams x_i at a specific time t. The goal of a clustering system for multiple time series is to find (and make available at any time t) a partition P of streams, where streams in the same cluster tend to be more alike than streams in different clusters. In partitional clustering, searching for k clusters, the result at time t should be a matrix P of $n \times k$ values, where each P_{ij} is one if stream x_i belongs to cluster c_j; otherwise, it is zero. Specifically, we can inspect the partition of streams in a particular time window from a starting time s until the current time t by using examples $X^{s,\ldots,t}$, which would give a temporal characteristic to the partition. In a hierarchical approach to the problem, the same possibilities apply, with the benefit of not having to previously define the target number of clusters, thus creating a structured output of the hierarchy of clusters. An example partition could be defined as $P^t = \{\{\{x_1\}, \{x_3, x_5\}\}, \{x_2, x_4\}\}$, stating that data streams x_1, x_3, and x_5 have some similarity between them (more pronounced between x_3 and x_5), being at the same time somehow dissimilar from x_2 and x_4.

3.2 A Hierarchical Approach

In this paper, the *Online Divisive-Agglomerative Clustering* (ODAC) system is presented, which is an algorithm for an incremental clustering of streaming time series that constructs a hierarchical tree-shaped structure of clusters by using a top-down strategy. The leaves are the resulting clusters, with each leaf grouping a set of variables. The union of all leaves is the complete set of variables. The intersection of any two leaves is the empty set. The system encloses an incremental distance measure and executes procedures for expansion and aggregation of the tree-based structure based on the diameters of the clusters.

The main assumption of the system is that decisions taken over a sample of the most recent data are, in the limit and under certain conditions, equivalent to those taken over an infinite set of observations. Given this, the system continuously monitors existing clusters' diameters over time. The *diameter* of a cluster is the maximum distance between variables of that cluster. For each existing cluster, the system finds the two variables defining the diameter of that cluster. At time t, if a given condition is met on this diameter, the system splits the cluster and assigns each of the chosen variables to one of the new clusters, becoming the *pivot* variable for that cluster. Afterwards, all remaining variables on the old cluster are assigned to the new cluster, which has the closest pivot. New leaves start new statistics, assuming that only forthcoming information will be useful for deciding whether or not this cluster should be split. Each node c_k will then represent relations between streams using examples $X^{i_k \ldots s_k}$, where i_k is the time at which the node was created, and s_k is the time at which the node was split (or the current time t for leaf nodes). This feature increases the system's ability to cope with changing concepts, as later on, a test is performed to check if the previously decided split still represents the structure of variables. On stationary data streams, the overall intracluster dissimilarity should decrease with each split. This way, if a cluster is split into two child leaves, the diameter of the new clusters should be less than or equal to the diameter of the parent node. If the diameter of a leaf is greater than its parent's diameter, then the previously taken decision no longer reflects the structure of data. The system reaggregates on the cluster's parent, restarting statistics. The forthcoming sections describe the inner core of the system.

3.3 Incremental Dissimilarity Measure

The system must analyze distances between incomplete vectors, possibly without having any of the previous values available. Thus, these distances must be incrementally computed. Since we want to make decisions with statistical support, we will use the Hoeffding bound to support our decisions, forcing the criterion, that is, the distance measure, to be scaled [23]. We use Pearson's correlation coefficient [24] between time series as the *similarity* measure, as done in [25]. Deriving from the correlation between two time series a and b calculated in [26], the factors used for computing the correlation can be updated incrementally, achieving an exact incremental expression for the correlation:

$$corr(a,b) = \frac{P - \frac{AB}{n}}{\sqrt{A_2 - \frac{A^2}{n}}\sqrt{B_2 - \frac{B^2}{n}}}. \quad (1)$$

The *sufficient statistics* needed to compute the correlation are easily updated at each time step: $A = \sum a_i$, $B = \sum b_i$, $A_2 = \sum a_i^2$, $B_2 = \sum b_i^2$, and $P = \sum a_i b_i$. In ODAC, the dissimilarity between variables a and b is measured by an appropriate metric, the *Rooted Normalized One-Minus-Correlation*, given by

$$rnomc(a,b) = \sqrt{\frac{1 - corr(a,b)}{2}}, \quad (2)$$

with range [0, 1]. We consider the cluster's *diameter* to be the highest dissimilarity between two time series belonging to the same cluster or the variable variance in the case of clusters with single variables.

3.4 Growing the Hierarchy

The main procedure of the ODAC system is to grow a tree-shaped structure that represents the hierarchy of the clusters present in the data. In this system, each example is processed only once. The system incrementally updates, at each new example arrival, the sufficient statistics needed to compute the dissimilarity matrix, enabling its application to clustering data streams. The dissimilarity matrix for each leaf is only computed when it is being tested for splitting or aggregation after receiving a minimum number of examples. When processing a new example, only the leaves are updated, avoiding the computation of unneeded dissimilarities, and this speeds up the process every time the structure grows.

3.4.1 Splitting Criteria

One problem that usually arises with this sort of models is the definition of a minimum number of observations necessary to assure convergence. A common way of doing this includes a user-defined parameter. After a leaf has received at least n_{min} examples, it is considered ready to be tested for splitting. Another approach is to apply techniques based on the Hoeffding bound [23] to solve this problem. The Hoeffding bound has the advantage of being independent of the probability distribution generating the observations [1], stating that after n independent observations of a real-valued random variable r with range R, with confidence $1 - \delta$, the true mean of r is at least $\overline{r} - \epsilon$, where $\overline{r}$ is the observed mean of the samples, and

$$\epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}}. \quad (3)$$

As each leaf is fed with a different number of examples, each cluster c_k will possess a different value for ϵ, designated ϵ_k. Let $d(a,b)$ be the distance measure between pairs of time series and $D_k = \{(x_i, x_j) \mid x_i, x_j \in c_k, i < j\}$ be the set of pairs of variables included in a specific leaf c_k. After seeing n samples at the leaf, let $(x_1, y_1) \in \{(x,y) \in D_k \mid d(x,y) \geq d(a,b), \forall (a,b) \in D_k\}$ be the pair of variables with maximum dissimilarity within the cluster c_k. In the same way, considering $D'_k = D_k \backslash \{(x_1, y_1)\}$, let

$$(x_2, y_2) \in \{(x,y) \in D'_k \mid d(x,y) \geq d(a,b), \forall (a,b) \in D'_k\}.$$

Let $d_1 = d(x_1, y_1)$, $d_2 = d(x_2, y_2)$, and $\Delta d = d_1 - d_2$ be a new random variable, consisting of the difference between the observed values through time. Applying the Hoeffding bound to Δd, if $\Delta d > \epsilon_k$, we can confidently say that with probability $1 - \delta$, the difference between d_1 and d_2 is larger than zero, hence selecting (x_1, y_1) as the pair of variables representing the diameter of the cluster. That is

$$d_1 - d_2 > \epsilon_k \Rightarrow diam(c_k) = d_1. \quad (4)$$

With this rule, the ODAC system will only split the cluster when the true diameter of the cluster is known, with the statistical confidence given by the Hoeffding bound. This rule triggers the moment when the leaf has been fed with enough examples to support the decision. Although a time series is not a purely random variable, we have decided to model the time-series first-order differences in order to reduce the negative effect of autocorrelation on the Hoeffding bound. Moreover, with this approach, the missing values can be easily treated with a zero value, considering that when unknown, the time series is constant.

