A single pass trellis-based algorithm for clustering evolving data streams

Simon Malinowski and Ricardo Morla

INESC-TEC, Faculty of Engineering, University of Porto

Abstract. The main paradigm for clustering evolving data streams in the last 10 years has been to divide the clustering process into an online phase that computes and stores detailed statistics about the data in micro-clusters and an offline phase that queries micro-cluster statistics and returns desired clustering structures. The argument for two-phase algorithms is that they support evolving data streams and temporal multi-scale analysis, which single pass algorithms do not. In this paper, we describe a single pass fully online trellis-based algorithm, named ClusTrel, designed for centroid-based clustering that supports evolving data streams and generates clustering structures right after a new point is processed. The performance of ClusTrel is assessed and compared to state of the art algorithms for clustering of data streams showing similar performance with smaller memory footprint.

1 Introduction

The increasing number of sensors and monitoring devices in today's intelligent systems makes data stream mining a topic of current interest. Mining data streams is of use in various application domains such as network management, health monitoring, sports, finance, etc. Amongst the different topics related to data stream mining, clustering the data points from a data stream is the subject of many research works over the last few years [1–9].

Algorithms designed to cluster data streams need to deal with specific additional requirements compared to those designed for offline clustering. These additional requirements make this topic extremely challenging. Systems designed for clustering of data streams should comply with the following requirements:

– The points of the stream have to be processed as they arrive and cannot be stored in memory.
– The stream cannot be interrupted to process the points: there is limited time before the next item arrives.
– An up-to-date clustering of the stream should be maintained to give more importance to recent points.
– Stream clustering algorithms should be capable of detecting noise in the streams.
– No *a priori* information on the number of clusters in the stream. The clustering algorithm must self-adapt to detect the number of clusters.

– The system should be able to deliver a clustering structure frequently. This enables the detection of concept drifts promptly after they occur.

Most of the algorithms proposed in the literature for data stream clustering fulfill the above requirements to different extents. Current algorithms for data stream clustering rely on two different phases: a first phase that maintains sufficient statistics about the stream and a second phase that generates the clustering structure of the stream given the gathered statistics. The first phase is done online and the second one is left for offline whenever a clustering snapshot is required. This offline phase represents an increase in processing time and complexity as it needs to be done every time a clustering snapshot is desired. Because of this increase in time and complexity, the frequency of the offline process is typically much smaller than that of the incoming data points, which introduces delay in the detection of concept changes. Slow reaction to changes in the clustering structure can prove to be detrimental to service quality and security in application domains such as network anomaly and intrusion detection [10].

In this paper, we present ClusTrel, a single pass fully online trellis-based algorithm designed for centroid-based clustering of data streams. Thanks to its trellis structure ClusTrel updates the clustering snapshot after each point and can hence detect concept changes for every incoming data point. This algorithm is also capable of selecting the number of clusters based on a clustering evaluation index. We use the MOA software [11] to generate synthetic streams and compare the performance of ClusTrel with state of the art algorithms for clustering data streams. Simulation results show that ClusTrel is able to reach similar performance while reducing the number of micro-clusters stored in memory.

The remainder of this paper is structured as follows. Related work is surveyed in Section 2, the ClusTrel algorithm is described in Section 3. Simulation results on synthetic and real streams are given in Section 4 and conclusions are drawn in Section 5.

2 Related work

The initial approaches for clustering data streams focused on supporting the single pass requirements [1, 3]. A common pitfall of these approaches is that new points have the same weight as old points, making it difficult to adapt to changes in the stream. These approaches also require that the target number of clusters is provided as input to the clustering algorithm, which is an obvious limitation in the case of evolving data streams.

Most recent two-phase algorithms for clustering data streams rely on micro-clusters [2, 4–7]. Micro-clusters are a compact representation of clusters that maintain sufficient statistics that can be updated online. Creating and indexing micro-clusters from the data stream is established as the online part of most of these algorithms, while the offline process generates the final clustering structure from the current set of micro-clusters.

