9-2004

SCLOPE: An algorithm for clustering data streams of categorical attributes

Kok-Leong ONG

Wenyuan LI

Wee-Keong NG

Ee Peng LIM

Singapore Management University, eplim@smu.edu.sg

DOI: https://doi.org/10.1007/978-3-540-30076-2_21

SCLOPE: An Algorithm for Clustering Data Streams of Categorical Attributes

Kok-Leong Ong[1], Wenyuan Li[2], Wee-Keong Ng[2], and Ee-Peng Lim[2]

[1] School of Information Technology, Deakin University
Waurn Ponds, Victoria 3217, Australia
leong@deakin.edu.au
[2] Nanyang Technological University, Centre for Advanced Information Systems
Nanyang Avenue, N4-B3C-14, Singapore 639798
liwy@pmail.ntu.edu.sg, {awkng,aseplimg}@ntu.edu.sg

Abstract. Clustering is a difficult problem especially when we consider the task in the context of a data stream of categorical attributes. In this paper, we propose SCLOPE, a novel algorithm based on CLOPE's intuitive observation about cluster histograms. Unlike CLOPE however, our algorithm is very fast and operates within the constraints of a data stream environment. In particular, we designed SCLOPE according to the recent CluStream framework. Our evaluation of SCLOPE shows very promising results. It consistently outperforms CLOPE in speed and scalability tests on our data sets while maintaining high cluster purity; it also supports cluster analysis that other algorithms in its class do not.

1 Introduction

In recent years, the data in many organizations take the form of continuous streams, rather than finite stored data sets. This possesses a challenge for data mining, and motivates a new class of problem called *data streams* [1, 2]. Designing algorithms for data streams is a challenging task: (a) there is a sequential one-pass constraint on the access of the data; (b) and it must work under bounded (i.e., fixed) memory with respect to the data stream.

Also, the continuity of data streams motivates time-sensitive data mining queries that many existing algorithms do not adequately support. For example, an analyst may want to compare the clusters, found in one window of the stream, with clusters found in another window of the same stream. Or, an analyst may be interested in finding out how a particular cluster evolves over the lifetime of the stream. Hence, there is an increasing interest to revisit data mining problems in the context of this new model and application.

In this paper, we study the problem of clustering a data stream of categorical attributes. Data streams of such nature, e.g., transactions, database records, Web logs, etc., are becoming common in many organizations [3]. Yet, clustering a categorical data stream remains a difficult problem. Besides the dimensionality and sparsity issue inherent in categorical data sets, there are now additional

Y. Kambayashi et al. (Eds.): DaWaK 2004, LNCS 3181, pp. 209–218, 2004.

stream-related constraints. Our contribution towards this problem is the SCLOPE algorithm inspired by two recent works: the CluStream [4] framework, and the CLOPE [3] algorithm.

We adopted two aspects of the CluStream framework. The first is the *pyramidal* timeframe, which stores summary statistics at different time periods at different levels of granularity. Therefore, as data in the stream becomes outdated, its summary statistics looses details. This method of organization provides an efficient trade-off between the storage requirements and the quality of clusters from different time horizons. At the same time, it also facilities the answering of time-sensitive queries posed by the analyst.

The other concept we borrowed from CluStream, is to separate the process of clustering into an *online* micro-clustering component and an *offline* macro-clustering component. While the online component is responsible for efficient gathering of summary statistics (a.k.a *cluster features* [4]), the offline component is responsible for using them (with the user inputs) to produce the different clustering results. Since the offline component does not require access to the stream, this process is very efficient.

Set in the above framework, we report the design of the online and offline components for clustering categorical data organized within a pyramidal timeframe. We begin with the online component, where we propose an algorithm to gather the required statistics in one sequential scan of the data. Using an observation in the FP-Tree [5], we eliminated the need to evaluate the clustering criterion. This dramatically drops the cost of processing each record, and allows it to keep up with the high data arrival rate.