3.4.2 Resolving Ties

The rule presented in (4) redirects the research to a different problem. There might be cases where the two topmost distances are nearly or completely equal. To distinguish the cases where the cluster has many variables nearly equidistant from the cases where there are two or more highly dissimilar variables, a tweak must be done. Having in mind the application of the system to a data stream with high dimension, possibly with hundreds or thousands of variables, we turn to a heuristic approach. Based on the techniques presented in [1] and [21], we introduce a parameter to the system τ, which determines how long we will let the system check for the real diameter until we force the splitting and aggregation tests. At any time, if $\tau > \epsilon_k$, the system overrules the criterion of (4), assuming that the leaf has been fed with enough examples; hence, it should consider the highest distance to be the real diameter.

3.4.3 Controlling the Growth

To prevent the hierarchy from growing unnecessarily, we define another criterion that has to be fulfilled to perform the splitting. The splitting criterion should reflect some relation among the distances between variables of the cluster. Given this fact, we can impose a cluster to be split if it includes a high difference between $(d_1 - \overline{d})$ and $(\overline{d} - d_0)$, where d_0 stands for the minimum distance between variables belonging to the cluster, and $\overline{d}$ is the average of all distances in the cluster. In our approach, we relate the expression with the global difference $d_1 - d_0$. Our heuristic is that for a given cluster c_k, we choose to split this leaf into a node with two child leaves if the following condition is met:

$$(d_1 - d_0)\left|d_1 + d_0 - 2\overline{d}\right| > \epsilon_k. \tag{5}$$

This expression gives the global positioning of the mean with respect to the range of the existing distances, representing two inherent concepts: the farther the highest distance from the minimum distance, the higher the possibility of a split occurrence, and the farther the mean distance from the average of the maximum and minimum distances $(d_1 + d_0)/2$, the higher the probability of splitting.

3.4.4 Expanding the Tree

When a split point is reported, the *pivots* are variables x_1 and y_1, where $d_1 = d(x_1, y_1)$. The system assigns each of the remaining variables of the old cluster to the cluster that has the closest pivot. The sufficient statistics of each new cluster are initialized. The total space required by the two new clusters is always less than the one required by the previous cluster. Algorithm 1 sketches the splitting procedure.

Algorithm 1 TestSplit
Input: a cluster c_k
Output: a Boolean value stating if cluster c_k was split or not

```
{Tests the cluster for splitting}
1: let $d_1$, $d_2$, $d_0$, and $\overline{d}$ be the distances previously defined
   for (4) and (5);
2: if $d_1 - d_2 > \epsilon_k$ or $\tau > \epsilon_k$ then
3:    if $(d_1 - d_0)|(d_1 - \overline{d}) - (\overline{d} - d_0)| > \epsilon_k$ then
4:       create $c_x$ and $c_y$, with $x_1$ and $y_1$ being the pivots,
         that is, $x_1 \in c_x \wedge y_1 \in c_y$;
5:       for each remaining variable $x_i \in c_k$ not yet assigned
         do
6:          if $d(x_i, x_1) \leq d(x_i, y_1)$ then $x_i \in c_x$;
7:          else $x_i \in c_y$;
8:       end for
9:       return True;
10:   end if
11: end if
12: return False;
```

3.5 Aggregating at Concept Drift Detection

The main setting of our system is the monitoring of existing clusters' diameters. In the development of a hierarchical structure of clusters, the overall intracluster dissimilarity should decrease with each split. This should be observed on stationary data. On stationary data streams, the diameter of a cluster decreases every time a split occurs. However, usual real-world problems deal with nonstationary data streams, where time series that were correlated in the past are no longer correlated to each other in the current time period. They may also be approaching time series of other clusters. The main problem here is to distinguish between scenarios where a leaf grows into a more elaborated structure or a change in the relevance of previous splits is found.

The strategy that is adopted in this work is based on the analysis of the diameters. This way, no computation is needed between the variables of different nodes.[1] If the correlation structure between time series is stationary, a cluster split will never increase the cluster's diameter. For each given leaf c_k, we should search to see if the split decision that created it still represents the structure of data. Thus, we shall test the diameters of c_k and c_k's parent c_j, assuming that the child's diameter should not be larger than the diameter of the parent node. We define a new random variable $\Delta a = diam(c_k) - diam(c_j)$. Applying the Hoeffding bound to this random variable, if $\Delta a > \epsilon$, then the diameter of the child node is confidently larger than the parent's diameter. Given this, we choose to aggregate on c_j if

$$diam(c_k) - diam(c_j) > \epsilon_{jk}, \tag{6}$$

where $\epsilon_{jk} = \max(\epsilon_j, \epsilon_k)$, supporting the decision with the bound that was defined with less data. The system decreases the number of clusters, as the previous division is no longer supported, which means that it may not reflect the best divisive structure for recent data. The resulting leaf starts new computations, and a concept drift is detected. Algorithm 2 introduces the algorithm. Fig. 4 illustrates the evolution of a cluster structure in time-changing data. The complete experience is presented in Section 4.3.

Algorithm 2 TestAggregate
Input: a cluster c_k
Output: a Boolean value stating if cluster c_k was aggregated or not

```
{Tests the cluster for aggregation}
1: update dissimilarities of $c_k$, if needed, and
   $\epsilon_{jk} = \max(\epsilon_j, \epsilon_k)$
2: if $diam(c_k) - diam(c_j) > \epsilon_{jk}$ then
3:    cut $c_j$'s subtree, turning $c_j$ into a leaf with reset
      sufficient statistics;
4:    return True;
5: end if
6: return False;
```

3.6 Memory Usage and Time Complexity

The ODAC system presents the required features of an adaptive learning system. For each leaf in which the

1. Note that only the leaves are updated with new examples. Each node is associated with a specific time window representing the state of the time series in that period.

diameter is known (with the confidence level given by the Hoeffding bound), the system tests for aggregation first, so in case of a concept drift, it will not start growing unnecessarily. Algorithm 3 presents our method, merging the splitting with the aggregative procedure.

```
Algorithm 3 ODAC
Input: a set of streaming time series $X = \langle x_1, x_2, \dots, x_n \rangle$
Output: a hierarchical clustering structure $S$ with leaves
    (clusters) $L = \langle l_1, l_2, \dots, l_m \rangle$
1: repeat
2:   read the new example $X^t$ and update sufficient
     statistics on the leaves $L$;
3:   for each leaf $l_k$ not yet tested do
4:     update dissimilarities and the Hoeffding bound $\epsilon_k$
       for this leaf;
5:     if TestAggregate ($l_k$) or TestSplit ($l_k$) then
6:       announce the new structure $S$;
7:     end if
8:   end for
9: until EOF
```

Complexity analysis can be done with respect to memory usage and time consumption. In both, our system has some interesting features. A system that aims at efficiently clustering data streams must comply with constant memory usage [2]. In ODAC, the size needed to keep the sufficient statistics at each node with n variables is $O(n^2)$. Let us consider splitting this node into two new leaves, with n_1 and n_2 streams being assigned to each of them, respectively, where $n = n_1 + n_2$. Considering that $(n_1 + n_2)^2 > n_1^2 + n_2^2$, $\forall n_1, n_2 > 0$, although the space used by children nodes is still $O(n_1^2 + n_2^2) \in O(n^2)$, the reduction in the number of sufficient statistics is $O(n_1 n_2)$, with $n_1 n_2 \in [n-1, (n/2)^2]$. The system's space complexity is quadratic on the number of variables but is constant on the number of examples. This allows the handling of an infinite amount of examples, usually present in data streams. Hence, it is usable in the data streams paradigm, especially considering that the number of variables is constant.