The CluStream [2] algorithm keeps a fixed maximum number of micro-clusters at each instant. New points that fall within the boundary of already existing

micro-clusters are appended to them, while others form new micro-clusters. This avoids having to merge clusters at every step and can be implemented using micro-cluster feature vectors. CluStream also allows for the deletion of old micro-clusters and can store snapshots of the current clustering structure at different time horizons. The offline process of CluStream takes as input the desired number of clusters and uses a k-means algorithm to generate the final clustering from the current set of micro-clusters. ClusTree algorithm [7] uses an R^*-tree structure in order to index the micro-clusters that are updated online. The different levels of the tree build a hierarchical representation of the clustering. The authors propose different strategies to insert the micro-clusters into the tree. ClusTree can hence handle slow to very fast stream by adapting the way micro-clusters are inserted in the tree. The LiarTree algorithm [9] extends the concepts of ClusTree to cater for noise detection and the improvement of novelty detection in the streams.
Alternative approaches use density-based clustering such as those in [5, 6, 8]. The algorithms presented in [5] and [6] are also based on an online and an offline phase. In [5], the online phase consists of creating and updating micro-clusters, and separating outlier micro-clusters from core micro-clusters. A density-based clustering strategy (e.g. DBSCAN [12]) is used as the offline phase to generate the final clustering. The online part of D-Stream [6] is based on density grids. In [8], the authors propose a single pass density based clustering approach named *FlockStream* that makes use of so-called agents.
The ClusTrel algorithm proposed here is a single pass algorithm similar to *FlockStream* but is a centroid-based approach. The state of the art algorithms for centroid-based clustering of data streams are CluStream [2] and ClusTree [7]. The approach proposed in this paper differs from CluStream and ClusTree as it is a single pass algorithm that gathers statistics from the stream and generates a macro-clustering after every incoming point.

3 The ClusTrel algorithm

3.1 Preliminary notations

Let $S = S_1, \ldots, S_n, \ldots$ be a stream taking its values in $\mathbb{R}^d$. We assume that our processing system does not have enough memory to store all the points of the stream.

Definition 1. *A micro-cluster is a compact representation of a cluster. It is defined by a cluster feature vector (CFV). In this paper, a CFV of a micro-cluster is a $(d+2)$ tuple $(n, c, ssqd)$, where n is the number of points associated to the cluster, c is its centroid and $ssqd$ is the sum of the squared distances of all points in the cluster to c.*

The CFV of a micro-cluster enables the gathering of sufficient statistics about the points of the stream assigned to that micro-cluster without the need for storing all points in memory. We will see later how the CFVs are updated when new points are inserted in micro-clusters.

Definition 2. *We denote by $\mathcal{C}_k^{(t)}$ a clustering structure that represents the clustering of the stream S up to sample S_t into k micro-clusters. The k micro-clusters are represented by their CFVs.*

ClusTrel uses a clustering evaluation measure named the MDB index. It is based on the Davies-Bouldin index [13] and is defined as follows:

Definition 3. *Let $\mathcal{C}_k^{(t)}$ be a clustering structure. The MDB index of $\mathcal{C}_k^{(t)}$ is defined as :*

$$MDB = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \frac{SSQ(C_i) + SSQ(C_j)}{dist(C_i, C_j)^2}, \tag{1}$$

where $SSQ(C_i)$ is the average squared distance of all points in cluster C_i to its centroid and $dist(C_i, C_j)$ is the distance between the centroids of C_i and C_j.

The advantage of the MDB index over the classical DB index is that it is computable online from the CFVs of a clustering structure. Low values of the MDB index are associated with clustering structures composed of compact and well-separated clusters. The MDB is used in ClusTrel to determine the number of clusters that is most adapted to the input stream.

