Clustering Multivariate Data Streams by Correlating Attributes using Fractal Dimension

Christian C. Bones[1], Luciana A. S. Romani[2], Elaine P. M. de Sousa[1]

[1] University of São Paulo, Brazil
chris,parros@icmc.usp.br
[2] Embrapa Agriculture Informatics – Campinas – SP – Brazil
luciana.romani@embrapa.br

Abstract. A data stream is a flow of data produced continuously along the time. Storing and analyzing such information become challenging due to exponential growth of the data volume collected. Recently, some algorithms have been proposed to cluster data streams as a whole, but just few of them deal with multivariate data streams. Even so, these algorithms merely aggregate the attributes without touching upon the correlation among them. Aiming to overcome this issue, we propose a new framework to cluster multivariate data streams based on their evolving behavior over time, exploring the correlations among their attributes by computing the fractal dimension. In order to evaluate our framework we used real multisource and multidimensional climate data streams. Our results show that the clusters' quality and compactness can be improved compared to the competing methods, leading to the thoughtfulness that attributes correlations cannot be put aside. In fact, the clusters' compactness are 14 to 25 times better using our method. Also our framework was 3 to 20 times faster than our competitors. Our framework also proves to be an useful tool to assist meteorologists in understanding the climate behavior along a period of time.

Categories and Subject Descriptors: H.2 [**Database Management**]: Miscellaneous; H.2.8 [**Database Applications**]: Data mining; H.3.3 [**Information Search and Retrieval**]: Clustering

Keywords: Data Mining, Data Streams, Clustering, Fractal Dimension

1. INTRODUCTION

The increasing number of devices and sensors that continuously generate a huge amount of data leads to new challenges and applications. For instance, sensors have been used to monitor the pollution in cities, the level of rivers (to prevent flooding) and the meteorological conditions. These flows of data, generated ad infinitum, usually at a high speed rate, are called data streams.

In this scenario, extracting knowledge from data streams becomes an active research topic [Aroche-Villarruel et al. 2015; Fanaee-T and Gama 2015; Faria et al. 2016; Lughofer and Sayed-Mouchaweh 2015; Bifet and De Francisci Morales 2014; Zhang et al. 2014; Pereira and de Mello 2014; Chairukwattana et al. 2014; Widiputra et al. 2011] with applications in several contexts. There are some main challenges to be overcome to extract valuable knowledge from data streams, such as: *(i)* read data only once; *(ii)* capture and represent data evolution along the time; *(iii)* provide answers as soon as the user demands them. It is also desirable to deal with multidimensional data, considering all the variables composing each stream, i.e., multivariate data streams.

In data stream mining, a useful approach to extract valid information from data streams is to group in the same cluster data streams that have similar properties and behavior over time, whereas data streams of different clusters must present dissimilar characteristics. For instance, clustering techniques

The authors are grateful to CAPES, CNPq and FAPESP for their financial support.

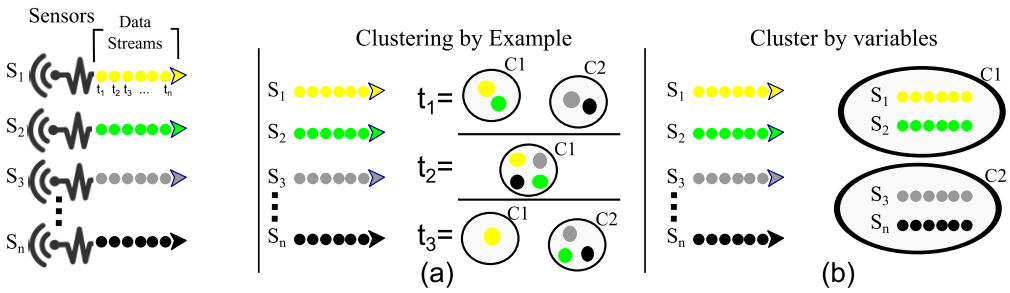


Fig. 1. Difference between clustering by examples and clustering by variables.

could be applied to cluster data streams from climate sensors to try to identify those with similar behavior along a period of time.

In this work we focus on clustering tasks, specially considering multivariate data stream. Some methods have been developed to work with multiple data streams [Rodrigues et al. 2008; Chaovalit and Gangopadhyay 2009; Widiputra et al. 2011; Pereira and de Mello 2014]. Their approaches in data streams clustering fall into two main categories that are: *Clustering by example*, in which all data points are clustered independently according to the similarity among them, regardless of the data sources these data points are from; *Clustering by variable* or *Clustering the entire data stream*, in which a data stream is compared with other data streams and those similar will be clustered into the same cluster. Fig. 1 shows the difference between these two approaches: Fig. 1(a) illustrates clustering by examples at different times t_i such that clusters C_1 and C_2 are created considering the properties of individual data points at time t_1 and, therefore, points from different streams can be at the same cluster. On the other hand, the clustering by variable approach in Fig. 1(b) shows that each cluster is built considering the general properties of the entire flow of data from each sensor.

Although several online clustering methods have been proposed to process and analyze flows of data in real time [Rehman et al. 2014; Pereira and de Mello 2014; Zhang et al. 2014; Ackermann et al. 2012; Widiputra et al. 2011; Chaovalit and Gangopadhyay 2009; Rodrigues et al. 2008; Aggarwal et al. 2007], only few of them try to extract valuable information and group the entire data streams based on their similar behavior over the time [Pereira and de Mello 2014; Widiputra et al. 2011; Chaovalit and Gangopadhyay 2009; Rodrigues et al. 2008]. Furthermore, these methods have failed on clustering data streams that have more than one attribute, i.e., multivariate flows, because they only consider the similarity of attributes independently. And besides these methods claimed to deal with real time sources of data, they take a considerable time to process each datum.

In this work, we are interested in clustering the entire data stream, as we focus on clustering climate sensors that behave similarly along the time. Moreover, we are especially interested in clustering sensors that generate multivariate data streams, taking into account the correlation among the attributes. As climate data streams usually have more than one attribute (e.g. temperature and precipitation) and, in such way, those attributes may have some correlation, considering each attribute as an independent flow is not the best approach for clustering climate sensors. We are also interested in validating our method to process new data from many sources as soon as they arrive in an affordable time, meeting the expectations of real time computing.

In this context, this article proposes a new framework called Evolving Fractal-based Clustering of Data Streams (eFCDS), which is an extension of the work presented in [Bones et al. 2015]. The main module is a novel method for clustering multivariate data streams, based on the calculation of fractal dimension. Our method utilizes the fractal dimension, calculated for data streams that have more than one attribute, to cluster data streams that behave similarly along the time. Also, our method keeps the evolution of the data stream by checking cluster membership whenever a new value of fractal

dimension is obtained. In other words, our method checks if the data stream is still belonging to the same cluster or if it should be allocated in another one that better describes its behavior in that period of time. Moreover, our method takes in consideration the time processing of each datum to potentially be used with real time application.

In order to evaluate our method, we used a dataset provided by EMBRAPA Agriculture Informatics - Empresa Brasileira de Pesquisa Agropecuária - Campinas. The dataset includes multivariate data streams from climate sensors of different Brazilian regions. Our results not only indicated that our approach can be a useful method to assist specialists in analyzing large amounts of climate data, but also helps to identify regions with the same climate behavior along the time.