A system that aims at efficiently clustering data streams must also comply with a constant execution time with respect to the number of examples [2]. When updating the system with a new example, the update of sufficient statistics results in $O(n^2)$ operations. This is, as expected, quadratic in the number of variables. The update does not depend on the number of examples seen. Therefore, the system's update time is constant with respect to the number of examples, satisfying the data stream requirements. The splitting procedure (Algorithm 1) needs to compute the two maximum, the minimum, and the mean of the dissimilarity matrix. This procedure is linear to the number of distances, hence $O(n^2)$, quadratic with the number of variables. When computing dissimilarities, assuming the worst-case scenario, where only one leaf exists, the number of dissimilarities computed is also $O(n^2)$ and is thus quadratic with respect to the number of variables. An important feature of this algorithm is that every time a split is performed on a leaf with n variables, the global number of dissimilarities needed to be computed at the next iteration diminishes at least $n-1$ (worst-case scenario) and at most $(n/2)^2$ (best-case scenario). As a result, the time complexity of each iteration of the system is constant with respect to the number of examples and decreases with every split. It is therefore capable of addressing continuous data streams.

4 Experimental Evaluation

ODAC is an online clustering system for time-series data streams. Since the scope of the system is very well defined, the experimental evaluation is performed, aiming at the verification of specific hypotheses. First, if the system is applied to a set of time series with stationary clustering structure, the system should converge to the real structure of the streams. However, if the streams present dynamic behavior, then the system should detect changes in the clustering structure and adapt it accordingly. Finally, we should evaluate how the system performs on real data produced by applications that generate time-series data streams and how the system scales to the high number of time series usually produced in these contexts. Moreover, sensitivity tests should be performed in order to assess the sensitivity of the system to slight changes in parameters.

4.1 Evaluation Criteria for Clustering Validity

Usually, the criteria used to evaluate clustering methods concentrate on the quality of the resulting clusters or their fitness to a given known structure. Given the hierarchical characteristic of our system, we are also interested in assessing the quality of the hierarchy constructed by the algorithm.

We can generally find three approaches to investigate clusters' quality, known as *external*, *internal*, and *relative* criteria [4]. *External* criteria make an evaluation based on a prespecified structure, which reflects our prior knowledge or intuition over the data set. Usually used *external criteria* include the *Folkes and Mallows Index* [27] and *Hubert's* Γ *Statistic* [28]. An *internal* criterion usually stands for an evaluation based on measures involving the data members themselves (for example, the *Divisive Coefficient* [10] or the *Cophenetic Correlation Coefficient (CPCC)* [9]). The third approach compares the resulting cluster structure with other clustering results for the same data set. Examples of *relative* indices are *Dunn's Index* [29] and the *Modified Hubert's* Γ *Index* [4]. The results used to make this comparison may be gathered using the same algorithm with different specifications or using different algorithms.

For the set of experiments, we will focus on two *relative* criteria to find the best number of clusters and assess the quality of the resulting clusters and an *internal* criterion to assess the hierarchical quality of the resulting clustering structure.

4.1.1 Cluster Quality

To validate our system, we consider the *Modified Hubert's* Γ, that is, $MH\Gamma$, *Statistic* and *Dunn's Validity Index* (*DVI*) in order to assess the quality of the resulting clusters. We have decided to use both, as they represent a good complement of each other: the first one is specially motivated for detecting compact and well-separated clusters but has the drawback of being dependent on the number of clusters, and the second criterion does not depend on the number of clusters but is more unstable, as it is based on the single linkage distances and diameters. The modified Hubert's Γ index is given by

$$MH\Gamma = \frac{1}{M}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} P_{ij}Q_{ij}, \quad (7)$$

where $M = \frac{n(n-1)}{2}$, P is the proximity matrix, and Q is an $N \times N$ matrix, where each Q_{ij} is the distance between the representative points (centroids, medoids, etc.) of the clusters to which i and j belong. High values of this index represent compact and well-separated clusters. The DVI is given by

$$DVI = \min_{i,j}\left\{\frac{d(c_i, c_j)}{\max\limits_{k}\{diam(c_k)\}}\right\}, \quad (8)$$

where $d(c_i, c_j)$ is the single linkage dissimilarity function between two clusters, and $diam(c_k)$ is the diameter of cluster c_k. High values of this index also represent compact and well-separated clusters. The $MH\Gamma$ measure increases with the number of clusters, so the quest for the best value is made by looking at the plot of the measure along a different number of clusters and finding the *knee* in the plot (the number of clusters that produces the highest increase $\Delta MH\Gamma$ in the quality measure). The second index is independent of the number of clusters. This way, the highest value is considered the best one.

4.1.2 Hierarchy Quality

Hierarchical clustering methods such as the ODAC system allow the analysis of another quality measure, the *CPCC*, which measures quality in hierarchical structures. The *CPCC* is defined as

$$CPCC = \frac{\sum\limits_{i=1}^{N-1}\sum\limits_{j=1+1}^{N} C_{ij}P_{ij} - \mu_P\mu_C}{\sqrt{\sum\limits_{i=1}^{N-1}\sum\limits_{j=1+1}^{N} P_{ij}^{\,2} - \mu_P^{\,2}\sum\limits_{i=1}^{N-1}\sum\limits_{j=1+1}^{N} C_{ij}^{\,2} - \mu_C^{\,2}}}, \quad (9)$$

where C is the *Cophenetic Proximity Matrix*, with each c_{ij} being the proximity level at which the two objects i and j appeared together in the same cluster for the first time and

$$\mu_P = \frac{1}{M}\sum_{i=1}^{N-1}\sum_{j=1+1}^{N} P_{ij} \quad and \quad \mu_C = \frac{1}{M}\sum_{i=1}^{N-1}\sum_{j=1+1}^{N} C_{ij}. \quad (10)$$

The closer the value of this index is to 1, the better the match, and the better the hierarchy fits the data. High values of any of these indices indicate the presence of a good clustering structure.

4.2 System Evaluation on Stationary Environments

As few data streams are really available for free use, the first evaluation of the proposed system is made using synthetic data sets, created with specific characteristics, as described in Section 4.2.1. The system is then tested in real data, that is, the PDMC Sensor Data Set.