ClusTrel also uses the average SSQ index as a measure of quality of a clustering structure. The average SSQ index is simply the average square distance of the points of the stream to the centroid of the cluster they are assigned to.

3.2 The ClusTrel algorithm

ClusTrel is a dynamic programming algorithm, based on the Viterbi algorithm [14]. Given the data stream, ClusTrel minimizes one of the two cluster evaluation indices described above. It dynamically builds a trellis whose states are clustering structures, as shown in Figure 1 where the horizontal axis represents time. The two parameters n_m and n_M represent the minimum and maximum numbers of clusters to be explored by ClusTrel to find a clustering structure for the stream S.

ClusTrel considers three kinds of transition in the trellis from a given clustering structure, as shown in Figure 1. We denote these by *Inc*, *Dec*, and *Keep* transitions. Each transition generates a new clustering structure given a current structure and an incoming point. An *Inc* transition generates a structure with an additional cluster, a *Dec* transition generates a structure with one less cluster, while the structure generated by a *Keep* transition has the same number of clusters as the input one. These three transitions incorporate the incoming point in a structure and modify the CFVs that need to be updated. With a clustering structure of k clusters and an incoming point p as input, the three transitions of ClusTrel are detailed in the following.

***Inc* transition.** An *Inc* transition produces a clustering structure of $k+1$ clusters updated with p. It generates a new cluster on the point p. The CFVs of the k already existing clusters are unchanged and the CFV of the newly created cluster is simply $(1, p, 0)$.

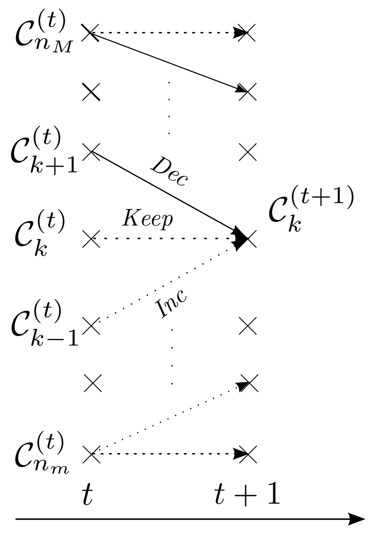
$\mathcal{C}_{n_M}^{(t)}$
$\mathcal{C}_{k+1}^{(t)}$
$\mathcal{C}_{k}^{(t)}$
$\mathcal{C}_{k-1}^{(t)}$
$\mathcal{C}_{n_m}^{(t)}$
$\mathcal{C}_{k}^{(t+1)}$
Dec
Keep
Inc
t
$t+1$

Fig. 1. The trellis structure of ClusTrel. The states of the trellis are clustering structures with a number of clusters varying from n_m to n_M. The three kinds of arrows represent the possible transitions between structures at two consecutive time stamps.

Keep **transition.** A *Keep* transition generates a clustering structure with k clusters. It first finds the cluster in the structure whose centroid is the closest to p, denoted $\overline{C}$. The point p is then appended into $\overline{C}$. Only the CFV of $\overline{C}$ needs to be updated. If this CFV is equal to (n, c, s), it is updated into (n', c', s'), where

$$\begin{cases} n' = n + 1, \\ c' = (p + n \times c)/n', \\ s' = s + dist(p, c')^2 + n \times dist(c, c')^2, \end{cases} \tag{2}$$

where *dist* represents the Euclidean distance.