The rest of this article is organized as following. Section 2 presents background concepts and related work. Section 3 describes our approach to cluster data streams. Experimental results are discussed in Section 4 and Section 5 presents final remarks and future work.

2. BACKGROUND AND RELATED WORK

Data stream is an ordered collection of data $s_1, s_2, ...$, generated continuously by one or more sources, that usually can be read only once due to high speed generation rate [Guha et al. 2003]. In addition, a data stream could have more than one attribute per datum, defining a multivariate data stream. Formalizing these concept, let $\mathbb{S} = \{S_1, \ldots, S_n\}$ be a set of data streams sources where each $S_i = \left\{\vec{d_1}, \ldots, \vec{d_\infty}\right\}$ is a multivariate data stream. Also, each S_i is assumed to contain f attributes such that $\vec{d_i} = [a_1, \ldots, a_f]$ is the set of attributes values.

Data stream clustering is a data mining technique to perform the grouping of flows of data, so that objects within the same group must have very similar characteristics and properties [Aggarwal et al. 2006]. These characteristics should differ as much as possible from group to group. One of the most common ways to cluster objects is to measure the distance between them [Gama 2010]. However, clustering similar characteristics can be very costly [Guha et al. 2003]. Also, clustering data streams can be used to overview the data distribution and as a preprocess for other algorithms [Gama 2010]. In order to achieve these goals, some basic requirements must be presented in data stream clustering algorithms [Barbará 2002]: (i) Representation of compact size; (ii) Quickly and incremental processing of new data items; (iii) Traceability of changes in groups; (iv) Quick and clear identification of outliers.

Therefore, any new data stream clustering method must be adapted to perform clustering in a continuous, concise and evolving online way entry of the observed sequence. Furthermore, the temporal characteristic of the data stream must also be considered using a small amount of storage space [Guha et al. 2003] and processing time. Such requirements are imposed to contemplate the continuous manner in which the data arrives and the need for analyzing them in real time [Gama 2010].

As previously mentioned, data stream clustering algorithms follow one of the two main approaches: clustering by example, and clustering by variable. We are interested in the latter approach, discussed in Section 2.1.

2.1 Clustering by Variable

Most of the methods proposed in the literature to cluster the entire data streams deal with flows that have only one attribute [Widiputra et al. 2011]. Those methods that support more than one attribute do not consider the correlation among attributes to cluster the data streams [Chaovalit and Gangopadhyay 2009; Rodrigues et al. 2008; Rehman et al. 2014; Miller et al. 2014]. They only consider to cluster if all the attributes are similar to all the attributes of other data stream, possibly leading to results that do not accurately correspond to the behavior of the data streams along the time.

Beyond that, the number of methods that cluster data streams that behave similarly over the time is even more restricted, including the ECM method [Widiputra et al. 2011] and POD-Clus method [Chaovalit and Gangopadhyay 2009]. These two methods, detailed as follows, will be used to compare the achievements of our approach.

ECM

The Evolving Clustering Method (ECM) performs predictions using univariate data streams [Widiputra et al. 2011]. ECM builds local models in two main steps: 1) the extraction of relationships between data streams profiles; and 2) the detection and clustering of recurrent trends when a particular profile emerges. ECM calculates the relationship among data streams profiles using Pearson's correlation [Rodrigues et al. 2008], extracting only the most significant coefficients obtained through tests of statistical significance. Given two data streams a and b the Pearson's correlation is:

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

where $A = \sum a_i$, $B = \sum b_i$, $A_2 = \sum {a_i}^2$, $B_2 = \sum {b_i}^2$, $P = \sum a_i b_i$. The Pearson's correlation can be easily updated as soon as each new datum arrives. Then, the ECM calculates the dissimilarity between the data streams a and b by RNOMC (Rooted Normalized One-Minus-Correlation) equation. A drawback is that ECM requires that the whole time series is previously available to perform the calculations offline. Based on a set of decision rules, the algorithm decides how to group the series opting to create, remove, or join groups.

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

Due to its measure of dissimilarity, ECM only detects linear relationships. Another point that limits the application of ECM is how the algorithm decides the union, creating or adding of a new element to the cluster, for example, as an element d_j will only be added to the group of d_i if d_j is correlated with all existing elements in d_i, so if d_i has thousand elements and d_j has only one element not correlated with d_i, then d_j will not be added to d_i's cluster. Another disadvantage is that ECM clusters data streams with only one attribute.

POD-Clus

The POD-Clus algorithm (*Probability and Distribution-based Clustering*) [Chaovalit and Gangopadhyay 2009] has four approaches: *clustering by examples* and *clustering the entire data streams* without catching the evolution of the clusters; *clustering by examples* and *clustering the entire data streams* monitoring the evolution of the clusters. Only the latter one is of interest in the context of this work.

The POD-Clus seeks to maintain summaries and discard detailed information of the data points, using normal distributions for this purpose. Each POD-Clus' cluster k receives n data points and stores some average statistics: such as the average μ, standard deviation σ and the updated covariance matrix, whenever new data arrives. These statistics are used to measure the similarity between data streams considering each attribute according to the Equation 3. The distance of a multivariate data stream S to a cluster center C is defined as [Kumar and Patel 2007]:

$$D_{SC} = \sum_{f=1}^{F} \frac{\left(\mu_{Sf} - \mu_{Cf}\right)^2}{\sigma_{Sf}^2}, \tag{3}$$

where S denotes a data stream, C denotes a cluster, f denotes a data feature, μ_{Sf} is the mean of f in the data stream S, σ_{Sf} is the standard deviation of f in data streams S, and μ_{Cf} is the mean of f in the cluster C.

POD-Clus also monitors the progress of each cluster, identifying the emergence, the disappearance, the union and the division of the clusters. To detect the appearance of a new group, it maintains a cluster of outliers and when one of these clusters reaches the minimum amount of data streams defined, this cluster becomes a new effective cluster. A cluster disappears when it stops receiving new data by a certain amount of time and old data receives less importance according to a fading factor. To join a cluster to another, it is checked the amount of data which may be in both clusters, generating an overlap between them. If the amount of overlapping data is greater than a predetermined threshold, then the two clusters are merged. To split a group POD-Clus monitors its density and compares it with the normal distribution. If there is significant difference, the group is divided.

The POD-Clus assumes that the whole data stream is derived from a normal distribution, otherwise it does not guarantee a good representation of the clusters. Another POD-Clus' drawback is the fact that to join a data stream to a cluster all its attributes f have to be similar to the C's features f, disregarding the correlation between the attributes.

2.2 Fractal Dimension

In this work we propose a clustering data stream method based on the calculus of the fractal dimension, which is used to identify the correlation among the attributes of a data stream in order to capture the data streams' behavior over the time. Then, this behavior is used to measure the similarity among data streams aiming to achieve better clusters results.

A fractal is characterized by the self-similarity property, i.e., it is an object that presents roughly the same characteristics when analyzed over a large range of scales. From the Fractal Theory, the Correlation Fractal Dimension D_2 is particularly useful for data analysis, since it can be applied to estimate the intrinsic dimension of real datasets that exhibit fractal behavior, i.e., exactly or statistically self-similar datasets [Belussi and Faloutsos 1995]. The Correlation Fractal Dimension D_2 measures the non-uniform behavior of real data considering both linear and nonlinear attribute correlations [Sousa et al. 2007]. Therefore, D_2 represents the dimensionality of the dataset regardless of the dimension E of the space defined by its attributes. For instance, a set of points defining a line $z = ax + by + c$ embedded in a three-dimensional space with dimensions $[X; Y; Z]$ (and thus E = 3) has $D_2 = 1$, as there is a linear correlation between its attributes [Sousa et al. 2007].