We then discuss the offline component, where we based its algorithmic design on CLOPE. We were attracted to CLOPE because of its good performance and accuracy in clustering large categorical data sets, i.e., when compared to k-means, CLARANS [6], ROCK [7], and LargeItem [8]. More importantly, its clustering criterion is based on *cluster histograms*, which can be constructed quickly and accurately (directly from the FP-Tree) within the constraints of a data stream environment.

2 Maintenance of Summary Statistics

For ease of discussion, we assume that the reader are familiar with the CluStream framework, the CLOPE algorithm, and the FP-Tree [5] structure. Also, without loss of generality, we define our clustering problem as follows. A data stream $\mathcal{D}$ is a set of records $\mathcal{R}_1, \ldots, \mathcal{R}_i, \ldots$ arriving at time periods $t_1, \ldots, t_i, \ldots$, such that each record $\mathcal{R} \in \mathcal{D}$ is a vector containing attributes drawn from $\mathcal{A} = \{a_1, \ldots, a_j\}$. A clustering $\mathcal{C}_1, \ldots, \mathcal{C}_k$ on $\mathcal{D}_{(t_p,t_q)}$ is therefore a partition of records $\mathcal{R}_x, \mathcal{R}_y, \ldots$ seen between t_p and t_q (inclusive), such that $\mathcal{C}_1 \cup \ldots \cup \mathcal{C}_k = \mathcal{D}_{(t_p,t_q)}$ and $\mathcal{C}_\alpha \neq \varnothing$ and $\forall \alpha, \beta \in [1, k]$, and $\mathcal{C}_\alpha \cap \mathcal{C}_\beta = \varnothing$.

From the above, we note that clustering is performed on all records seen in a given time window specified by t_p and t_q. To achieve this without accessing the stream (i.e., during offline analysis), the online micro-clustering component

has to maintain sufficient statistics about the data stream. Summary statistics, in this case, is an attractive solution because they have a much lower space requirement than the stream itself. In SCLOPE, they come in the form of micro-clusters and cluster histograms. We define them as follows.

Definition 1 (Micro-Clusters). *A micro-cluster $\mu^{\mathcal{C}}$ for a set of records $\mathcal{R}_x$, $\mathcal{R}_y, \ldots$ with time stamps $t_x, t_y, \ldots$ is a tuple $\langle \overline{L}, \overline{H} \rangle$, where $\overline{L}$ is a vector of record identifiers, and $\overline{H}$ is its cluster histogram.*

Definition 2 (Cluster Histogram). *The cluster histogram $\overline{H}$ of a micro-cluster $\mu^{\mathcal{C}}$ is a vector containing the frequency distributions $\texttt{freq}(a_1, \mu^{\mathcal{C}}), \ldots, \texttt{freq}(a_{|\mathcal{A}|}, \mu^{\mathcal{C}})$ of all attributes $a_1, \ldots, a_{|\mathcal{A}|}$ in $\mu^{\mathcal{C}}$, In addition, we define the following derivable properties of $\overline{H}$:*

- *the **width**, defined as $|\{a : \texttt{freq}(a, \mu^{\mathcal{C}}) > 0\}|$, is the number of distinct attributes, whose frequency in $\mu^{\mathcal{C}}$ is not zero.*
- *the **size**, defined as $\sum_{i=1}^{|\mathcal{A}|} \texttt{freq}(a_i, \mu^{\mathcal{C}})$, is the sum of the frequency of every attribute in $\mu^{\mathcal{C}}$.*
- *the **height**, defined as $\sum_{i=1}^{|\mathcal{A}|} \texttt{freq}(a_i, \mu^{\mathcal{C}}) \times |\{a : \texttt{freq}(a, \mu^{\mathcal{C}}) > 0\}|^{-1}$, is the ratio between the size and width of $\overline{H}$.*