Dec **transition.** A *Dec* transition generates a clustering structure of $k-1$ clusters. It first merges two clusters C_i and C_j of the current clustering structure. We denote the CFVs of these clusters (n_i, c_i, s_i) and (n_j, c_j, s_j) respectively. C_i and C_j are merged into C_i' whose CFV (n_i', c_i', s_i') is equal to

$$\begin{cases} n_i' = n_i + n_j, \\ c_i' = (n_i \times c_i + n_j \times c_j)/n_i', \\ s_i' = s_i + s_j + n_i \times dist(c_i, c_i')^2 + n_j \times dist(c_j, c_i')^2. \end{cases} \tag{3}$$

It then incorporates p into the closest cluster of the structure and updates the appropriate CFV according to Eqn(2). Performing a *Dec* transition on a clustering structure $\mathcal{C}_k^{(t)}$ is equivalent to selecting a pair of clusters $(C_i, C_j), i \neq j, 1 \leq$

$i, j \leq k$ to merge. We propose here three different ways to choose the two clusters to be merged in a *Dec* transition. The corresponding transitions are denoted *Dec1*, *Dec2* and *Dec3* in the following.

Dec1 **transition.** The *Dec1* transition selects the clusters (C_i, C_j) that are the closest in $\mathcal{C}_k^{(t)}$.

Dec2 **transition.** For every cluster C_i in $\mathcal{C}_k^{(t)}$, the closest cluster to C_i is searched for. This cluster is denoted $\overline{C}_i$. Merging C_i and $\overline{C}_i$ leads to a new clustering structure whose average SSQ sq_i can be computed according to Eqn(3). The pair of clusters to merge is chosen as $(C_j, \overline{C_j})$ such that $j = \arg\min_{1 \leq i \leq k} sq_i$.

Dec3 **transition.** For the *Dec3* transition, the pair (C_i, C_j) is chosen as the one that minimizes the SSQ index over all possible pairs of clusters in the given clustering structure.

The processing time associated with the three *Dec* transitions described above increases from *Dec1* to *Dec3*. We will see in the experimental results section that *Dec2* and *Dec3* transitions can lead to better clustering performance, but also that the gain brought by *Dec3* over *Dec2* is small, so that *Dec2* seems to be a good trade-off between performance and complexity. In the following, ClusTrel-1, ClusTrel-2 and ClusTrel-3 will refer to ClusTrel used respectively with the *Dec1*, *Dec2* and *Dec3* transitions.

Initialization step of ClusTrel. The initialization step of ClusTrel consists of generating the first clustering structure of the trellis: $\mathcal{C}_{n_m}^{(n_m)}$ which is composed of n_m clusters of 1 point, hence taking into account the first n_m points of the stream S. All the others structures in the same vertical slice of the trellis are empty.

Updating step of ClusTrel. Let us now assume that the t first points of the stream are already processed and that the trellis is filled with the clustering structures $\mathcal{C}_{n_m}^{(t)}, \mathcal{C}_{n_m+1}^{(t)}, \ldots, \mathcal{C}_{n_M}^{(t)}$. Given the incoming point S_{t+1}, ClusTrel updates these $n_M - n_m + 1$ structures. As can be seen in Figure 1, a structure at time $(t+1)$ can be updated from at most three different structures at time (t) (only two if the state is at the top or bottom of the trellis). ClusTrel algorithm computes $\mathcal{C}_k^{(t+1)}$ by choosing the best clustering structure (in terms of a cluster evaluation index) that ends up in state $\mathcal{C}_k^{(t+1)}$ from the previous slice of the trellis. In other words,

$$\forall k \in [n_m, n_M], \mathcal{C}_k^{(t)} = \arg\min(Inc(\mathcal{C}_{k-1}^{(t-1)}), Dec(\mathcal{C}_{k+1}^{(t-1)}), Keep(\mathcal{C}_k^{(t-1)})), \quad (4)$$

where the arg min is taken in terms of the desired cluster evaluation index (MDB or SSQ). This step performs a local minimization of the index of the clustering structures in the trellis given the data stream. In the following, we use ClusTrel together with the minimization of the SSQ index.

For memory purposes, only the latest structures are kept in memory. Updating the structures at time $t+1$ only needs the incoming point and the structures at time t. Hence, the maximum number of CFVs stored in a slice of the trellis is equal to $1/2 \times (n_M + n_m)(n_M - n_m + 1)$. However, some micro-clusters can appear in different clustering structures. In order to reduce the amount of memory needed, a list of unique micro-clusters is kept by the ClusTrel algorithm. These micro-clusters are indexed, and the clustering structures $\mathcal{C}_{n_m}^{(t)}, \ldots, \mathcal{C}_{n_M}^{(t)}$ are described by the indexes of the micro-clusters that compose each structure.