A method to measure the fractal dimension of datasets embedded in E-dimensional spaces is the Box-Counting method, which defines D_2 as [Schroeder 1991]:

$$D_2 = \frac{\partial log(\sum_i C_{r,i}^2)}{\partial log(r)} \quad r \in [r_1, r_2] \tag{4}$$

where r is the side of the cells in a (hyper) cubic grid that divides the address space of the dataset and $C_{r,i}$ is the count of points in the ith cell. Based on Equation 4, the log-log graph of the sum of squared occupancy $\sum_i C_{r,i}^2$ for distinct values of r is called Box-Counting plot.

The work presented in [de Sousa et al. 2007] proposes a technique to detect changes in multidimensional evolving data streams, based on the information of intrinsic behavior provided by the fractal dimension D_2. The authors also present the algorithm SID-meter to continuously measure D_2 over time aimed at monitoring the evolving behavior of the data, such that significant variations in successive measures of D_2 can indicate changes in the intrinsic characteristics, as well as in attribute correlations in the data.

The SID-meter uses a sliding window to determine the amount of data to be used to compute D_2. Its sliding window is divided in counting periods n_c and each n_c is composed of a delimited number of data, called events. Firstly, the SID-meter creates an initial bounding hyper-cube by using the events that are in the first n_c. The lowest rL_i and the highest rH_i values for each attribute a_i of the received events are computed. The range $r_0 = max_{i=1}^{E}(rH_i - rL_i)$ determines the size of the side in

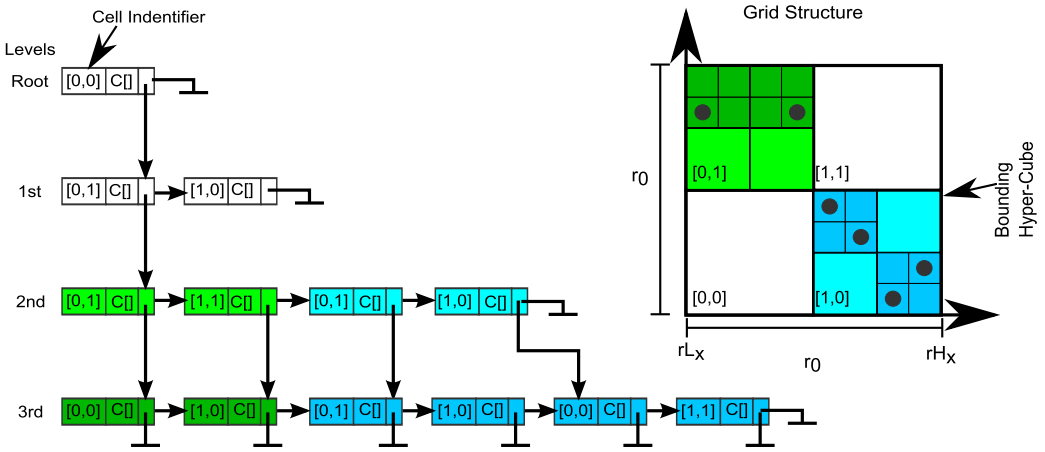


Fig. 2. Representation of a 2-dimensional Hyper-Cube with 3-level Grid Structure and its respective Counting Tree.

the initial hyper-cube. Figure 2 exemplifies the initial hyper-cube created to bound six events of a 2-dimensional data stream. The side r_0 of the bounding hyper-cube is determined by the range $[rL_x; rH_x]$ of the attribute represented in the x-axes.

The hyper-grid structure is created by generating up to R successive E-dimensional grids of cell side $r_j = r_{j-1}/2$, where R determines the number of points in the Box-Counting plot. Thus, for each cell at level j, 2^E cells are generated in level $j+1$. Figure 2 illustrates the grid structure for the six 2-dimensional events. The counting tree is a memory based data structure. Each level j in the tree corresponds to the grid of the cell side r_j and each node corresponds to a cell. Every cell is part of one cell in the immediate upper level and is identified by identifier $[b_1 b_2 \cdots b_E]$, such that $b_i = 0$ for cells in the lower half of dimension i and $b_i = 1$ otherwise. Figure 2 also illustrates the 3-level counting tree generated for the grid structure created with six 2-dimensional events. The array $C\,[]$ at each node of the counting tree represents the n_c, each one storing the event counter in the corresponding period.

$$C_{r_j,i} = \sum_{k=0}^{k<nc} C_i\,[k] \tag{5}$$

Thus, D_2 can be computed through Equation 4. The value of $C_{r_j,i}$ at each node i is calculated through Equation 5, for each counting tree level j (recall that a level j corresponds to a grid of cell side r_j). The Box-Counting plot is created by performing a full navigation through the counting tree and plotting $< \log\left(\sum_{i=0}^{i<2^E} C^2_{r_j,i}\right), \log(r_j) >$ for each value r_j. Then, the slope of the line that best fits the curve gives an estimate of D for the events inside the sliding window.

3. EVOLVING FRACTAL-BASED CLUSTERING OF DATA STREAMS FRAMEWORK

In this article we propose the framework Evolving Fractal-Based Clustering of Data Streams, eFCDS for short. A preliminary naïve version of the framework, with no support for overlapping identification or outlier discovery, was introduced in [Bones et al. 2015]. We now present the improved eFCDS, which deals with both of these issues: 1) it reduces overlap between clusters by merging or splitting them; and 2) it detects outliers by searching for potential outlier clusters (i.e. clusters with a single data stream) that could not be merged with their closest (non-outlier) clusters. Moreover, we evaluate the performance of the eFCDS with a larger real dataset, containing 50 years of climate data.

The eFCDS aims at partitioning the set $\mathbb{S}$, defined in Section 2, in a collection $P = \{C_1, \ldots, C_m\}$ of m disjoint clusters regarding to the fractal dimension analysis of each S_i. Each cluster C is composed

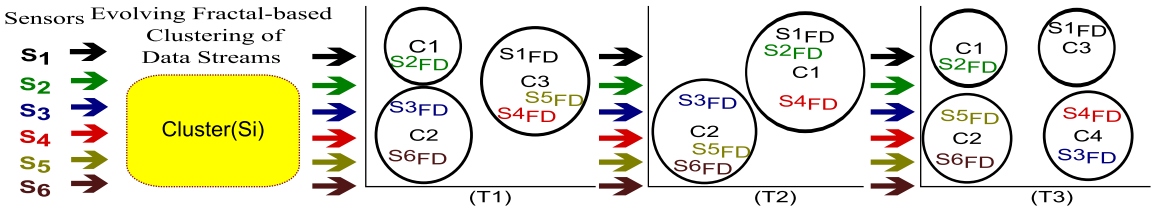


Fig. 3. General idea of clustering data streams, where each SX_{FD} represent a sensor, that is clustered using the Fractal Dimension measure D_2 in the time T_i.

of the data stream sources considered similar to each other if they do not exceed an user-defined maximum standard deviation parameter. Our framework clusters data streams with a similar behavior in an interval of time T_i and also takes into account the correlation among the attributes measured by the fractal dimension D_2, calculated by Equation 4. Our method also follows the evolution of the data streams, where clusters can disappear or be created.

