

SCLOPE: An Algorithm for Clustering Data Streams of Categorical Attributes

Kok-Leong Ong¹, Wenyuan Li², Wee-Keong Ng², and Ee-Peng Lim²

¹ School of Information Technology, Deakin University
Waurin 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, aseplim}@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 poses a challenge for data mining, and motivates a new class of problem called *data streams* [4, 6, 10]. 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 motivate 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 [16]. 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

stream-related constraints. Our contribution towards this problem is the **SCLOPE** algorithm inspired by two recent works: the **CluStream** [1] framework, and the **CLOPE** [16] 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 loses 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 facilitates 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* [1, 17]), 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 in Section 2, where we propose an algorithm to gather the required statistics in one sequential scan of the data. Using an observation in the **FP-Tree** [11], 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 in Section 3, 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* [3], **CLARANS** [12], **ROCK** [9], and **LargeItem** [15]. 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.

Following that, we discuss our empirical results in Section 4, where we evaluate our design along 3 dimensions: performance, scalability, and cluster accuracy in a stream-based context. Finally, we conclude our paper with related works in Section 5, and future works in Section 6.

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** [11] structure. Also, without loss of generality, we define our clustering problem as follows. A data stream \mathcal{D} is a set of records $\mathcal{R}_1, \dots, \mathcal{R}_i, \dots$ arriving at time periods t_1, \dots, t_i, \dots , such that each record $\mathcal{R} \in \mathcal{D}$ is a vector containing attributes drawn from $\mathcal{A} = \{a_1, \dots, a_j\}$. A clustering $\mathcal{C}_1, \dots, \mathcal{C}_k$ on $\mathcal{D}_{(t_p, t_q)}$ is therefore a partition of records $\mathcal{R}_x, \mathcal{R}_y, \dots$

seen between t_p and t_q (inclusive), such that $\mathcal{C}_1 \cup \dots \cup \mathcal{C}_k = \mathcal{D}_{(t_p, t_q)}$ and $\mathcal{C}_\alpha \neq \emptyset$ and $\forall \alpha, \beta \in [1; k]$, and $\mathcal{C}_\alpha \cap \mathcal{C}_\beta = \emptyset$.

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 μ^c for a set of records $\mathcal{R}_x, \mathcal{R}_y, \dots$ with time stamps t_x, t_y, \dots is a tuple $\langle \bar{L}, \bar{H} \rangle$, where \bar{L} is a vector of record identifiers, and \bar{H} is its cluster histogram.

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

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

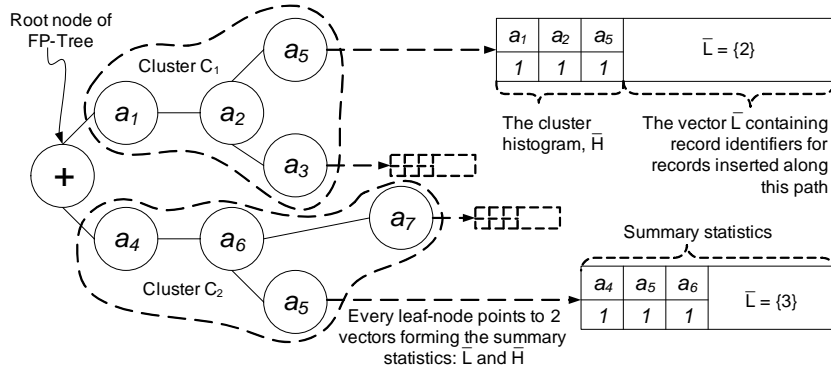


Fig. 1. Each path in the FP-Tree leads to a cluster histogram \bar{H} and a vector \bar{L} containing the record identifiers. Notice that records with common attributes share common prefixes, leading to a natural way of identifying clusters.

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 from a FP-Tree-like [11] 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., $\text{freq}(a, \mathcal{D}_{(t_p, t_q)})$, 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 confirmed 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) $\mu_1^{\mathcal{C}}, \dots, \mu_k^{\mathcal{C}}$, where k is determined by the number of unique paths $\mathcal{P}_1, \dots, \mathcal{P}_k$ in the FP-Tree.