Selection of the final clustering structure. Whenever a snapshot of the current clustering is desired, ClusTrel can output the clustering structure in the trellis with the most adapted number of clusters, w.r.t. to the MDB index. This index is used for the selection of the best clustering structure as it gives a compromise between compact and well-separated clusters, which the SSQ index does not. The MDB index of the structures in the trellis is calculated online without the need for further information. The different values of the MDB index in the trellis can also give soft information about the different clustering structures that might be interesting to consider for the given stream.

3.3 Summary of the ClusTrel features

ClusTrel is a single pass centroid-based clustering algorithm that does not require any offline process to deliver the final clustering result. ClusTrel does not need the number of clusters to search for as an input parameter as it searches for the best clustering structure with a number of clusters in $[n_m, n_M]$. In addition, it is able to output the best clustering structure (in terms of one of two cluster evaluation indices) at any time, and without the need for further processing. The selection and output of the best clustering structure is inherent to ClusTrel.

As far as memory is concerned, at most $1/2 \times (n_M + n_m)(n_M - n_m + 1)$ micro-clusters are kept in memory after the processing of an incoming point. For each of these micro-clusters, a CF vector is stored. An important point is that the maximum number of clusters n_M does not need to be much higher than the order of magnitude of the real number of clusters in the data, as will be shown in the simulation results section.

ClusTrel is able to detect changes in concept in the stream when the selected number of clusters evolves in a certain period a time. The fact that ClusTrel can generate clustering structures at every time instant make this detection faster. For two-phase algorithms, the offline process has to be executed frequently in order to quickly detect changes in concept in the data stream, which leads to a larger complexity of the overall system. The ClusTrel algorithm can also deal with up-to-date clustering by incorporating the concept of weighing down old points with a decay function (as in [4–8]).

As far as processing time is concerned, ClusTrel is more demanding than CluStream, DenStream or ClusTree, as it is a single pass algorithm that performs the statistics gathering phase and the macro-clustering phase in parallel. This algorithm may not be designed for very fast streams but is more adapted to applications where the focus is put on the accuracy of clustering and on fast detection of concept changes.

4 Experimental results

We present some experimental results to assess the performance of ClusTrel with synthetic streams and streams from a real data set [15]. We focus here on the comparison between the clustering performance of ClusTrel and the ones of the two state of the art algorithms for centroid-based data stream clustering: CluStream [2] and ClusTree [7].

4.1 Using synthetic streams

The synthetic data streams used in this section were generated using the MOA software [11]. Three different streams with the following features are used:

1. 5 kernels in 2-dimensional space, 10,000 points
2. 8 kernels in 4-dimensional space, 12,000 points
3. 12 kernels in 6-dimensional space, 18,000 points.

The kernels that compose the streams are all Gaussian kernels. The radii of these Gaussian kernels is a parameter of the MOA stream generator. We have chosen values of the radii so that some of the kernels overlap in the space. The results presented for ClusTree and CluStream in this section are obtained with the MOA software that is freely available[1]. For all of these experiments, the correct number of clusters in the streams is detected by ClusTrel by looking at the MDB index of the clustering structures in the trellis. Hence, the SSQ indices given in this section always refer to clustering structures with as many clusters as the number of kernels in the stream.