Fig. 3 shows the idea of data stream sources (e.g. sensors) generating new data continuously and sending them to the proposed *Evolving Fractal-based Clustering of Data Streams* framework (eFCDS), which produces evolutive clusters. Notice that for each interval of time T_i, the gathered data are clustered following the fractal dimension of the available data. As new information are received, the clusters are created or rearranged so as to ensure the similarity among their elements and the correlation among the attributes along the time. For instance, from period T_1 to T_2 it is possible to notice a cluster disappearance and from period T_2 to T_3 two new clusters were created.

Let us now illustrate the main components of our proposed framework. The process initiates when a defined number of data stream sources are chosen to be analyzed. As the sources are producing data, they can be directly forwarded to the eFCDS framework in order to be processed, as shown in Fig. 4.

The input component of the eFCDS is the Sliding Window Module. This module receives the data streams and bounds their information by a sliding window. The sliding window specifies the amount of data buffered for the fractal dimensional calculation. The window is divided into counting periods (t), and each period has a defined number (e) of events such that each event represents $\vec{d}$ data points. Therefore, $t \times e$ defines the window size l and e also represents its movement step. The window takes a default size l, which usually considers either domain experts or the seasonality of the data. Fig. 5 shows sliding windows w_i of size $l = 12$, that means four counting periods t ($t = 4$) where each period has three events ($e = 3$).

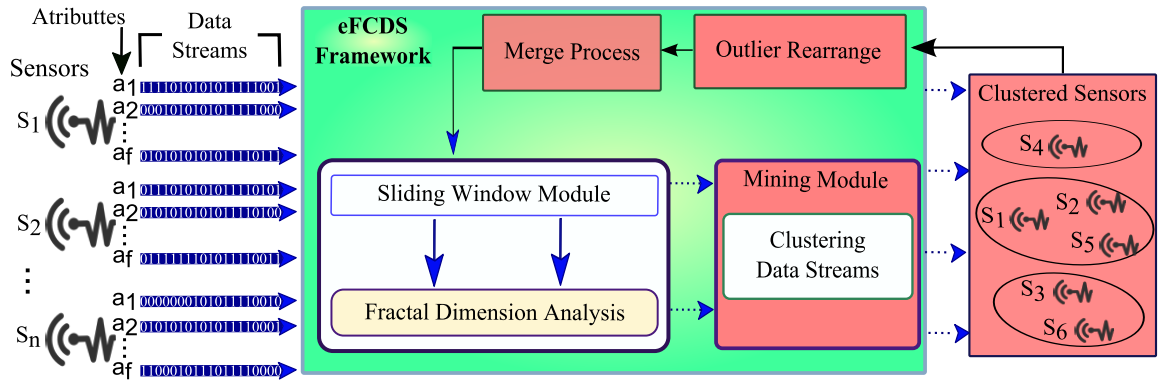


Fig. 4. An overview of our approach to cluster data streams.

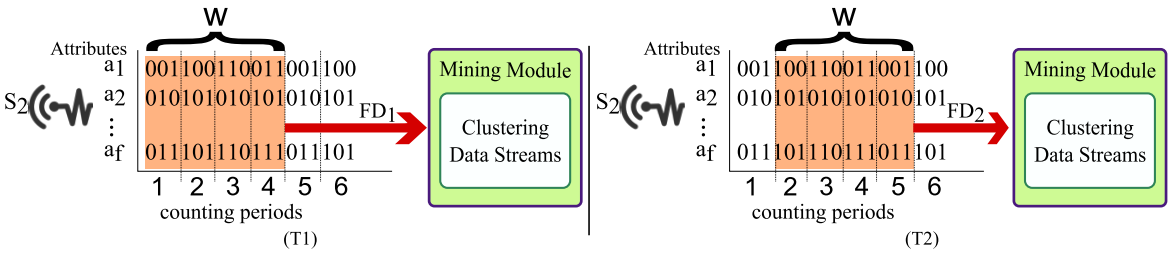


Fig. 5. Sliding Window (W) of a sensor, counting periods $t = 4$, events $e = 3$. In the instant T1 the window starts at the period 1 and ends at period 4; after shifting the window in T2, (W) starts at period 2 and ends at period 5.

As soon as there are enough information fulfilling the amount of events e, those data can be now unbuffered for their unique reading, e.g., instant T1 in Fig 5. In such case, the sliding window is shifted forward in e units and the region storing the already processed piece of information is released in order to store the continuous incoming data, e.g., instant T2 in Fig 5. Additionally, the previously windowed information is dispatched to the module responsible for performing the Fractal Dimension analysis.

The Fractal Dimension Analyzer considers an analog approach applied by SID-Meter [de Sousa et al. 2007] to perform the incremental calculus of D_2. Unlike the methods already proposed in the literature, this kind of technique enables to iterate over data containing more than one attribute. Thus, by applying techniques such as the box-counting (Section 2.2), this module transforms a piece of multivariate data stream of size l, which represents the measures, to exactly one point of fractal correlation, summarizing that amount of data and the correlation among their attributes. Our intuition is that using a technique which correlates the involved attributes along the time leads to a better recognition of similar data stream sources than using the raw values of the attributes.

Therefore, at each instant T_i it is generated a fractal dimension measure D_2 for each sensor, representing the sensor's behavior at the corresponding period.

Subsequently, the reduced amount of fractal points is sent to the data mining step. The Mining Module is the main component of the eFCDS framework and aims at clustering the correlated points according to their similarity, obtained by the Manhattan distance (L1) [Sokal 1985], and a user-defined maximum standard deviation (σ_{MAX}). In order to perform the clustering step, let us introduce the Algorithm 1.

The Algorithm 1 iterates over the set X comprising the points of fractal correlation (fd) such that each point x is labeled with its respective data stream S_i. Initially, it is verified whether or not the data stream source S_i already belongs to some existing cluster C_j composing the partition P (line 2). Supposing S_i was not clustered yet, it is necessary to search for a cluster to insert S_i into. So, for each cluster $C \in P$, the boolean procedure `findCluster` (line 8) looks for the cluster C_j with the lowest difference between the centroid of C_j and the analyzed point x. This process follows the model introduced in Equation 6, where $C = \{c_1, \ldots, c_k, \ldots, c_n\}$.

$$\Delta(C, x) = \left(\frac{1}{n} \sum_{k=1}^{n} c_k.fd \right) - x.fd \tag{6}$$

Algorithm 1: Evolving Fractal-based Clustering of Data Streams (eFCDS)

Input: The set X containing pairs (S_i,fd) of streams and its fractal dimension, respectively;
The partition P of clusters, such that in the first iteration $P = \{\}$;
The maximum standard deviation σ_{MAX}.
Output: The partition P of clusters with similar data streams.

```
1  foreach x ∈ X do
2      if x.S_i is in a cluster C_j ∈ P then
3          if updateCluster(C_j, x, σ_MAX) = false then
4              if findCluster(x, σ_MAX) = false then
5                  C_new ← createNewCluster (x);
6                  rearrangeToClosest(C_j, C_new);
7      else
8          if findCluster(x, σ_MAX) = false then
9              C_new ← createNewCluster (x);
10 return P;
```