2.1 Algorithm Design

We begin by introducing a simple example. Consider a data stream $\mathcal{D}$ with 4 records: $\{\langle a_1, a_2, a_3\rangle, \langle a_1, a_2, a_5\rangle, \langle a_4, a_5, a_6\rangle, \langle a_4, a_6, a_7\rangle\}$. By inspection, an intuitive partition would reveal two clusters: $\mathcal{C}_1 = \{\langle a_1, a_2, a_3\rangle, \langle a_1, a_2, a_5\rangle\}$ and $\mathcal{C}_2 = \{\langle a_4, a_5, a_6\rangle, \langle a_4, a_6, a_7\rangle\}$, with their corresponding histograms: $\overline{H_{\mathcal{C}_1}} = \{\langle a_1, 2\rangle, \langle a_2, 2\rangle, \langle a_3, 1\rangle, \langle a_5, 1\rangle\}$ and $\overline{H_{\mathcal{C}_2}} = \{\langle a_4, 2\rangle, \langle a_5, 1\rangle, \langle a_6, 2\rangle, \langle a_7, 1\rangle\}$. Suppose now we have a different clustering, $\mathcal{C}'_1 = \{\langle a_1, a_2, a_3\rangle, \langle a_4, a_5, a_6\rangle\}$ and $\mathcal{C}'_2 = \{\langle a_1, a_2, a_5\rangle, \langle a_4, a_6, a_7\rangle\}$. We then observe the following, which explains the intuition behind CLOPE's algorithm:

- clusters $\mathcal{C}_1$ and $\mathcal{C}_2$ have better intra-cluster similarity then $\mathcal{C}'_1$ and $\mathcal{C}'_2$; in fact, records in $\mathcal{C}'_1$ and $\mathcal{C}'_2$ are totally different!
- the cluster histograms of $\mathcal{C}'_1$ and $\mathcal{C}'_2$ have a lower size-to-width ratio than $\overline{H_{\mathcal{C}_1}}$ and $\overline{H_{\mathcal{C}_2}}$, which suggests clusters with higher intra-cluster similarity have higher size-to-width ratio in their cluster histograms.

Ideally, a straightforward application of CLOPE should provide us with the summary statistics we need. Unfortunately, CLOPE requires multiple scans of the data, where the number of iteration depends on the desired level of intra-cluster similarity. This violates the one-pass requirement. Furthermore, CLOPE requires multiple evaluation of the clustering criterion for each record, an expensive operation when the size of the stream is massive.

Our solution in SCLOPE is based on the following observation: the optimal height of individual cluster histograms (for each micro-cluster) can be obtained

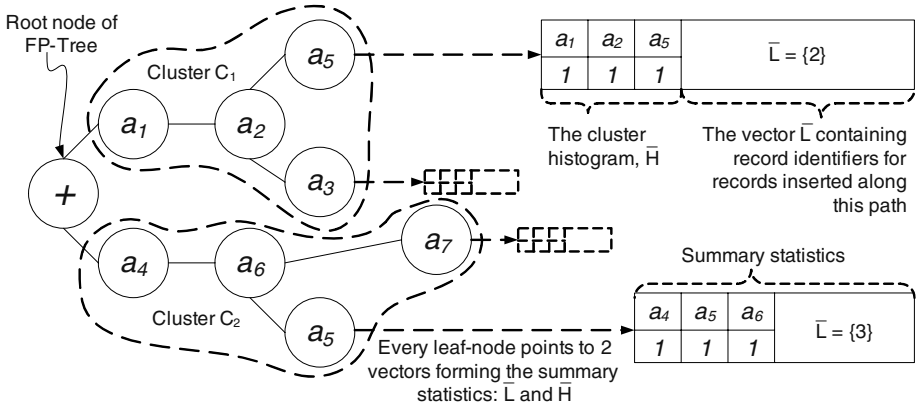

Fig. 1. Each path in the `FP-Tree` leads to a cluster histogram $\overline{H}$ and a vector $\overline{L}$ containing the record identifiers. Notice that records with common attributes share common prefixes, leading to a natural way of identifying clusters.

from a `FP-Tree`–like [5] structure. And this can be done in one sequential scan without the need to compute the clustering criterion. To understand this observation, we first revisit our example but this time with the `FP-Tree`. The formalism and algorithm follows after that.