For each data set (with n variables), 10 runs of *K-Means* (R [30] implementation) are executed, each one with all the possible numbers of clusters k. Quality measures are calculated in these runs, and an average is considered to find the best number of clusters for the data set. Afterwards, ODAC is applied in the same data set, enabling a comparison between the final structure and the one provided by *K-Means*. This procedure is performed for artificial and real data sets. Comparison with a batch DIANA [10] system, using the same correlation-based distance measure, is also presented. On all experiments, the δ parameter of the Hoeffding bound is set to 0.05, determining a confidence level of 95 percent. The τ parameter of the ODAC system is set to 0.02. The output is presented in graphics, where squared objects are used for leaves, and round objects are for nodes. The information inside each node represents the node number, the diameter for that cluster, and the number of examples seen by that node. For leaves, the variables included in the cluster are also displayed.

4.2.1 Evaluation on Artificial Data

The data sets used in this section were created using a time-series generator that produces n time series belonging to a predefined number k of clusters with a noise constant β. Each cluster c_k has a pivot time series p, and the remaining time series are created as $p + \lambda$, where

$$\lambda \sim U(-\beta p, \beta p). \quad (11)$$

We have created three data sets, with 10 variables each, specially prepared to test different hypotheses: a *closed* data set, where all 10 variables are created by the same concept, the *two-cluster* data set, where two well-defined clusters are created, with five variables each, and the *4C* data set, which represents a more complex yet stationary structure of clusters. External criteria results are not shown here, as all runs resulted in the expected clustering structure.

Closed Data Set. A hierarchical clustering procedure should always create a hierarchical structure of clusters. Nevertheless, if a data set represents a closed cluster, with all variables highly correlated, then we expect a single cluster as the system output. We have created a set of 10 time series with 100,000 examples and tested different values for the parameter β of the artificial data generator. The final result is a single cluster. Although for $\beta \geq 0.3$, some splitting occurred, this was quickly reversed by aggregation. Only for values of $\beta \geq 0.9$ did we observe spoiled results, as the cluster spread out, growing an unstable hierarchy of clusters. No meaning can be extracted from the quality measures. Based on these results, we have decided to set $\beta = 0.3$ for the following experiences.

Two-Cluster Data Set. In order to study the splitting capabilities of ODAC in the presence of a simple cut point, we have built a data set with 10 time series divided into two clusters:

$$\{\{a1, a2, a3, a4, a5\}, \{a6, a7, a8, a9, a10\}\}.$$

The results are presented in Table 1, where the *nc* column presents the number of clusters with which the results are gathered. In the ODAC system, the correct outcome is achieved immediately. The *DVI* index on *K-Means* marks the best (and real) number of clusters (and the real cluster

TABLE 1
K-Means and ODAC Quality Results for the *Two-Cluster* (Top) and *Four-Cluster* (Bottom) Data Sets

Two Clusters

System	*nc*	MHΓ	ΔMHΓ	DVI	CPCC
K-Means	**2**	0.278	–	**1.934**	–
	3	0.286	0.008	0.728	–
	4	0.294	0.008	0.732	–
	5	0.298	0.004	0.730	–
ODAC	**2**	0.166	–	1.936	**0.998**

Four Clusters (4C)

System	*nc*	MHΓ	ΔMHΓ	DVI	CPCC
K-Means	2	0.178	–	0.995	–
	3	0.366	0.188	1.000	–
	4	0.388	0.022	**1.937**	–
	5	0.392	0.004	0.734	–
ODAC	**4**	0.332	–	1.937	**0.651**

structure). The *CPCC* value for the ODAC result is almost perfect.

Four-Cluster (4C) Data Set. For an evaluation on a more complex structure of clusters, we have created a data set with 100,000 observations of 10 variables, with the inner cluster noise $\beta = 0.3$ and configuration:

$$\{\{a1, a2\}, \{a3, a4, a5\}, \{a6\}, \{a7, a8, a9, a10\}\}.$$

The aim is to check for a good partitioning in order to find the best hierarchy. The *DVI* values on the *K-Means* (Table 1) mark the true number of clusters, also found by ODAC. Interestingly, the *CPCC* gave a low value, which might be explained by the known feature of the measure lightly favoring the balanced hierarchies. The final structure here is not balanced, as can be seen in the final result presented in Fig. 1.

4.2.2 Evaluation on Real-World Data

The Physiological Data Modeling Contest Workshop (PDMC), held at the 21st International Conference on Machine Learning (ICML '04), was aimed at information extraction from streaming sensors data. The training data set for the competition[2] consists of approximately 10,000 hours of this data, containing several variables: userID, sessionID, sessionTime, characteristic[1..2], annotation, gender, and $\mathrm{sensor}[1, \ldots, 9]$. We have concentrated on sensors 2 to 9, since we were interested in finding the relations between different sensor data. Data was extracted by userID, resulting in several data sets of eight continuous variables.

Comparing Users. In this comparison, we have tried finding the right number of clusters for each user with more than 50,000 examples, which is possibly the same for all. We should note that the best result is defined differently for each of the quality measures. For the $MH\Gamma$ statistic, we must seek for the *knee* in the plot, as this measure increases with the number of clusters, whereas for the *DVI* measure, the maximum value is used. Table 2 presents the *K-Means* evaluation for the PDMC data sets, with $\mathrm{userID} = 1$, and $\mathrm{userID} = 25$ (80,182 and 14,1251 observations). Apparently, from a conservative point of view, the best number of clusters is 3. Fig. 2 sketches the ODAC system's final structure for the same data sets. The system achieved the quality level presented in the bottom line of Table 2. The same structure is found, agreeing with the expected number of clusters given by the *K-Means* experience. For the data set with $\mathrm{userID} = 25$, again, the value 3 for the number of clusters appears. This time, the structure found by ODAC is a bit different but also represents three clusters.

2. Available at http://www.bodymedia.com/support/TrainingSet.zip.

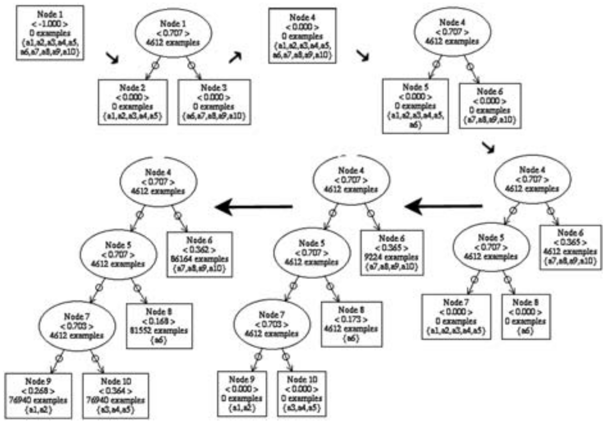


Fig. 1. ODAC system growth and final structure (the *4C* data set).

Comparing ODAC and DIANA. We have tried finding some cluster structure on the data variables to compare it with a batch divisive analysis system by using the same dissimilarity measure for each user. For $\mathrm{userID} = 1$, 93,344 examples were processed. Our system evolved splitting and aggregating until a stable structure was gathered after 55,000 examples, never changing from then on. Fig. 3 shows the comparison with a batch DIANA system by using the same correlation-based distance measure. The final result is approximately equivalent to the structure achieved by the batch system using the same dissimilarity measure. For other userIDs, the results were very similar.