Figure 2-(a) shows the clustering performance for the first stream in terms of the average SSQ index of ClusTree for different number of levels, and of CluStream for different numbers of micro-clusters kept at each time instant. The performance of ClusTrel-1 is also depicted as the horizontal black curve and is obtained for $n_M = 6$. For this stream, the clustering performance obtained with ClusTrel-1, ClusTrel-2 and ClusTrel-3 are the same. The best SSQ indices reached by ClusTree and CluStream are given by the last value of their respective curves. It can be seen that CluStream reaches the same performance as ClusTrel when 85 clusters are kept at each instant. This result highlights the benefit of the trellis structure of ClusTrel. The optimization of the clustering made at each time instant allows a decrease in the number of clusters (21 micro-clusters) that need to be kept in memory in order to obtain good performance.

Similar conclusions can be drawn when the second stream is used. The corresponding results are shown in Figure 2-(b). The performance of ClusTrel is again the same for the 3 variants of ClusTrel. The horizontal line represents the SSQ obtained by ClusTrel for $n_M = 10$. The best average SSQ obtained with ClusTree is always higher than the one obtained with ClusTrel. CluStream with 75 micro-clusters reaches the same performance as ClusTrel with 55 micro-clusters maximum.

[1] http://moa.cs.waikato.ac.nz

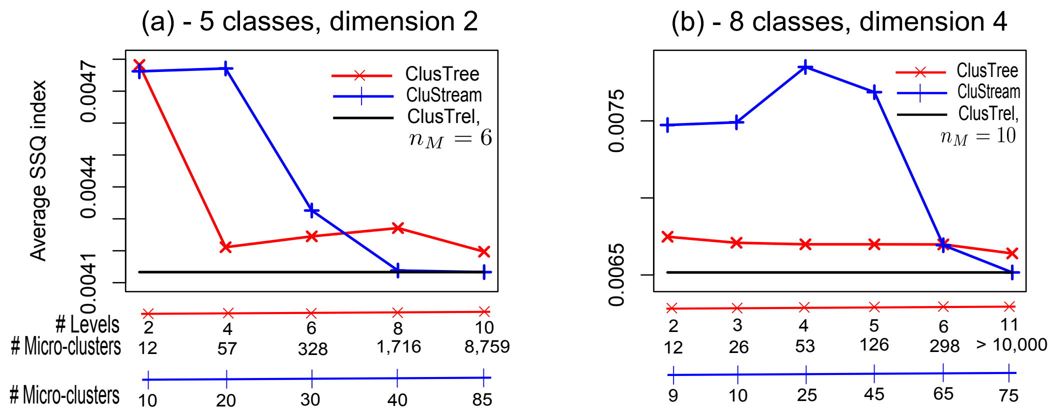

Fig. 2. Average SSQ index obtained by ClusTrel, ClusTree and CluStream for the first two synthetic streams.

Table 1. Average SSQ index obtained by ClusTree and CluStream for the third synthetic stream. The performance of ClusTrel for $n_m = 15$ is the same as the one of CluStream with 120 micro-clusters.

ClusTree [7]					
# Levels	4	5	6	7	12
# Micro-clusters	56	133	331	677	> 10,000
SSQ index	0.018456	0.016093	0.016068	0.016035	0.015817
CluStream [2]					
# Micro-clusters	20	30	40	50	120
SSQ index	0.017162	0.016059	0.015349	0.015348	0.015347

The clustering performance of ClusTree and CluStream on the third stream are given in Table 1. For this stream, the average SSQ index obtained by ClusTrel with $n_M = 15$ (120 micro-clusters maximum) is equal to 0.015347, the same value obtained using CluStream with 120 micro-clusters.

These experiments made on synthetic streams highlight the benefit of the trellis structure of ClusTrel w.r.t. the maximum number of clusters that need to be kept in memory. Even when the height of the trellis is close to the real number of classes in the stream, ClusTrel is able to generate a clustering with a good SSQ index compared to the ones obtained by ClusTree and CluStream. In the following, we assess the performance of ClusTrel with a stream coming from a real data set and highlight the impact of ClusTrel-2 and ClusTrel-3 on the clustering performance.