Figure 1 shows the `FP-Tree` constructed for our example. We have omitted the *count*, *node links* and *header table* in the original `FP-Tree` definition as they are not needed in `SCLOPE`. We also ignore the one-pass constraint for the time-being, and note that the `FP-Tree` in the figure requires two scans – the first to determine the singleton frequency, i.e., $\texttt{freq}(a, \mathcal{D}_{(t_p,t_q)})$, and the second to insert each $\mathcal{R} \in \mathcal{D}_{(t_p,t_q)}$ into the `FP-Tree` after arranging all attributes $a \in \mathcal{R}$ according to their descending singleton frequency.

The implication above is that similar records are inherently "clustered" together through the sharing of a common prefix in the `FP-Tree`. In our example, we can visually confirm two natural clusters from the common prefixes a_1, a_2 and a_4, a_5, which suggests that $\mathcal{C}_1$ and $\mathcal{C}_2$ would be a better clustering than $\mathcal{C}'_1$ and $\mathcal{C}'_2$. In other words, we can actually consider each path (from the root to a leaf node) to be a micro-cluster, where the common prefixes *suggest* the micro-clusters to merge. This leads us to the following.

Observation 1. *An* `FP-Tree` *construction on* $\mathcal{D}_{(t_p,t_q)}$ *produces a set of micro-clusters (not necessary the optimal)* $\mu^{\mathcal{C}}_1, \ldots, \mu^{\mathcal{C}}_k$*, where* k *is determined by the number of unique paths* $\mathcal{P}_1, \ldots, \mathcal{P}_k$ *in the* `FP-Tree`*.*

Due to space, we shall skip the rationale of *all* our observations, and refer the reader to our technical report [9]. Nevertheless, we want to point out the fact that Observation 1 does not guarantee an optimal result. While this may not sound ideal, it is often sufficient for most stream applications. In fact, a near optimal solution that can be quickly obtained (without the need to evaluate the clustering criterion) is preferred in the design of the online component.

Not obvious is that `CLOPE`'s clustering technique, which is to maximize the height of its cluster histograms, is closely related to the properties of `FP-Tree`'s

Algorithm 1 Online Micro-clustering Component of `SCLOPE`

on begining of (window w_i) **do**
1: **if** $(i = 0)$ **then** $\mathcal{Q}' \leftarrow \{$a random order of $v_1, \ldots, v_{|\mathcal{A}|}\}$
2: $\mathcal{T} \leftarrow$ new `FP-Tree` and $\mathcal{Q} \leftarrow \mathcal{Q}'$
3: **for all** (incoming record $\mathcal{R} \in \mathcal{D}_{(t_p,t_q)}$) **do**
4: **order** $\mathcal{R}$ according to $\mathcal{Q}$ and $\forall a \in \mathcal{R}$, $\texttt{freq}(a, \mathcal{Q}')$++
5: **if** ($\mathcal{R}$ can be inserted completely along an existing path $\mathcal{P}_i$ in $\mathcal{T}$) **then**
6: $\forall a \in \mathcal{R}$, $\overline{L_i} \leftarrow \overline{L_i} \cup \texttt{rid}(\mathcal{R}_i) \wedge \texttt{freq}(a, \overline{H_i})$++
7: **else**
8: $\mathcal{P}_j \leftarrow$ new path in $\mathcal{T}$ and $\overline{H_j} \leftarrow$ new cluster histogram for $\mathcal{P}_j$
9: $\forall a \in \mathcal{R}$, $\texttt{freq}(a, \overline{H_j}) \leftarrow 1$ and $\forall a \notin \mathcal{R}$, $\texttt{freq}(a, \overline{H_j}) \leftarrow 0$
10: **end if**
11: **end for**