4.3 System Evaluation on Dynamic Environments

For a more complex evaluation of the system's dynamics (splitting and aggregation), we have defined experiments to

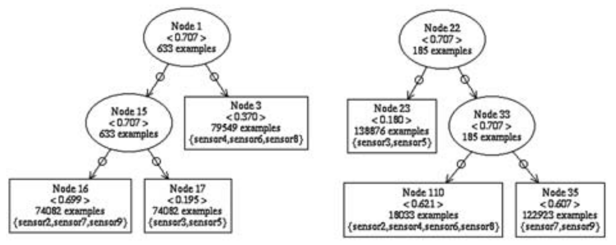
TABLE 2
K-Means and ODAC Quality Results for the PDMC Data Sets for Users 6 (Top) and 25 (Bottom)

UserID = 6

System	*nc*	MHΓ	ΔMHΓ	DVI	CPCC
K-Means	2	0.141	–	0.300	–
	3	**0.308**	**0.167**	**0.300**	–
	4	0.311	0.003	0.198	–
	5	0.393	0.082	0.300	–
	6	0.395	0.002	0.156	–
ODAC	**3**	0.377	–	0.891	0.251

UserID = 25

System	*nc*	MHΓ	ΔMHΓ	DVI	CPCC
K-Means	2	0.099	–	0.272	–
	3	**0.360**	**0.261**	**0.325**	–
	4	0.362	0.002	0.242	–
	5	0.380	0.018	0.242	–
	6	0.384	0.004	0.228	–
ODAC	**3**	0.191	–	1.026	0.479


Node 1 < 0.707 > 633 examples
Node 15 < 0.707 > 633 examples
Node 3 < 0.370 > 79549 examples {sensor4,sensor6,sensor8}
Node 16 < 0.699 > 74082 examples {sensor2,sensor7,sensor9}
Node 17 < 0.195 > 74082 examples {sensor3,sensor5}
Node 22 < 0.707 > 185 examples
Node 23 < 0.180 > 138876 examples {sensor3,sensor5}
Node 33 < 0.707 > 185 examples
Node 110 < 0.621 > 18033 examples {sensor2,sensor4,sensor6,sensor8}
Node 35 < 0.607 > 122923 examples {sensor7,sensor9}

(a) (b)

Fig. 2. ODAC final structure gathered with the physiological data. (a) $\mathrm{userID} = 6$. (b) $\mathrm{userID} = 25$.

determine if the system is able to detect and adapt to changes in the clustering structure. We have generated a data set by concatenating examples from two known structures, hence simulating a concept change. The data set is equivalent to $4C$ in the first 50,000 examples, and the last 50,000 examples were generated from a different concept. The concepts used in the drifting data set shifted from

$$\{\{a1, a2\}, \{a3, a4, a5\}, \{a6\}, \{a7, a8, a9, a10\}\}$$

to

$$\{\{a1, a2, a3, a4, a10\}, \{a5, a6, a7\}, \{a8, a9\}\}.$$

The evaluation criteria are the definition of the clustering structure, the ability to detect change, the delay in detection, and the ability to react and to converge to the new concept.

The evolution of the clustering structure is sketched in Fig. 4. The figure presents only the structures defined after an event (split or merge) has occurred. The figure contains also the number of examples read at the time of each snapshot and the time elapsed since the concept drift. The system converged at the first concept's structure in 18,448 examples. The concept drift occurs at example 50,000 and was detected 3,220 examples later. The cluster structure collapsed to a single leaf at example 62,448. Only after 9,224 examples from the moment that the complete structure collapsed, at example 71,672 (21,672 examples after the drift) did the system reach the correct structure for the second concept, which remains stable until the end of the data set (100,000). Simultaneously, we observed a dynamic behavior in the system, splitting and aggregating along the examples, until it stabilized on the final tree structure. The values in Table 3 are separated for each concept. The global results suggest good performance on splitting, concept drift detection, and aggregation.

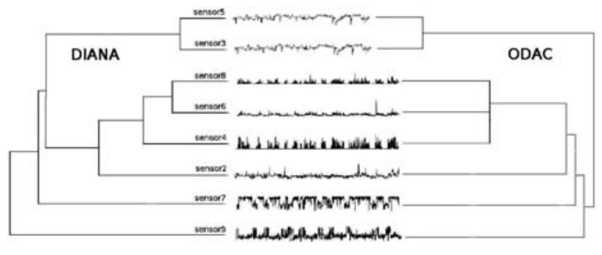


Fig. 3. Comparison between ODAC and DIANA (data for $\mathrm{userID} = 1$). Although a higher level structure is different, at the cluster level, both structures are quite close. This plot presents the proposed quality of the incremental system when compared with a batch system.

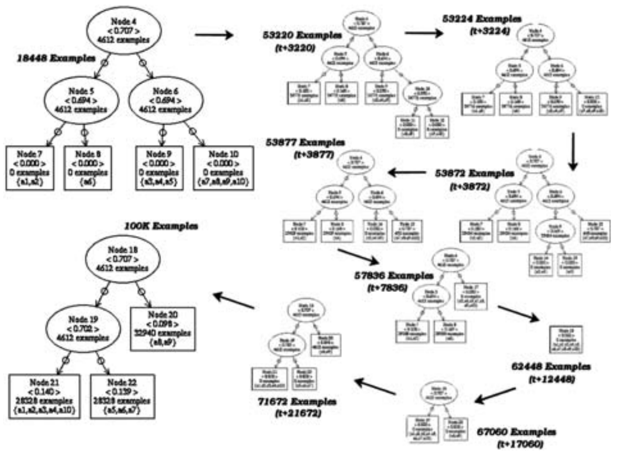

18448 Examples
Node 4 < 0.707 > 4612 examples
Node 5 < 0.694 > 4612 examples
Node 6 < 0.694 > 4612 examples
Node 7 < 0.000 > 0 examples {a1,a2}
Node 8 < 0.000 > 0 examples {a6}
Node 9 < 0.000 > 0 examples {a3,a4,a5}
Node 10 < 0.000 > 0 examples {a7,a8,a9,a10}
53220 Examples (t+3220)
53224 Examples (t+3224)
53877 Examples (t+3877)
53872 Examples (t+3872)
100K Examples
Node 18 < 0.707 > 4612 examples
Node 19 < 0.702 > 4612 examples
Node 20 < 0.098 > 32940 examples {a8,a9}
Node 21 < 0.140 > 28328 examples {a1,a2,a3,a4,a10}
Node 22 < 0.139 > 28328 examples {a5,a6,a7}
57836 Examples (t+7836)
62448 Examples (t+12448)
71672 Examples (t+21672)
67060 Examples (t+17060)

Fig. 4. ODAC structure evolution (concept drift data set). Start: the first concept is defined for the data set. After 18,448 examples, ODAC reaches the first concept's structure and stabilizes. After 50,000 examples (t), a concept drift occurs in the data set. After 53,220 examples $(\mathrm{t} + 3,220)$, ODAC detects changes in the structure. After 62,448 examples $(\mathrm{t} + 12,448, \mathrm{s})$, ODAC collapses all structure. After 71,672 examples,$(\mathrm{t} + 21,672, \mathrm{s} + 9,224)$, ODAC gathers the second concept and stabilizes. End: the second concept remains in the data set, and the correct final structure of the second concept was discovered.