Once the cluster C which minimizes the Equation 6 regarding to the element x is obtained, x is assigned to C iff the condition expressed in Equation 7 holds. Otherwise, the new cluster C_{new} containing the element x is created (line 9) and included in the partition P.

$$\sqrt{\frac{1}{n+1}\left(\left(x.fd - \left(\frac{1}{n}\sum_{p=1}^{n} c_p.fd\right)\right)^2 + \sum_{k=1}^{n}\left(c_k.fd - \left(\frac{1}{n}\sum_{p=1}^{n} c_p.fd\right)\right)^2\right)} \leq \sigma_{MAX} \tag{7}$$

Considering now the data stream S_i is already member of some cluster, it is necessary to check whether or not the data stream source S_i remains in the previously assigned cluster. In order to employ such verification (line 3), the Mining module re-execute the calculus of Equation 7 regarding to the new value of the fractal dimension $x.fd$. If so, the statistic (function Δ) of the considered cluster C_j is updated. In the case where the left-hand side of Equation 7 exceeds the user-defined maximum standard deviation σ_{MAX}, the algorithm tries to reallocate the element x in an existing cluster C_k so as to minimize the value computed in Equation 6 (line 4) and also hold the condition defined in Equation 7. If there is not such cluster C_k satisfying both conditions, a new one (C_{new}) is created to include the element x (line 5). Now, when creating a new cluster to x, the elements already in C_j are checked if they are better included in C_{new} (minimizing Equation 6) and inserted in it if they do (line 6). Notice that the `rearrangeCluster` procedure is able to suppress existing clusters if all of their elements are moved and they become empty. After that, the algorithm iterates over the clusters to identify those with single data streams, i.e., potential outliers. If anyone is found the algorithm tries to rearrange the data stream into the closest non outlier cluster following the aforementioned criteria. It is important to notice that if the closest cluster does not admit the addition of the outlier cluster no other will bear.

The last step of Algorithm 1 is to check if there is overlapping between clusters (line 6) and merge them according to the *Merge Cluster* algorithm (Alg. 2). Initially the Merge algorithm checks whether or not there are clusters that could be merged. It checks these possibility by calculating the diameter of each cluster, i.e., the difference of the two farthest data streams into the cluster, and then test the diameter against the other ones to identify if there are overlapping between them, as depicted in Fig. 6(a). If the overlapping is detected, those clusters that are in overlapping are cataloged (Alg. 2:line 1). Consequently the respective clusters will be merged if their union does not exceed the maximum standard deviation (Alg. 2:line 3), as presented in Fig. 6(b). Otherwise, it is found the bounds of both clusters i and j (Alg. 2:line 4) by getting the S_i that is closest to the C_j centroid and

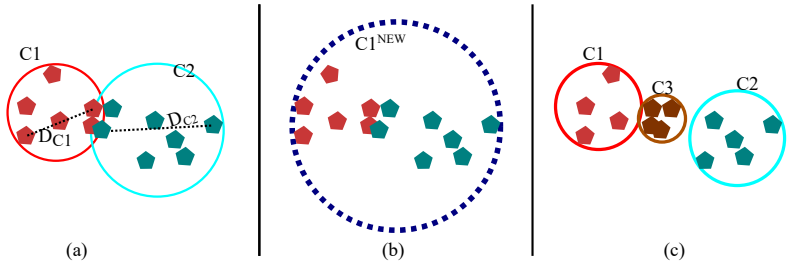


Fig. 6. (a) Overlapping between clusters; (b) Attempt to merge clusters if the standard deviation does not extrapolate the σ_{MAX}; (c) If failed to merge the clusters then a new cluster is created and the streams will remain in their originally cluster or will migrate to the new one.

Algorithm 2: Merge Cluster

Input: The partition P of clusters;
The maximum standard deviation σ_{MAX}.
Output: The partition P of clusters with similar data streams.

```
1 ov[] ← findOverlap(P);
2 foreach pair(i, j) ∈ ov do
3     if merge(i, j) > σ_MAX then
4         findMargin(i, j);
5         C_new ← createNewCluster (ij);
6         checkReAlloc(C_i, C_j, C_ij) ;
7 return P;
```

the S_j that is closest to C_i centroid. Both S_i and S_j will be attached to a new cluster ij (Alg. 2:line 5) and all the remaining data streams in C_i and C_j will be checked if they stay in their own clusters or if they will be associated to the new cluster C_{ij} (Alg. 2:line 6), as depicted in Fig. 6(c).

Thus, at the end of one iteration over the set X, the partition P is composed of clusters containing the similar data streams sources considering the fractal dimension (line 10). As the sources are continuously generating measures, the sliding window shifts forward and, as soon as there are enough data to fulfill the window, a new fractal dimension D_2 is generated and the Mining module is re-invoked. Therefore, the obtained clusters evolves to reflect the changes in the gathered data along the time.

4. EXPERIMENTAL EVALUATION AND RESULTS

As detailed in the Section 3, the eFCDS is a framework to cluster data streams that behave similarly over the time, based on the correlation of their attributes by the calculus of fractal dimension. Thus, in this Section we will dispose the results obtained by our approach.

For the purpose of evaluating our proposal, we applied the following methodology:

—Test of our approach using a real dataset, obtained from climate sensors.
—Analysis of the time consuming to check whether the method is suitable to deal with real-time data streams or not.
—Evaluation of the quality of the resulting clusters by the use of the Silhouette measure.
—Comparison of the eFCDS with baseline methods, previously described in Section 2.1.

By the appliance of the aforementioned methodology, we aim to analyze the following key points:

—Capacity to process and cluster new data in real-time.
—Capacity to stand the evolution of data streams without thereby degrade the quality and cohesion of the obtained clusters.
—Clusters' cohesion in order to check the aptitude of the method to build clusters where the streams are adjoining the cluster centroid.
—Ability of the eFCDS framework to capture similar data streams behavior along the time.
—Quality of the generated clusters against the baseline methods.

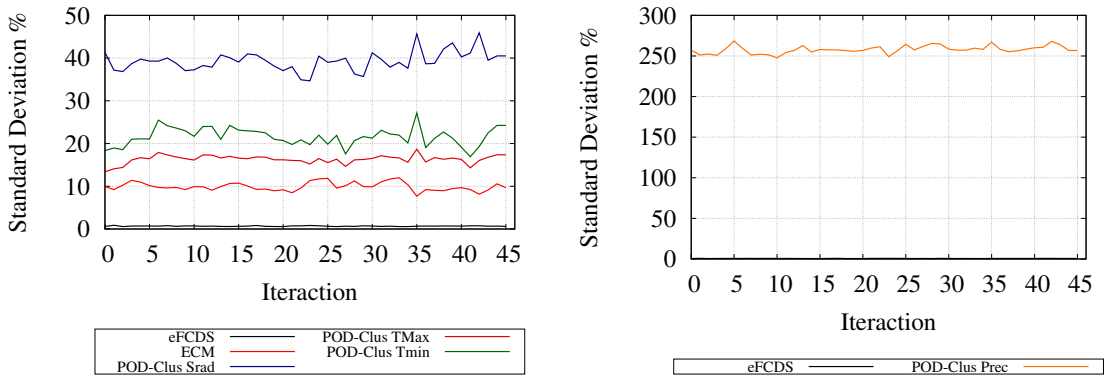
Therefore, in order to evaluate the eFCDS capacity to cluster sensors and cluster new data at a reasonable time the eFCDS was tested with a real dataset composed of 288 climate sensors (sources of data streams) located in different Brazilian regions. Each sensor collects four distinct daily measures (minimum|maximum temperature, precipitation and solar radiation). The considered measures belong to almost all Brazilian's regions, in a period beginning on January 1960 to December 2010. That means 21024000 measures generated by the sensors. This dataset was obtained in cooperation with the Embrapa Agriculture Informatics (Campinas, SP, Brazil) and it was chosen due to the absence of missing data in the evaluated period.