4.2 Using real data

We use the Forest Covertype data set available from [15] to assess the performance of ClusTrel with real data streams. This data set is composed of approximately 500,000 10-dimensional points (only the numerical attributes are considered). The points are labeled with an integer from 1 to 7 corresponding to 7 different classes. Experimental results are given here in terms of the average SSQ index together with the widely-used purity index, defined as follows. For a set of classes $C = \{c_1, \ldots, c_L\}$ and a set of clusters $\mathit{\Gamma} = \{\gamma_1, \ldots, \gamma_K\}$, the purity index of the clustering is equal to

$$purity(C, \mathit{\Gamma}) = \frac{1}{N} \sum_k \max_l \big(\mathrm{card}(\gamma_k \cap c_l)\big), \tag{5}$$

where N is the total number of points. This index looks at how the different classes are distributed amongst the clusters. The closer the purity is to 1, the better the distribution of classes in the clusters.

We built a first stream composed of 100,000 points from the Forest Covertype data set. The proportion of the classes in this stream is similar to those in the whole data set.

The average SSQ and purity indices obtained by ClusTrel, ClusTree and CluStream for different parameters are given in Figure 3-(a) and (b) respectively. The performance obtained by the three variants of ClusTrel is given in this figure. The SSQ and purity index are calculated on a clustering structure of 7 clusters output by these algorithms (not at the micro-clustering level). The results on this figure demonstrate that the use of ClusTrel-2 and ClusTrel-3 brings an improvement in terms of clustering performance. The SSQ index obtained with ClusTrel-2 and ClusTrel-3 is almost always less or equal than the one of ClusTrel-1. The best performance is reached for this stream using ClusTrel-3 and $n_M = 11$.

Fig 3-(b) shows that the purity index obtained by ClusTrel is higher than the ones obtained with ClusTree and CluStream for the different set of parameters.

Regarding the difference between the performance obtained by ClusTrel-2 and ClusTrel-3, it is important to note that they are equal or very close almost every time on this experiment. They are exactly equal for $n_M = 8, 9, 10$ and 12. The difference between the two is very close for $n_M = 13$ (it cannot actually be seen on the figure, but there is a less than a 0.1% difference). The benefit of ClusTrel-3 is only significant for $n_M = 11$ here. ClusTrel-1 and ClusTrel-2 hence seems to provide a good trade-off between processing complexity and clustering performance. The results given in Table 2 also support this observation. These average SSQ indices are obtained on another stream of 75,000 points from the Forest CoverType data set. It can be seen that the increase of the n_M parameter does not improve the clustering performance of ClusTrel-1 while it does for ClusTrel-2 and ClusTrel-3. The gain obtained with ClusTrel-3 over ClusTrel-2 is again small.

5 Conclusion

We propose in this paper ClusTrel, a single pass algorithm designed for the clustering of data streams. Clustering structures with different numbers of clusters and their statistics are maintained while processing the stream. These structures are chosen so that they minimize a cluster evaluation index at every step.