We will skip the rationale of Observation 1 since it's a straightforward extension of our example. Rather, we want to point out the fact that Observation 1 indicates that it does not guarantee an optimal clustering 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 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 μ_i^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.*

Rationale. Let path $\mathcal{P}_i = \{a_j, \dots, a_k\}$, such that $\forall r \in \mathcal{P}_i, r \subseteq \{a_j, \dots, a_k\}$. Let r_s be a record to be inserted into the FP-Tree, such that $\exists a_s \in r_s, a \notin \mathcal{P}_i$. By FP-Tree construction, r_s will not be laid on \mathcal{P}_i due to a_s , i.e., a new path will be created for r_s , and that the *width* of \overline{H}_i remains the same. However, if $\forall a \in r_s, a \in \mathcal{P}_i$ holds, then r_s lies along \mathcal{P}_i leading to \overline{H}_i 's *width* being fixed, but its *height* increases. \square

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 .*

Rationale. Let $\mathcal{P}_i = \{a_l, a_m, a_n\}$, $\mathcal{P}_j = \{a_l, a_p, a_n\}$ and $\mathcal{P}_k = \{a_p, a_q, a_r, j\}$. If two paths overlaps (e.g., $\mathcal{P}_i \cap \mathcal{P}_j = \{a_l, a_n\}$), then by FP-Tree's construction property, a common prefix will be created on their common attributes. In other words, two paths (e.g., $\mathcal{P}_i \cap \mathcal{P}_k = \emptyset$) without common attributes will not have a common prefix. Since each path corresponds to their micro-cluster, a quick way to determine the similarity degree of two clusters is to test for the length of their common prefix, i.e., the number of common attributes. \square

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 the one under consideration? Since stream applications require only approximate results, we can conveniently exploit the property of

Algorithm 1 Online Micro-clustering Component of SCLoPE

```
on beginning of (window  $w_i$ ) do
1: if ( $i = 0$ ) then  $\mathcal{Q}' \leftarrow \{\text{a random order of } v_1, \dots, v_{|\mathcal{A}|}\}$ 
2:  $\mathcal{T} \leftarrow \text{new FP-Tree}$  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  $\forall a \in \mathcal{R}, \text{freq}(a, \mathcal{Q}')++$ 
5:   if ( $\mathcal{R}$  can be inserted completely along an existing path  $\mathcal{P}_i$  in  $\mathcal{T}$ ) then
6:      $\forall a \in \mathcal{R}, \overline{L}_i \leftarrow \overline{L}_i \cup \text{rid}(\mathcal{R}_i) \wedge \text{freq}(a, \overline{H}_i)++$ 
7:   else
8:      $\mathcal{P}_j \leftarrow \text{new path in } \mathcal{T}$  and  $\overline{H}_j \leftarrow \text{new cluster histogram for } \mathcal{P}_j$ 
9:      $\forall a \in \mathcal{R}, \text{freq}(a, \overline{H}_j) \leftarrow 1$  and  $\forall a \notin \mathcal{R}, \text{freq}(a, \overline{H}_j) \leftarrow 0$ 
10:  end if
11: end for

on end of (window  $w_i$ ) do
12:  $\mathcal{L} \leftarrow \{(n, \text{height}(n)) : n \text{ is node in } \mathcal{T} \text{ with } > 2 \text{ children}\}$ 
13: order  $\mathcal{L}$  according to  $\text{height}(n)$ 
14: while ( $|\overline{H}_1, \dots| > \varphi$ ) do
15:   select  $\langle n, \text{height}(n) \rangle \in \mathcal{L}$  where  $\forall n \neq m, \text{height}(n) \geq \text{height}(m)$ 
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  $\mu_1^c, \dots, \mu_\varphi^c$  and cluster histograms  $\overline{H}_1, \dots, \overline{H}_\varphi$  for  $w_i$ 
```

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 attempts 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., $(\delta/w_s) + 1, \dots, (\delta/w_s) + j, \dots$, 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 than 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 $1 \leq j \leq p$ (see [14]).