The proposed evaluation was performed in a personal computer running the Ubuntu 14.04 LTS of 64 bits operating systems with 4GB memory, hard drive of 250GB and a processor Intel ® Core ™i7-3770 CPU @ 3.40GHz.

In order to measure clusters' quality, we applied the Silhouette measure. This is a measure commonly employed to evaluated how the elements are disposed in their respective clusters and how many of them could be associated to other clusters. If many streams should be associated to other clusters that means an ill formed clustering. The values of Silhouette ranges from -1 to 1, such that clusters that obtain values above 0.5 are considered of good quality. If the value range from 0 to 0.5 the element is in the right cluster but it is also close to other clusters. Otherwise, if the Silhouette value is below 0, that element should be associated to another cluster. So, the higher is the quantity of elements in the cluster that the Silhouette measure are above 0.5, the better is the cohesion of the respective cluster.

Finally, we compare our results against POD-Clus and ECM methods (both in Section 2.1) once they are techniques presented in the literature aiming at solving the mainly common issues that eFCDS aims, such as: fast data processing; compact representation; traceability of changes; outliers identification. As mentioned in Section 2.1, only POD-Clus supports multivariate data streams, but it measures the dissimilarity of the attributes during the clustering process without considering the correlation among them. Therefore, in order to deal with the correlation among multiple attributes, usually presented in real data, we calculate the fractal dimension for each sensor, thus creating an unidimensional flow to be used as input to ECM. The ECM's and the POD-Clus' as well as the eFCDS's parameters were empirically determined, as follows. For POD-Clus, the sliding windows size equal to 1825 with shift of 365 days; three streams are needed to turn an outlier group into a cluster; the cluster merge threshold was 0.5; and all the others parameters were set with default values. For eFCDS, the maximum standard deviation (σ_{max}) was set to 0.05; the sliding windows size (l) was equal to 1825 days that contain five counting periods t ($t = 5$) and each period has 365 events ($e = 365$), so that, each sliding windows generates one fractal dimension. The ECM's correlation threshold was set to 0.6 and the fractal dimension was obtained in the same way as eFCDS obtained it.

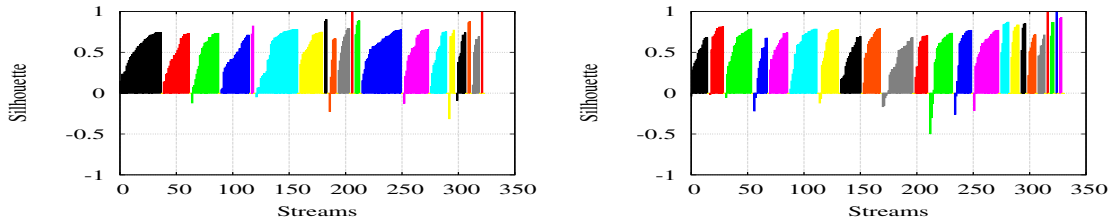
The main results are depicted in Fig. 7. Fig. 7(a) shows the average percentage of the standard deviations for POD-Clus' features, excluding precipitation, ECM and eFCDS fractal dimension. Fig. 7(b) shows the comparative results between eFCDS and POD-Clus precipitation, these result was segregated to better understand the results achieved by eFCDS. Recall that eFCDS deals with fractal dimension values and, as ECM deals with single data flows, the fractal dimension was also its input.



(a) Standard deviation percentage average on each iteration, except POD-Clus precipitation result

(b) Comparative of eFCDs Standard deviation percentage average against POD-Clus precipitation standard deviation percentage.

Fig. 7. Comparative results



(a) Silhouette at first iteration

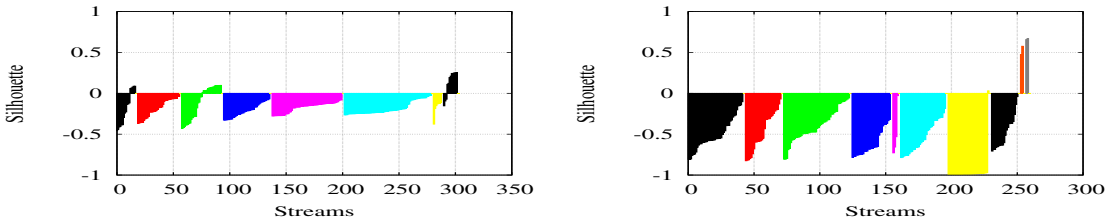
(b) Silhouette at last iteration

Fig. 8. eFCDS' Silhouette results

On the other hand, POD-Clus deals with the original measures of temperature (min & max), precipitation (Prec) and solar radiation (Srad), then to determine the cohesion of the resulting clusters we calculated the percent of standard deviation over the mean of the clusters for each attribute on each iteration. Thus, it is possible to notice that the percentage of standard deviation obtained by eFCDS was near zero, the black line, on both Figs. 7(a) and 7(b), over the zero value on Y axis, indicating high clusters cohesion. The clusters cohesion obtained by ECMs range from $\approx 7.7\%$ to $\approx 12\%$ and POD-Clus ranges from $\approx 13.4\%$ to $\approx 18.7\%$ on minimum temperature, $\approx 16.9\%$ to $\approx 27.1\%$ on maximum temperature, $\approx 34.7\%$ to $\approx 45.9\%$ on solar radiation and $\approx 247.6\%$ to $\approx 268.4\%$ on precipitation. Also notice that the POD-Clus precipitation standard deviation was almost 2.5 times higher than the mean of the data points. This higher standard deviation result is due to the particular characteristics of the rainfall in Brazil, where the rain vary from 1200 to 1500 millimeters a year. As POD-Clus considers the attributes individually (see Equation 3), the higher the standard deviation of the attribute the smaller its influence on the distance measure, affecting the quality of the final clusters.

Quality measures are presented in Figures 8(a) to Fig. 10(h). Fig. 8(a) and Fig. 8(b) show the Silhouette results obtained by the eFCDS technique on first and last iteration, respectively. The eFCDS built clusters with 68% of the data streams Silhouette measures upper to 0.5, indicating that most of the streams are appropriately placed and only 7.6% should be in a different cluster in the last iteration.

Fig. 9(a) and Fig. 9(b) present the Silhouette measures to the clusters computed by the ECM method on first and last iteration, respectively. In this analysis, only three streams in the last iteration are



(a) Silhouette at first iteration

(b) Silhouette at last iteration

Fig. 9. ECM' Silhouette results

confirmed to be included in the correct cluster (Silhouette value above 0.5), indicating that the ECM method produced bad quality clusters when compared to eFCDS.