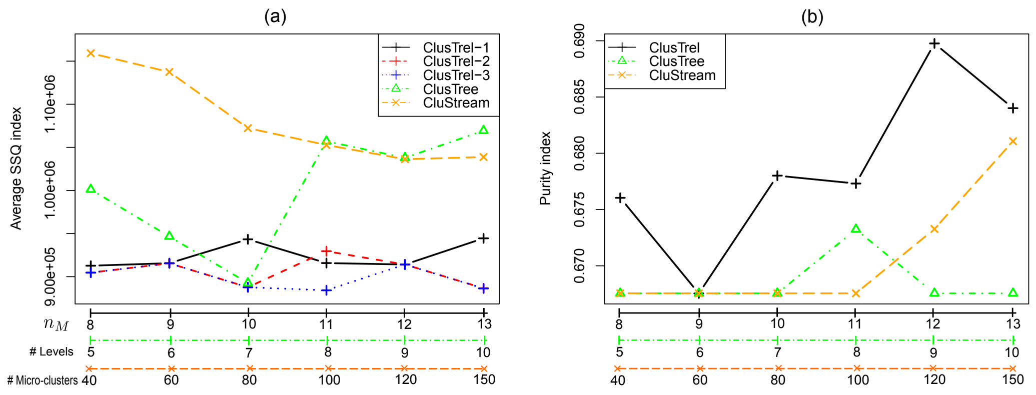

Fig. 3. Average SSQ index (a) and purity index (b) obtained by ClusTrel, ClusTree and CluStream for different parameters on a stream obtained from the Forest CoverType data set.

Table 2. Average SSQ index obtained with the three variants of ClusTrel on a second stream based on the Forest CoverType data set.

n_M	12	24	36
ClusTrel-1	$8.640e+5$	$8.720e+5$	$8.860e+5$
ClusTrel-2	$8.630e+5$	$8.564e+5$	$8.4884e+5$
ClusTrel-3	$8.630e+5$	$8.564e+5$	$8.4882e+5$

ClusTrel is able to deliver a clustering of the stream as soon as a new point is processed. Thanks to the structure of ClusTrel, changes in concept in the streams can be detected quickly without the need for further processing. The performance of ClusTrel has been compared to two state of the art algorithms for clustering data streams. Simulation results show that ClusTrel is competitive with these algorithms in terms of clustering performance while enabling a reduction in the number of clusters that need to be stored in memory.

Acknowledgments. This work is financed by the ERDF European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT Fundao para a Cincia e a Tecnologia (Portuguese Foundation for Science and Technology) within project CMU-PT/RNQ/0029/2009

References

1. L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering. In *Proc. of Intl. Conf. on Data Engineering*, pages 685 –694, 2002.

2. C.C. Aggarwal, J. Han, J. Wang, and P.S. Yu. A framework for clustering evolving data streams. In *VLDB*, pages 81–92, 2003.
3. S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *Knowledge and Data Engineering, IEEE Transactions on*, 15(3):515 – 528, may-june 2003.
4. C.C. Aggarwal, J. Han, J. Wang, and P.S. Yu. A framework for projected clustering of high dimensional data streams. In *Proc. of the Intl. Conf. on Very large data bases*, pages 852–863, 2004.
5. F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *In 2006 SIAM Conference on Data Mining*, pages 328–339, 2006.
6. Y. Chen and L. Tu. Density-based clustering for real-time stream data. In *Proc. of ACM SIGKDD Intl. Conf. on Knowledge discovery and data mining*, pages 133–142, 2007.
7. P. Kranen, I. Assent, C. Baldauf, and T. Seidl. The ClusTree: indexing micro-clusters for anytime stream mining. *Knowledge and Information Systems*, pages 1–24, 2010.
8. A. Forestiero, C. Pizzuti, and G. Spezzano. A single pass algorithm for clustering evolving data streams based on swarm intelligence. *Data Mining and Knowledge Discovery*, pages 1–26, 2011.
9. M. Hassani, P. Kranen, and T. Seidl. Precise anytime clustering of noisy sensor data with logarithmic complexity. In *Proc. of International Workshop on Knowledge Discovery from Sensor Data*, pages 52–60, 2011.
10. G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th conference on Security symposium*, pages 139–154, 2008.
11. A. Bifet, G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, and T. Seidl. MOA : Massive Online Analysis , a Framework for Stream Classification and Clustering . *Journal of Machine Learning Research*, pages 3–16, 2011.
12. M. Ester, H-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of ACM SIGKDD Intl. Conf. on Knowledge discovery and data mining*, pages 226–231, 1996.
13. D.L. Davies and D.W. Bouldin. A cluster separation measure. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227, April 1979.
14. A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. on Information Theory*, 13(2):260–269, April 1967.
15. A. Frank and A. Asuncion. UCI machine learning repository, 2010. http://archive.ics.uci.edu/ml.