The final step is 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 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

Algorithm 2 Offline Macro-clustering Component of SCLOPE

```
1: let  $\mathbb{C} = \{\langle \bar{H}_{i+1}, \mu_{i+1}^{\mathbb{C}} \rangle, \dots, \langle \bar{H}_{i+\varphi}, \mu_{i+\varphi}^{\mathbb{C}} \rangle, \dots, \langle \bar{H}_{j+1}, \mu_{j+1}^{\mathbb{C}} \rangle, \dots, \langle \bar{H}_{j+\varphi}, \mu_{j+\varphi}^{\mathbb{C}} \rangle\}$ 
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
```

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 called *profit*. Its definition, given below, is the same as **CLOPE**.

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

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

Table 1. Details of real-life data sets.

Data set	# Records	# Distinct items	Avg. Record Length	File size (KB)
Accidents	340,183	468	33.808	46,255
POS	515,597	1,657	6.530	15,167
Kosarak	990,002	41,270	8.100	35,192
Retail	88,162	16,470	10.306	3,894

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*. Our test environment is set up with a single-CPU Pentium-4 workstation running at 2GHz, with 1GB of RAM having Windows 2000 as the operating system. Our database, which contain real-world and synthetic data sets, is a text file of integers, where each number represents a categorical value, or an attribute-value pair.

Due to space constraints, we only compare our results with **CLOPE**, one of the best algorithm so far for clustering large categorical data sets. Our discussion should therefore provide the reader an idea of **SCLOPE**'s performance against other well-known algorithms, such as **ROCK** and **LargeItem**. For the interested, our technical report presents additional test results [14].

4.1 Performance and Scalability

For convenience, we will not include the performance of our offline macro-clustering results when comparing **SCLOPE** against **CLOPE**. In fact, such comparison is irrelevant because of two reasons. First, the offline component do not operate on limited resources, and takes micro-clusters as the input – not the actual records from the stream. Hence, its runtime is insignificant and is negligible when compared to its online counterpart. Second, the analysis is interactive and is dependent on the scope of analysis. Thus, including a runtime that is based on a specific context does not make semantic sense.

Experiments on Real-life Data For an accurate comparison, we tested performance using only real-life data sets from the FIMI repository (<http://fimi.cs.helsinki.fi/testdata.html>). The data characteristics are listed in Table 1 and their test results in Figure 2. We tested **SCLOPE** (online component only) and **CLOPE** on different cluster settings ranging from 50 to 500 clusters. Given the unbounded size of a data stream, having 500 clusters is not unusual, especially when they are to be used later for time-sensitive macro-cluster analysis.

For our tests, we implemented our algorithm using C and **CLOPE** was obtained from the author's homepage. Interestingly, we observed that the **CLOPE** performs

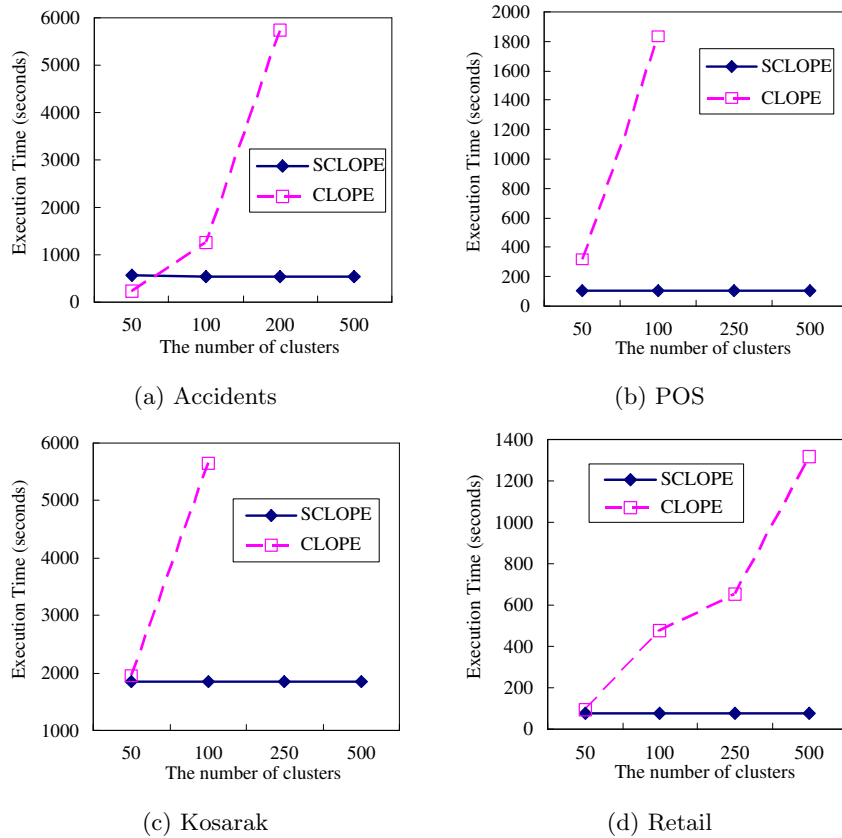


Fig. 2. Scalability test on 4 real-life data sets using different number of clusters.