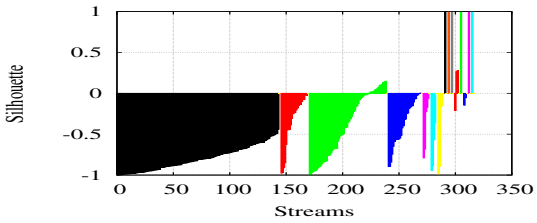
Figures 10(a) to 10(h) depict the Silhouette values of the clusters generated using the POD-Clus method on first and last iteration, to the measures of solar radiation, minimum and maximum temperature, and precipitation. As stated before, POD-Clus deals with each dimension independently and these approach contributed to the bad clusters' formation quality indicated by the analysis of the Silhouette measures of each dimension. The best POD-Clus results was obtained in the maximum temperature dimension in the last iteration, with over 5% upper to 0.5. Although, if we consider all the results over 0 then the best result has obtained in the minimum temperature dimension in the last iteration, with over 20% upper to 0.

By calculating the results depicted in Fig. 7(a) we concluded that the average standard deviation obtained by the eFCDS was nearly 93.5% less than the result of the ECM method and 99.9% regarding to the POD-Clus, i.e., the clusters generated by eFCDS are, approximately, 14 times more compact than the clusters generated by ECM and almost 20 times more compact than the ones generated by POD-Clus. It means that by clustering using eFCDS, a data streams is belonging to a cluster that it probably should be in and there are much more cohesion on each eFCDS' cluster than there are on ECM's and POD-Clus' clusters.

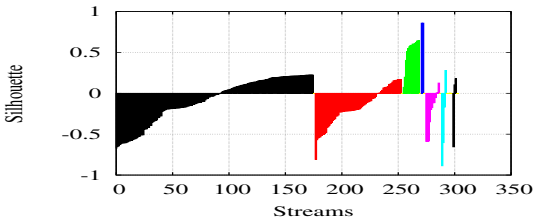
Fig. 11 shows that the processing time spent by eFCDS to process all data was 29.1% of the time spent by the ECM. But as the ECM do not calculate the fractal dimension, the fractal flow was generated apart and, unlike eFCDS, the corresponding cost was not includes in the ECM's processing time. Also the eFCDS' processing time was approximately 5.4% of the time obtained by POD-Clus, as it has to deal with four dimensions independently. Therefore, the eFCDS' consuming time by each datum was nearly 10^{-6} seconds, i.e., it can process approximately 100000 data per second, indeed allowing real time processing.

The experimental studies on climate data have been conducted in collaboration with domain specialists from Embrapa-Campinas (Agriculture Informatics). Empirical evaluations on clustering results have indicated that clustering those streams (sensors) in a dynamic fashion helps to find homogeneous climate regions and also identify changes that are occurring along the time. This information can be useful for: 1) climate change studies; studies of climatic phenomena, such as el niño and la niña, that usually change the regional and global weather patterns and may have negative impact on weather and climate; and 2) It can helps in agricultural planning.

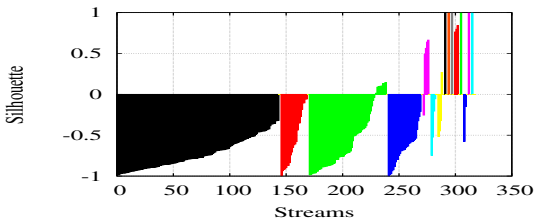
Thus, considering the obtained results, the eFCDS framework better represented the behavior of different data streams along the time. The approach based on fractals showed promising results to deal with multivariate data streams, helping to capture the behavior of climate streams over time with good quality.



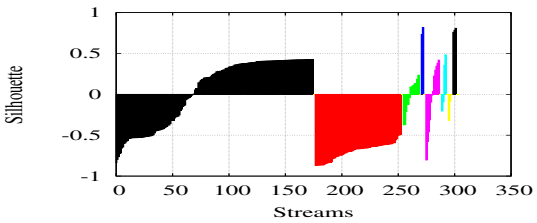
(a) Solar Radiation Silhouette at first iteration



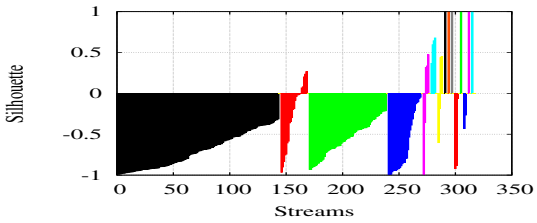
(b) Solar Radiation Silhouette at last iteration



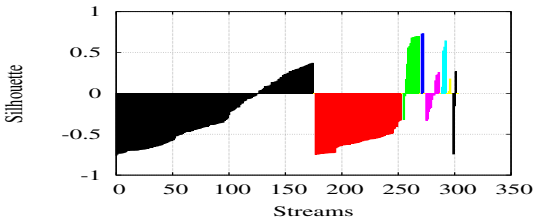
(c) Minimum Temperature Silhouette at first iteration



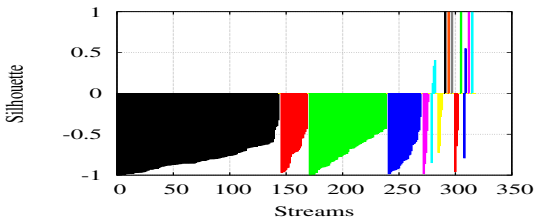
(d) Minimum Temperature Silhouette at last iteration



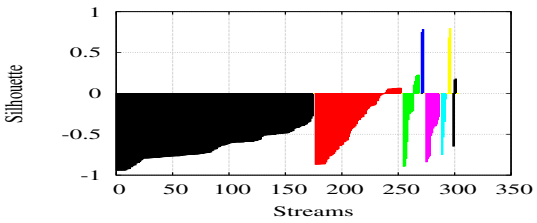
(e) Maximum Temperature Silhouette at first iteration



(f) Maximum Temperature Silhouette at last iteration



(g) Precipitation Silhouette at first iteration



(h) Precipitation Silhouette at last iteration

Fig. 10. POD-Clus' Silhouette results

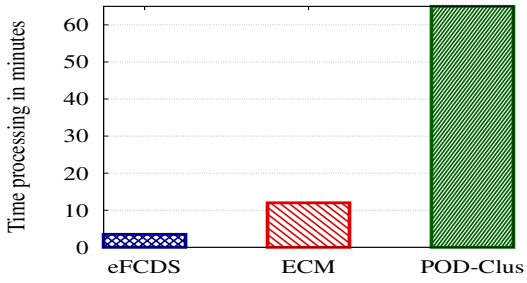


Fig. 11. Time in minutes spent by each method to process all data (21024000 measures).

5. CONCLUSION

Clustering of data streams is one of the most employed approach to analyze data which are potentially endless and evolve over the time. Nevertheless, the literature provides few methods for clustering the entire data streams and most of them often deal with data streams composed of a single attribute or do not apply appropriate strategies for multivariate flows.

In this article we introduced a framework to cluster a set of multivariate data streams considering the fractal correlation among their attributes, also complying the basic requirements to cluster data streams. It also takes into account the behavior of distinct streams along the time. The proposed Evolving Fractal-based Clustering of Data Streams framework is composed of minor modules, each one responsible for a specific processing of the data. The core of the eFCDS framework lies in the computation of the fractal dimension of a piece of the data stream and its subsequent clustering. The use of the fractal dimension allowed to better identify the correlation among the attributes of the data streams. Also, our method performs the clustering of multivariate data streams supporting their evolution in an incremental way, thereby helping to improve the quality of the clusters.