on end of (window w_i) **do**
12: $\mathcal{L} \leftarrow \{\langle n, \texttt{height}(n)\rangle : n$ is node in $\mathcal{T}$ with > 2 children$\}$
13: **order** $\mathcal{L}$ according to $\texttt{height}(n)$
14: **while** $(|\overline{H}_1, \ldots| > \varphi)$ **do**
15: **select** $\langle n, \texttt{height}(n)\rangle \in \mathcal{L}$ **where** $\forall n \neq m$, $\texttt{height}(n) \geqslant \texttt{height}(m)$
16: **select** paths $\mathcal{P}_i, \mathcal{P}_j$ **where** $n \in \mathcal{P}_i, \mathcal{P}_j$
17: $\overline{H_{new}} \leftarrow \overline{H_i} \cup \overline{H_j}$
18: **delete** $\overline{H_i}, \overline{H_j}$
19: **end while**
20: **output** micro-clusters $\mu_1^{\mathcal{C}}, \ldots, \mu_\varphi^{\mathcal{C}}$ and cluster histograms $\overline{H_1}, \ldots, \overline{H_\varphi}$ for w_i

construction. Recall that each path in the `FP-Tree` can contain multiple records, and that the construction is to maximize the overlapping (or sharing of common prefixes) of nodes, we actually have a natural process of obtaining a good cluster histogram for each micro-cluster. Observation 2 states this property.

Observation 2. *Given a micro-cluster $\mu_i^{\mathcal{C}}$ from a path $\mathcal{P}_i$, its cluster histogram $\overline{H_i}$ has a height that is naturally optimized (again, not necessary optimal) by the* `FP-Tree` *construction process.*

In the simplest case, once the `FP-Tree` is obtained, we can output the micro-clusters as the summary statistics for offline analysis. Unfortunately, these micro-clusters are often too fine in granularity and thus, continue to consume a lot of disk space. One solution is to agglomeratively merge the micro-clusters until they are sufficiently lightweight. However, doing so by evaluating the clustering criterion will prevent the algorithm from keeping up with the data rate of the stream. A strategy to do this efficiently is required.

Observation 3. *Given any micro-cluster $\mathcal{C}_i$ in the* `FP-Tree` *and its corresponding unique path $\mathcal{P}_i$, the micro-cluster(s) that give a good intra-cluster similarity (when merged with $\mathcal{C}_i$) are those whose paths overlap most with $\mathcal{P}_i$.*

Essentially, the above observation answers the question: How can we quickly determine, without the need to evaluate the clustering criterion, the micro-cluster to merge with t+he one under consideration? Since stream applications

require only approximate results, we can conveniently exploit the property of common prefixes (i.e., Observation 3) to select the pair of micro-clusters to be merged. This is realized in Algorithm 1, lines $12-19$, where the key operation is to merge the cluster histograms and the record identifiers.

The idea is to start at the node having more than one child (i.e., more than one path/micro-cluster). This node would be the furthest from the root (therefore, lines $12-13$) and thus, contains the set of paths with the longest common prefix. By Observation 3, any two paths passing through this node would have a good intra-cluster similarity. Thus, we select any two paths passing through the node, and merge its corresponding cluster histograms and record identifiers (i.e., $\overline{H_i}$ and $\overline{H_j}$, lines $17-18$). This process repeats until the set of micro-clusters are sufficiently reduced to fit in the given space.

2.2 Working in Bounded Memory

Up to this point, we have shown how the `FP-Tree` is used to produce the summary statistics we need. In this sub-section and the next, we discuss how we made adjustments to satisfy the data stream constraints.

We begin with the issue of bounded memory. Without doubt, any attempt to process an unbounded data stream is likely to exhaust the limited computing resources before producing any results. To overcome this, we process the stream in a sliding window fashion. We assume δ to be the space allocated for storing summary statistics in the pyramidal timeframe. In the beginning, δ will be uniformly shared with each window having w_s space. For easy discussion, this space can be expressed in terms of the maximum number of micro-clusters (and histograms) allowed in a given window.

At the start of each window, we begin with an empty `FP-Tree` and insert each record into the data structure according to the rules given in Algorithm 1, lines $4-9$. This continues until we reach the end of the window, where we begin `FP-Tree` minimization to produce the summary statistics of size w_s. Clearly, by this process, there will be a time when the δ space is filled by the first δ/w_s windows. Therefore, space must be created for the subsequent windows, i.e., $(\delta/w_s)+1, \ldots, (\delta/w_s)+j, \ldots$, and so on. In the pyramidal timeframe, there are two strategies to do so: *compress* and *delete*.