To attest the ability to react to a concept drift, we have created 30 data sets with the same structure and monitored how long the system takes to react to a drift and stabilize in the new concept, even with high levels of noise. The average number of examples needed to detect a drift was around 4,900, and the average number of examples needed to stabilize on the next concept was 8,500 after the detection.

4.4 System Performance on Real-World Electrical Data

Time series of electrical data are one of the most widely studied sets of data, mainly for forecast purposes, usually by means of neural networks. One big problem with this approach is the time consumption of the neural network's training procedure. This fact, in conjunction with high dimensionality (common electrical networks combine thousands of different time series), suggests the need to cluster

TABLE 3
K-Means and ODAC Quality Results for Both Concepts of the Concept Drift Data Set

First Concept

System	*nc*	MHΓ	ΔMHΓ	DVI	CPCC
	2	0.176	–	0.993	–
K-Means	3	0.321	0.145	0.996	–
	4	0.387	0.066	**3.000**	–
	5	0.387	0.000	0.712	–
ODAC	**4**	0.331	–	3.000	0.839

Second Concept

System	*nc*	MHΓ	ΔMHΓ	DVI	CPCC
	2	0.278	–	0.996	–
K-Means	**3**	**0.345**	0.067	**3.000**	–
	4	0.345	0.000	0.710	–
	5	0.346	0.001	0.710	–
ODAC	**3**	0.146	–	3.000	0.801

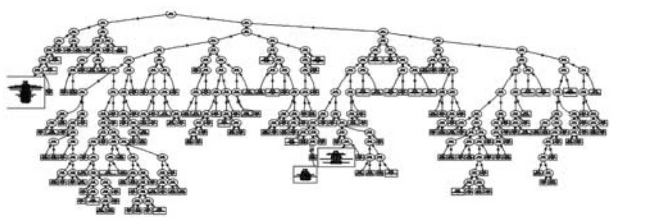


Fig. 5. Electrical demand data: active power. The resulting tree is quite large but is presented here to observe that the final structure is far from the worst-case scenario (list-like).

time series. Usually, an expert is required for this job. Our system may present some benefits in this particular task.

From the raw data received at each substation, gathered during four months, we have extracted only the variables related to the active power (according to the possibly erroneous variable ID). Each example represents the average value for the last five minutes of each variable. This resulted in a data set with 34,734 examples (120 days, 14 hours, and 30 minutes) with 887 different streams (variables). The system finds a complete structure of clusters over time, aggregating when necessary. The final structure can be analyzed in Fig. 5. The image is rather large and is presented only as an example of the structure obtained as a whole, revealing a tree-like structure away from the list-like worst-case scenario. We present a sketch of the variables for one of the clusters in Fig. 6. This cluster expresses a good intracluster correlation ($\mu = 0.629, \text{ and } \sigma = 0.227$), considering that this is real data from five variables along 7,385 observations. Nevertheless, clusters with worse intracluster correlation may be found, with some noisy variables included. This is probably due to the lack of examples fed to these clusters, reducing the confidence on splitting. It is expected that with more examples, these clusters would be split in the future.

4.4.1 Scalability Evaluation

A different set of sensors was used to conclude the scalability characteristics of our system. Taking into account

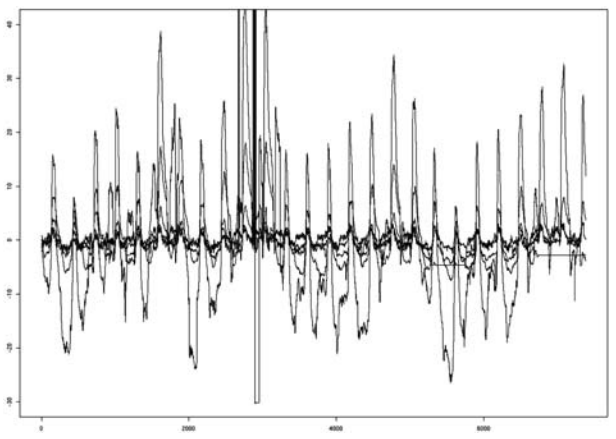


Fig. 6. Cluster from electrical data set: active power. In this cluster, five variables are kept together, along 7,385 examples, corresponding to 25 days, 15 hours, and 25 minutes. From this plot, the similarity between time series is clear, as the mean intracluster correlation is 0.629, with standard deviation 0.227.

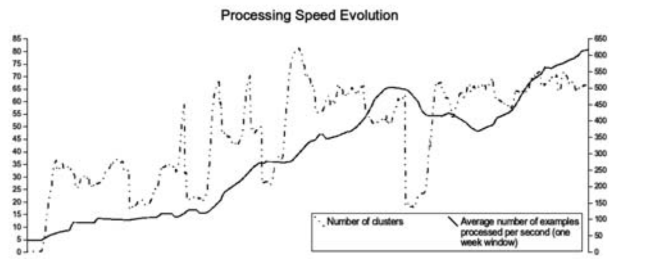


Fig. 7. Total processing speed evolution. This graph presents the evolution of the speed at which the system processes examples, both updating the sufficient statistics and testing clusters for splitting and aggregation, in terms of examples per second, averaged over a week window. The number of clusters is presented along time with respect to the secondary y-axis. Even with the increasing number of clusters to test, the total processing speed increases with splits.

only the current intensity sensors, we have aggregated observations on an hourly basis over more than 2.5 years. This data set represents 2,700 sensors along 22,364 observations. For this data set, we monitored both update speed and memory usage along time, as these are two of the main positive arguments of the proposed system. For all measures, we present a tendency graph based on an average on a week window. We can note that the overall speed of the system (Fig. 7) consistently grows along time, as the structure is growing, being even more pronounced in the update speed (Fig. 8). The advantages of local computation in the speedup achieved by splitting becomes clear. Moreover, even with the increasing number of clusters to test, the total processing speed increases with splits.

This is, in fact, motivated by the reduction of stored values needed for sufficient statistics computation, as shown in Fig. 9. The focus of the system on local computations reduces the problem of needing quadratic memory with respect to the number of sensors.