We performed a set of experimental evaluation of the eFCDS framework applied to real climate data and compared it against two recent related methods, ECM and POD-Clus, in order to verify the capacity of representing similar data streams behaviors along the time and keeping the quality of the produced clusters. As a measure of quality, the clusters obtained by eFCDS are much more compact than the clusters generated by ECM and POD-Clus in all the iterations.

Our framework proved to be faster than the two related methods, processing the same amount of data 3 times faster than the ECM and 20 times faster than POD-Clus. It also proves to be an useful tool to assist meteorologists in understanding the climate behavior along a period of time without the necessity to analyze the entire data set manually.

REFERENCES

Ackermann, M. R., Märtens, M., Raupach, C., Swierkot, K., Lammersen, C., and Sohler, C. StreamKM++: A Clustering Algorithm for Data Streams. *ACM Journal of Experimental Algorithmics* 17 (1): 1–30, 2012.

Aggarwal, C. C., Han, J., Wang, J., and Yu, P. S. A Framework for On-demand Classification of Evolving Data Streams. *IEEE Transactions on Knowledge and Data Engineering* 18 (5): 577–589, 2006.

Aggarwal, C. C., Han, J., Wang, J., and Yu, P. S. On Clustering Massive Data Streams: summarization paradigm. In C. C. Aggarwal (Ed.), *Data Streams - Models and Algorithms*. Advances in Database Systems, vol. 31. Springer, pp. 9–38, 2007.

Aroche-Villarruel, A. A., Martínez-Trinidad, J. F., Carrasco-Ochoa, J. A., and Pérez-Suárez, A. A Different Approach for Pruning Micro-clusters in Data Stream Clustering. In J. A. Carrasco-Ochoa, J. F. Martínez-Trinidad, J. H. Sossa-Azuela, J. A. O. López, and F. Famili (Eds.), *Pattern Recognition*. Lecture Notes in Computer Science, vol. 9116. Springer, pp. 33–43, 2015.

Barbará, D. Requirements for Clustering Data Streams. *ACM SIGKDD Explorations Newsletter* 3 (2): 23–27, 2002.

Belussi, A. and Faloutsos, C. Estimating the Selectivity of Spatial Queries Using the 'Correlation' Fractal Dimension. In *Proceedings of the International Conference on Very Large Data Bases*. Zurich, Switzerland., pp. 299–310, 1995.

Bifet, A. and De Francisci Morales, G. Big Data Stream Learning with SAMOA. In *Proceedings of the IEEE International Conference on Data Mining Workshops*. Shenzhen, China, pp. 1199–1202, 2014.

Bones, C. C., Romani, L. A. S., and de Sousa, E. P. M. Clustering Multivariate Climate Data Streams using Fractal Dimension. In *Proceedings of the Brazilian Symposium on Databases*. Petrópolis, Brazil, pp. 41–52, 2015.

Chairukwattana, R., Kangkachit, T., Rakthanmanon, T., and Waiyamai, K. SE-Stream: Dimension Projection for Evolution-based Clustering of High Dimensional Data Streams. In V. N. Huynh, T. Denoeux, D. H. Tran, A. C. Le, and S. B. Pham (Eds.), *Knowledge and Systems Engineering*. Advances in Intelligent Systems and Computing, vol. 245. Springer, pp. 365–376, 2014.

Chaovalit, P. and Gangopadhyay, A. A Method for Clustering Transient Data Streams. In *Proceedings of the ACM Symposium on Applied Computing*. Honolulu, USA, pp. 1518–1519, 2009.

de Sousa, E. P., Traina, A. J., Traina Jr., C., and Faloutsos, C. Measuring Evolving Data Streams Behavior through their Intrinsic Dimension. *New Generation Computing* 25 (1): 33–60, 2007.

Fanaee-T, H. and Gama, J. EigenEvent: An Algorithm for Event Detection from Complex Data Streams in Syndromic Surveillance. *Intelligent Data Analysis* 19 (3): 597–616, 2015.

Faria, E. R., Gonçalves, I. J., de Carvalho, A. C., and Gama, J. Novelty Detection in Data Streams. *Artificial Intelligence Review* 45 (2): 235–269, 2016.

Gama, J. *Knowledge Discovery from Data Streams.* Chapman and Hall, Boca Raton, USA, 2010.

Guha, S., Meyerson, A., Mishra, N., Motwani, R., and O'Callaghan, L. Clustering Data Streams: theory and practice. *IEEE Transactions on Knowledge and Data Engineering* 15 (3): 515–528, 2003.

Kumar, M. and Patel, N. R. Clustering Data with Measurement Errors. *Computational Statistics & Data Analysis* 51 (12): 6084–6101, 2007.

Lughofer, E. and Sayed-Mouchaweh, M. Autonomous Data Stream Clustering Implementing Split-and-merge concepts – towards a plug-and-play approach. *Information Sciences* 304 (1): 54–79, 2015.

Miller, Z., Dickinson, B., Deitrick, W., Hu, W., and Wang, A. H. Twitter Spammer Detection using Data Stream Clustering. *Information Sciences* 260 (1): 64–73, 2014.

Pereira, C. M. M. and de Mello, R. F. TS-stream: Clustering Time Series on Data Streams. *Journal of Intelligent Information Systems* 42 (3): 531–566, 2014.

Rehman, M. Z., Li, T., Yang, Y., and Wang, H. Hyper-ellipsoidal Clustering Technique for Evolving Data Stream. *Knowledge-Based Systems* 70 (1): 3–14, 2014.

Rodrigues, P. P., Gama, J., and Pedroso, J. P. Hierarchical Clustering of Time-series Data Streams. *IEEE Transactions on Knowledge and Data Engineering* 20 (5): 615–627, 2008.

Schroeder, M. R. *Fractals, Chaos, Power Laws: minutes from an infinite paradise.* WH Freeman and Company, New York, USA, 1991.

Sokal, R. R. The Principles of Numerical Taxonomy: twenty-five years later. In M. Goodfellow, D. Jones, and F. G. Priest (Eds.), *Computer-assisted Bacterial Systematics.* Elsevier, Orlando, USA, pp. 1–20, 1985.

Sousa, E. P. M., Traina Jr., C., Traina, A. J. M., Wu, L., and Faloutsos, C. A Fast and Effective Method to Find Correlations among Attributes in Databases. *Data Mining and Knowledge Discovery* 14 (3): 367–407, 2007.

Widiputra, H., Pears, R., and Kasabov, N. Multiple Time-series Prediction through Multiple Time-series Relationships Profiling and Clustered Recurring Trends. In J. Huang, L. Cao, and J. Srivastava (Eds.), *Advances in Knowledge Discovery and Data Mining.* Lecture Notes in Computer Science, vol. 6635. Springer, pp. 161–172, 2011.

Zhang, X., Furtlehner, C., Germain-Renaud, C., and Sebag, M. Data Stream Clustering with Affinity Propagation. *IEEE Transactions on Knowledge and Data Engineering* 26 (7): 1644–1656, 2014.