Intuitively, we first make room by compressing the statistics that became old, and then deleting them as they become outdated. Our strategy to create space for the subsequent $(\delta/w_s)+1$ windows is to redistribute the δ spaces among all the $(\delta/w_s)+p$ windows created so far. In other words, rather then to have w_s amount of space for each window, we reduce w_s by a fraction using a "decay" function: $(1-e^{-\mu})\times w_s$ that is dependent on the window's age. Thus, if we have seen $(\delta/w_s)+p$ windows, then the size of the $(\delta/w_s)+j^{\,th}$ window, would be $(1-e^{-j/p})\times w_s$ where $1 \leqslant j \leqslant p$ (see [9]).

The final step is to resize the summary statistics in each window. We first reconstruct the `FP-Tree` from the summary statistics. This procedure is similar to the `FP-Tree` construction, where we simply insert a path (i.e., a group of records) instead of a record at a time. We then perform minimization until the

Algorithm 2 Offline Macro-clustering Component of `SCLOPE`

1: **let** $\mathbb{C} = \{\langle \overline{H}_{i+1}, \mu^{\mathcal{C}}_{i+1}\rangle, \ldots, \langle \overline{H}_{i+\varphi}, \mu^{\mathcal{C}}_{i+\varphi}\rangle, \ldots, \langle \overline{H}_{j+1}, \mu^{\mathcal{C}}_{j+1}\rangle, \ldots, \langle \overline{H}_{j+\varphi}, \mu^{\mathcal{C}}_{j+\varphi}\rangle\}$
2: **repeat**
3: **for all** ($\mathcal{C}_{\mathcal{F}} \in \mathbb{C}$) **do**
4: **move** $\mathcal{C}_{\mathcal{F}}$ to an existing cluster **or** new cluster $\mathcal{C}_j$ that maximizes profit
5: **if** ($\mathcal{C}_{\mathcal{F}}$ has been moved to some cluster $\mathcal{C}_k$) **then**
6: **update** cluster label of $\mathcal{C}_{\mathcal{F}}$ to k
7: **end if**
8: **end for**
9: **until** no further cluster is moved **or** processing time is exceeded

set of micro-clusters fit in the smaller space. Thus, older windows will have less space allocated and depending on the domain requirements, they may be deleted if they become obsolete.

2.3 Accessing Data in One-Sequential Pass

Our solution to the sequential one-pass access of data streams is to use an incremental update strategy to compute the ordering of attributes based on their descending singleton frequencies. The idea is simple: we begin by assuming a default order (Algorithm 1, line 1), e.g., attributes are seen in each incoming record. As we process each of them, we update the singleton frequency (line 4) of each attribute before inserting the record into the `FP-Tree`.

Upon reaching the end of window, we update the ordering of attributes (i.e., line 1), and use this new ordering in the next window. As a result, a record can have its attributes ordered differently in each window. Thus, it is possible to obtain a sub-optimal `FP-Tree` (initially) depending on the initial assumed order. Fortunately, this isn't an issue as the `FP-Tree` improves on optimality as the stream progresses. In our empirical results, this proved to be effective and reduces the construction to a single pass.

More importantly, this strategy is crucial to the success of exploiting Observation 3 for accurate clustering. Recall that a stream's characteristics actually changes over time, it will not be appropriate to use an assumed or pre-computed ordering. If it's used, a change in the stream's characteristics will caused all subsequent clustering to be sub-optimal, and there will not be any mechanism to recover from that. In that sense, our proposal is more robust because any sub-optimality in the `FP-Tree` (due to changing data characteristics) will be corrected on the next window cycle.

3 Cluster Discovery

Once summary statistics are generated, the analyst performs clustering over different time-horizons using the offline macro-clustering component. Since the offline component does not require access to the data, its design is not constrained by the one-pass requirement. Hence, we have Algorithm 2.