4.4.2 Sensitivity to the δ Parameter

We have used a subset of the current intensity sensors data to check the system's sensitivity to changes in the parameter δ used by the Hoeffding bound. An increase in this parameter represents a decrease in the confidence level of the Hoeffding bound. This data set represents 10 sensors along 6,000 observations. Fig. 10 plots the changes in the three measured validity indices for the selected sensors. The results on the DVI and the Modified Hubert's Γ statistic

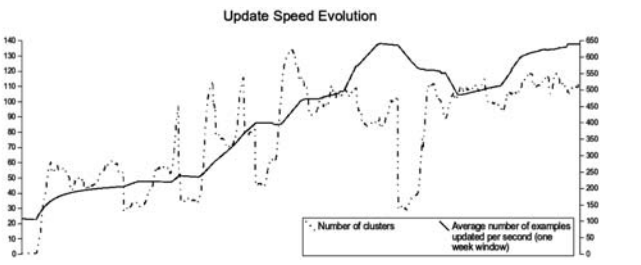


Fig. 8. Update speed evolution. This graph presents the evolution of the speed at which the system updates the sufficient statistics in terms of examples per second, averaged over a week window. The number of clusters is presented along time with respect to the secondary y-axis. The advantages of local computation in the speedup achieved by splitting becomes clear.

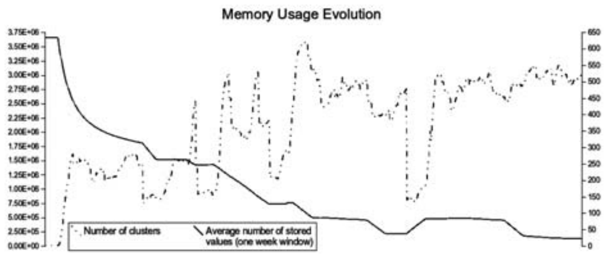


Fig. 9. Memory usage evolution. This graph presents the evolution of the global memory usage of system in terms of the values needed for sufficient statistics, averaged over a week window. The number of clusters is presented along time with respect to the secondary y-axis. The reduction on the stored values is significant. When the number of clusters is 1/5 the number of sensors, the memory usage diminishes to 5 percent of the original size.

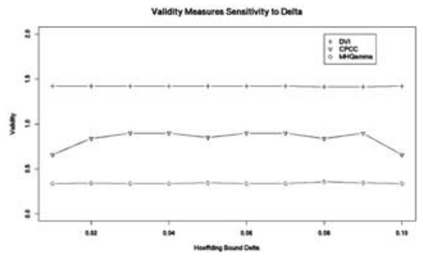


Fig. 10. Sensitivity of the system to changes in the δ parameter. This parameter seems to have some influence on the hierarchical quality of the structure (CPCC), without much influence on the clustering validity (DVI and $\mathrm{MH\Gamma}$).

indicate that δ has a small impact on the quality of the induced clusters. This parameter has more impact on the quality of the hierarchy found by the system, as the fluctuations in the CPCC statistic seem to indicate.

5 Strengths and Limitations

Hierarchical models present some characteristics that strengthen their usability. For example, they do not need a predefined number of target clusters. In the case of data streams, the most relevant property of hierarchical clustering is that the computational resources (time and memory) needed to process each example reduce as the hierarchy grows. Moreover, this decrease in the number of computed dissimilarities is lower bounded by the number of variables in the original cluster. This feature becomes even more relevant as the number of streaming series grows unbounded. In such cases, the computation of dissimilarities at the root level becomes a bottleneck, so the ability to grow a hierarchical structure and only compute dissimilarities locally to each cluster presents a much better approach to problems with thousands of time series.

However, in the data stream framework, the correlation structure between variables may change over time. This fact may require a restructuring of the clustering hierarchy, that is, to move variables from one cluster to another. The main problem is that this *moving* operation must follow the path defined by the actual structure. For example, considering the situation where the starting structure $\{\{\{a1, a2\}, \{a3, a4\}\}, \{\{a5, a6\}, \{a7, a8\}\}\}$ changes to $\{\{\{a1, a2, a5\}, \{a3, a4\}\}, \{\{a6\}, \{a7, a8\}\}\}$, although only one attribute $a5$ has changed, the system has to completely destroy the current hierarchy and reconstruct it from scratch. This behavior is expected to occur in any hierarchical model. However, we should note that this example is, in fact, the worst-case scenario. Maintaining dissimilarity matrices in all nodes would enable quick responses to abrupt changes in the lowest levels. However, this approach would also require exponential time and space to process examples. Our approach, based on updating the correlation matrix only at leaves, is limited to changes that smoothly evolve over time. Overall, the gain of not computing all dissimilarities at every bunch of examples compensates the possible burden of reconstructing the hierarchy from scratch from time to time if and when a concept drift might occur. Fuzzy clustering, where variables are shared in different clusters, seems to be a promising alternative. The main problem is how we can achieve the trade-off between a fast answer to changes and the time and space required to update the model.

6 Conclusions and Future Work

In this paper, we have presented ODAC, a clustering system for streaming time series. ODAC uses a top-down strategy to construct a binary tree hierarchy of clusters, with the goal of finding highly correlated sets of variables. A common measure of cluster quality is the cluster's diameter, which is defined as the highest dissimilarity between objects of the same cluster. The system evolves by continuously monitoring the clusters' diameters.

The examples are processed as they arrive by using a single scan over the entire data. The system incrementally computes the dissimilarities between time series, maintaining and updating the sufficient statistics at each new example arrival, updating only the leaves. The splitting criterion is supported by a confidence level given by the Hoeffding bound, which is detected when the system has gathered enough information to confidently define the diameter of each individual cluster.

ODAC is designed toward thousands of data streams that flow at a high rate. The two main characteristics are update time and memory consumption. Both reduce whenever the tree structure grows. This is a major achievement accomplished by ODAC, since only dissimilarities at the leaves must be computed. This way, every time the system grows, it becomes faster, overcoming the bottleneck of having to compute all dissimilarities at the root level, which is known to have a quadratic complexity on the number of streams.

The system includes an agglomerative phase, based on the diameters of existing clusters, also supported by the Hoeffding bound. The aggregation phase enables the adaptation of the cluster structure to smooth changes in the correlation structure of time series. To our best knowledge, this is the first incremental and hierarchical whole clustering system designed for high-speed time series, being able to dynamically adjust the clustering structure in reaction to environment changes.

Experimental results indicate that the performance is competitive for clustering time series, showing that the system is nearly equivalent to a batch divisive clustering on stationary time series. Experimental evaluation using artificial data sets shows the robustness of the system with respect to different time-series clustering hypotheses and the ability to adapt in the presence of evolving concepts. Results using real-world data sets show a competitive performance when compared with the batch clustering analysis. They also reveal a good performance on finding the correct number of clusters obtained by a bunch of runs of *K-Means*.

When considering the expansion of the structure, the strict splitting of variables appears as a possible drawback in the sense that a previous decision of moving a variable to a leaf when there is no statistical confidence on the decision of assignment may split variables that should be together. An approach based on fuzzy sets [31] would let forthcoming examples decide what should be done with those variables. This approach is already scheduled as a future work task, as it seems to be a very important option in time-series incremental clustering. Moreover, the computation effort forced by the high dimensionality of the data being processed may represent a drawback to the clustering procedure. Thus, techniques based on clipping [32] could also be considered.

Acknowledgments

The work of P.P. Rodrigues is supported by the Portuguese Foundation for Science and Technology under the PhD Grant SFRH/BD/29219/2006. The authors thank the Plurianual financial support attributed to LIAAD and the participation of Project ALES II under Contract POSC/EIA/55340/2004 and Project RETINAE under Contract PRIME/IDEIA/70/00078.

References

[1] P. Domingos and G. Hulten, "Mining High-Speed Data Streams," *Proc. ACM SIGKDD '00,* pp. 71-80, 2000.

[2] D. Barbará, "Requirements for Clustering Data Streams," *SIGKDD Explorations,* vol. 3, no. 2, pp. 23-27, Jan. 2002.

[3] E. Keogh and S. Kasetty, "On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration," *Proc. ACM SIGKDD '02,* pp. 102-111, July 2002.

[4] M. Halkidi, Y. Batistakis, and M. Varzirgiannis, "On Clustering Validation Techniques," *J. Intelligent Information Systems,* vol. 17, no. 2-3, pp. 107-145, 2001.

[5] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference and Prediction.* Springer Verlag, 2000.

[6] P.P. Rodrigues, J. Gama, and J.P. Pedroso, "ODAC: Hierarchical Clustering of Time Series Data Streams," *Proc. Sixth SIAM Int'l Conf. Data Mining (ICDM '06),* pp. 499-503, Apr. 2006.

[7] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering Data Streams: Theory and Practice," *IEEE Trans. Knowledge and Data Eng.,* vol. 15, no. 3, pp. 515-528, May/June 2003.

[8] F. Ferrer, J. Aguilar, and J. Riquelme, "Incremental Rule Learning and Border Examples Selection from Numerical Data Streams," *J. Universal Computer Science,* vol. 11, no. 8, pp. 1426-1439, 2005.

[9] A.K. Jain and R.C. Dubes, *Algorithms for Clustering Data.* Prentice Hall, 1988.

[10] L. Kaufman and P.J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley & Sons, 1990.

[11] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," *Proc. ACM SIGKDD '96,* pp. 226-231, 1996.

[12] W. Wang, J. Yang, and R.R. Muntz, "STING: A Statistical Information Grid Approach to Spatial Data Mining," *Proc. 23rd Int'l Conf. Very Large Data Bases (VLDB '97),* pp. 186-195, 1997.

[13] D.H. Fisher, "Knowledge Acquisition via Incremental Conceptual Clustering," *Machine Learning,* vol. 2, no. 2, pp. 139-172, 1987.

[14] P. Bradley, U. Fayyad, and C. Reina, "Scaling Clustering Algorithms to Large Databases," *Proc. ACM SIGKDD '98,* pp. 9-15, 1998.

[15] L. O'Callaghan, A. Meyerson, R. Motwani, N. Mishra, and S. Guha, "Streaming-Data Algorithms for High-Quality Clustering," *Proc. 18th IEEE Int'l Conf. Data Eng. (ICDE '02),* pp. 685-696, 2002.

[16] G. Cormode, S. Muthukrishnan, and W. Zhuang, "Conquering the Divide: Continuous Clustering of Distributed Data Streams," *Proc. 23nd IEEE Int'l Conf. Data Eng. (ICDE '07),* pp. 1036-1045, 2007.

[17] T.F. Gonzalez, "Clustering to Minimize the Maximum Inter-Cluster Distance," *Theoretical Computer Science,* vol. 38, nos. 2-3, pp. 293-306, 1985.

[18] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases," *Proc. ACM SIGMOD '96,* pp. 103-114, 1996.

[19] C. Aggarwal, J. Han, J. Wang, and P. Yu, "A Framework for Clustering Evolving Data Streams," *Proc. 29th Int'l Conf. Very Large Data Bases (VLDB '03),* pp. 81-92, Sept. 2003.

[20] S. Guha, R. Rastogi, and K. Shim, "CURE: An Efficient Clustering Algorithm for Large Databases," *Proc. ACM SIGMOD '98,* pp. 73-84, 1998.

[21] J. Gama, P. Medas, and P. Rodrigues, "Learning Decision Trees from Dynamic Data Streams," *Proc. 20th ACM Symp. Applied Computing (SAC '05),* pp. 573-577, Mar. 2005.

[22] G. Hulten, L. Spencer, and P. Domingos, "Mining Time-Changing Data Streams," *Proc. ACM SIGKDD '01,* pp. 97-106, 2001.

[23] W. Hoeffding, "Probability Inequalities for Sums of Bounded Random Variables," *J. Am. Statistical Assoc.,* vol. 58, no. 301, pp. 13-30, 1963.

[24] K. Pearson, "Regression, Heredity and Panmixia," *Philosophical Trans. Royal Soc.,* vol. 187, pp. 253-318, 1896.

[25] L. Leydesdorff, "Similarity Measures, Author Cocitation Analysis, and Information Theory," *J. Am. Soc. Information Science and Technology,* vol. 56, no. 7, pp. 769-772, 2005.

[26] M. Wang and X.S. Wang, "Efficient Evaluation of Composite Correlations for Streaming Time Series," *Proc. Fourth Int'l Conf. Web-Age Information Management (WAIM '03),* pp. 369-380, 2003.

[27] E. Fowlkes and C. Mallows, "A Method for Comparing Two Hierarchical Clusterings," *J. Am. Statistical Assoc.,* vol. 78, no. 383, pp. 553-569, 1983.

[28] L. Hubert and J. Schultz, "Quadratic Assignment as a General Data-Analysis Strategy," *British J. Math. and Statistical Psychology,* vol. 29, pp. 190-241, 1975.

[29] J. Dunn, "Well-Separated Clusters and Optimal Fuzzy Partitions," *J. Cybernetics,* vol. 4, no. 1, pp. 95-104, 1974.

[30] "R: A Language and Environment for Statistical Computing," *R Foundation for Statistical Computing.* R Development Core Team, http://www.R-project.org, 2005.

[31] L.A. Zadeh, "Fuzzy Sets," *Information and Control,* vol. 8, no. 3, pp. 338-353, 1965.

[32] A. Bagnall and G. Janacek, "Clustering Time Series with Clipped Data," *Machine Learning,* vol. 58, no. 2-3, pp. 151-178, 2005.

Pedro Pereira Rodrigues received the BSc and MSc degrees in computer science from the University of Porto, Porto, Portugal, in 2003 and 2005, respectively. He is currently working toward the PhD degree at the University of Porto, where he is also a researcher in the Artificial Intelligence and Decision Support Laboratory, working on the distributed clustering of streaming data from sensor networks. His research interests include machine learning and data mining from distributed data streams and the reliability of predictive and clustering analysis in streaming environments, with applications on industry-related and health-related domains.

João Gama received the PhD degree in computer science from the University of Porto, Porto, Portugal, in 2000. He is currently an assistant professor in the Faculty of Economics and a researcher in the Artificial Intelligence and Decision Support Laboratory, University of Porto. His research interests include online learning from data streams, combining classifiers and probabilistic reasoning.

João Pedro Pedroso received the BSc degree in chemical engineering from the University of Porto, Porto, Portugal, in 1989 and the PhD degree in applied sciences from the Université Catholique de Louvain, Louvain-la-Neuve, Belgium, in 1996. He is currently an assistant professor in the Faculty of Sciences, University of Porto. His research interests include optimization, mathematical programming, and operations research.