A typical time-sensitive cluster discovery begins with the analyst entering the time-horizon h, and the repulsion r. The time-horizon of interest usually spans one or more windows, and determines the micro-clusters involved in the analysis. On the other hand, the repulsion controls the intra-cluster similarity required, and is part of the clustering criterion defined as [3]:

$$\texttt{profit}(\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}) = \left[\sum_{i=1}^{k} \left(\frac{\texttt{size}(\mathcal{C}_i)}{\texttt{width}(\mathcal{C}_i)^r} \times |\mathcal{C}_i|\right)\right] \times \left(\sum_{i=1}^{k} |\mathcal{C}_i|\right)^{-1}$$

The most interesting aspect of Algorithm 2 is its design for time-sensitive data mining queries. When used together with the pyramidal timeframe, we can analyze different parts of the data stream, by retrieving statistics of different granularity to produce the clustering we need. And since this is the offline component of SCLOPE (which can run independent of the data stream), our design favors accuracy over efficiency, i.e., it makes multiple iterations through the statistics, and cluster using the profit criterion.

Nevertheless, our offline component is still fast despite the fact that it is based on the design of CLOPE. The rationale behind this speed is that CLOPE works with one record at a time while SCLOPE works with a group of records at a time. In our algorithm, each micro-cluster is treated as a pseudo-record, and are clustered accordingly to the given r value that in turn, determines the number of clusters k. Since the number of pseudo-records are very much lower than the physical records, it takes less time to converge on the clustering criterion.

4 Empirical Results

The objective of our empirical tests is to evaluate SCLOPE in 3 aspects: *performance*, *scalability*, and *cluster accuracy*. Due to space constraints, we only report the overall results of our experiments. The interested can refer to our technical report [9] for all the test details.

Performance For an accurate comparison, we tested the performance of SCLOPE using only real-life data sets from the FIMI repository (*fimi.cs.helsinki.fi*). When we compared our results against CLOPE, the best algorithm for clustering categorical data sets, our proposal outperforms CLOPE by a large margin. On cases where the required number of micro-clusters is large, e.g., 200, CLOPE takes more than 10 hours to complete while SCLOPE comes in under 900 seconds. In all 4 real-life data sets tested, our results revealed that SCLOPE is very suitable for processing data streams given its low runtime and insensitivity to the number of clusters. This is crucial since the number of micro-clusters need to be inherently large to facilitate different analysis tasks.

Scalability To test scalability, we used the IBM synthetic data generator. We tested two aspects of scalability: the number of attributes, and the number of records. In the first test, we injected a data set of 50K records with different number of attributes from 467 to 4194. We then recorded the runtime of SCLOPE

and CLOPE in creating 50, 100 and 500 clusters. In all cases, SCLOPE's runtime remains stable while CLOPE's runtime rises sharply as the attributes goes beyond 800. On the second part of the test, we vary the number of records from 10K to 500K. Again, SCLOPE is faster (or on par) with CLOPE in terms of their runtime for different number of clusters.

Accuracy In our final test, we used the mushroom data set (also from the FIMI repository) which contains two predefined classes: 4208 *edible* mushrooms and 3916 *poisonous* mushroom. To measure the accuracy, we used the purity metric (see [3]). In our experiment, we tried different combinations of w_s and φ which are the two parameters affecting the cluster quality in SCLOPE. From the results, SCLOPE consistently attains a higher purity than CLOPE in all situations. Together with SCLOPE's performance, we are convinced of its potential.

5 Related Work

Much of the early works in clustering were focused on numerical data, where most are efficient in situations where the data is of low-dimensionality. Representative of these include k-means, BIRCH [10], CLARANS [6], and CLIQUE [11].

In recent years, there has been a large amount of categorical data accumulated. Their dimensionality and size are often very much larger than numerical data, and exhibit unique characteristics that make numerical-based techniques awkward. This motivated the design of new algorithms leading to works such as CACTUS [12], ROCK [7], STIRR [13], and CLOPE.

While these algorithms are an advancement over numerical solutions, they are not designed with the constraints of data streams in mind. As a result, they are often resource intensive. For example, ROCK has a high computational cost, and require sampling in order to scale to large data sets. Closest to our work are therefore CluStream, STREAM [14], FC [15], and binary k-means.

In comparing the CluStream framework, our work differs by the virtue of the data type we investigate, i.e., we focus on categorical data. Likewise, STREAM and FC are numerical-based techniques, and is thus different from SCLOPE. In the case of binary k-means, a different clustering criterion is used, and its design does not support time-sensitive cluster analysis.

6 Conclusions

In this paper, we propose a fast and effective algorithm, called SCLOPE, that clusters an evolving categorical data stream. We chose to design our algorithm within the framework of CluStream so that it not only outperforms algorithms in its class, but also provide support for time-sensitive cluster analysis not found in most preceding works.

Our empirical tests, using real-world and synthetic data sets, proved that SCLOPE has very good performance and scalability. It also demonstrates good cluster accuracy despite the data stream constraints imposed on the algorithm.

More importantly, the accuracy of clusters generated by SCLOPE can be improved by varying the resource parameters: γ and δ, or allowing an extra scan of the data. This makes SCLOPE an attractive solution for clustering categorical data, either in the context of streams or conventional snapshots.

The drawback with the current design of SCLOPE is the lack of an error quantification on its approximated results. In some data stream applications, it is desirable for the analyst to specify the allowable error in the results, rather then to specify the amount of space. Therefore, an immediate future work would be to extend SCLOPE to handle both situations.

References

[1] Bradley, P.S., Gehrke, J., Ramakrishnan, R., Srikant, R.: Philosophies and Advances in Scaling Mining Algorithms to Large Databases. Communications of the ACM (2002)
[2] Hulten, G., Domingos, P.: Catching Up with the Data: Research Issues in Mining Data Streams. In: Workshop on Research Issues in Data Mining and Knowledge Discovery, Santa Barbara, CA (2001)
[3] Yang, Y., Guan, X., You, J.: CLOPE: A Fast and Effective Clustering Algorithm for Transactional Data. In: Proc. SIGKDD, Edmonton, Canada (2002)
[4] Aggarwal, C., Han, J., Wang, J., Yu, P.S.: A Framework for Clustering Evolving Data Streams. In: Proc. VLDB, Berlin, Germany (2003)
[5] J. Han and J. Pei and Y. Yin: Mining Frequent Patterns without Candidate Generation. In: Proc. SIGMOD, Dallas, Texas, USA (2000)
[6] Ng, R., Han, J.: Efficient and Effective Clustering Methods for Spatial Data Mining. In: Proc. VLDB, Santiago de Chile, Chile (1994)
[7] Guha, S., Rastogi, R., Shim, K.: ROCK: A Robust Clustering Algorithm for Categorical Attributes. In: Proc. ICDE, Sydney, Austrialia (1999)
[8] Wang, K., Xu, C., Liu, B.: Clustering Transactions Using Large Items. In: Proc. CIKM, Kansas City, Missouri, USA (1999)
[9] Ong, K.L., Li, W., Ng, W.K., Lim, E.P.: SCLOPE: An Algorithm for Clustering Data Streams of Categorical Attributes. Technical Report (C04/05), Deakin University (2004)
[10] Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: AN Efficient Data Clustering Method for Very Large Databases. In: Proc. SIGMOD, Canada (1996)
[11] Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. In: Proc. SIGMOD, Seattle, Washington, USA (1998)
[12] Ganti, V., Gehrke, J., Ramakrishnan, R.: CACTUS: Clustering Categorical Data Using Summaries. In: Proc. SIGKDD, San Diego, California, USA (1999)
[13] Gibson, D., Kleinberg, J.M., Raghavan, P.: Clustering Categorical Data: An Approach Based on Dynamical Systems. In: Proc. VLDB, New York, USA (1998)
[14] O'Callaghan, L., Meyerson, A., Motwani, R., Mishra, N., Guha, S.: Streaming Data Algorithms for High Quality Clustering. In: Proc. ICDE, USA (2002)
[15] Barbara, D.: Requirements for Clustering Data Streams. ACM SIGKDD Explorations **2** (2002)