poorly when the specified number of clusters is large. In fact, **CLOPE** takes more than 10 hours on our tests when dealing with large clusters (we have excluded these results in the plot). In all cases, our tests with real data sets confirmed that **SCLOPE** is very suitable for processing data streams given its low runtime. More importantly, it is not sensitive to the number of clusters. This is crucial because micro-clusters need to be inherently large in number such that they can be stored for different analysis tasks.

Experiments on Synthetic Data We used the IBM synthetic data generator for our scalability tests. We investigated scalability along two dimensions: the number of attributes and the number of records.

For our attribute scalability test, we injected a data set containing 50K records with different number of attributes ranging from 467 to 4194. We then record the runtime performance of **SCLOPE** and **CLOPE** on creating 50, 100 and 500 clusters. From Figure 3, we observed that **CLOPE** is sensitive to both the

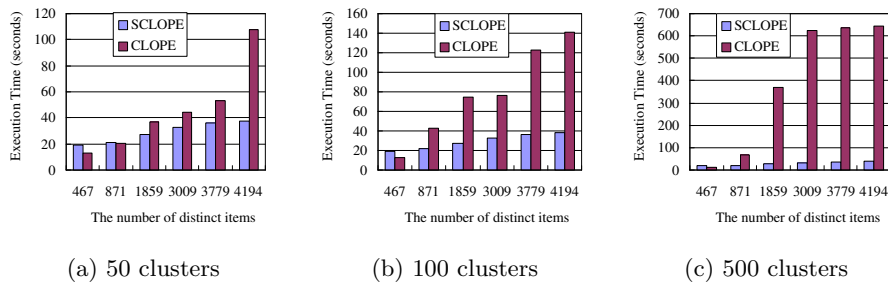


Fig. 3. Scalability test on the number of attributes using synthetic data.

Table 2. Cluster accuracy (*purity*) of the mushroom data set for the online micro-clustering component of SCLOPE using different w_s and φ values.

	$w_s = 10$	$w_s = 20$	$w_s = 30$
$\varphi = 10$	0.956	0.960	0.958
$\varphi = 20$	0.966	0.972	0.971
$\varphi = 30$	0.971	0.981	0.981

number of clusters (thus confirming our earlier observation), and the number of attributes. As either parameter becomes large, CLOPE’s runtime rises sharply while SCLOPE’s runtime remains stable.

The stability of an algorithm in response to an increase in either parameter is important, i.e., an algorithm that is sensitive to these two parameters is likely to fail in keeping up with the data rate. In the case of CLOPE, it has to compute the delta-profit of a record against all clusters [16]. Thus, an increase in the number of clusters cause performance to drop drastically. In the case of SCLOPE, 95% of its time is spent on the construction of the FP-Tree. And in that 95%, there is no need to evaluate the clustering criterion!

For our scalability tests, we fixed the number of attributes to 1000 and vary the size of the database between 10K, 100K, 250K and 500K records. The results are reported in Figure 4. In most cases, SCLOPE performs better than CLOPE although its scalability is sensitive in situations where the number of micro-clusters is small (e.g., 50; see Figure 4(a)). Fortunately, since SCLOPE’s online component is to produce micro-clusters, it is unlikely that the analyst will set the number of clusters to a small value like 50 or 100. Nevertheless, we pointed this out for the interest of our readers.

4.2 Cluster Accuracy

To test the cluster accuracy of SCLOPE, we used the mushroom data set (also available from the FIMI repository), which contains 117 distinct attributes and 8124 records. The records are predefined into two classes of 4208 *edible* mushrooms and 3916 *poisonous* mushroom. Since there are only two class labels, we also employ the *purity* metric (see [16]) to measure the cluster accuracy.

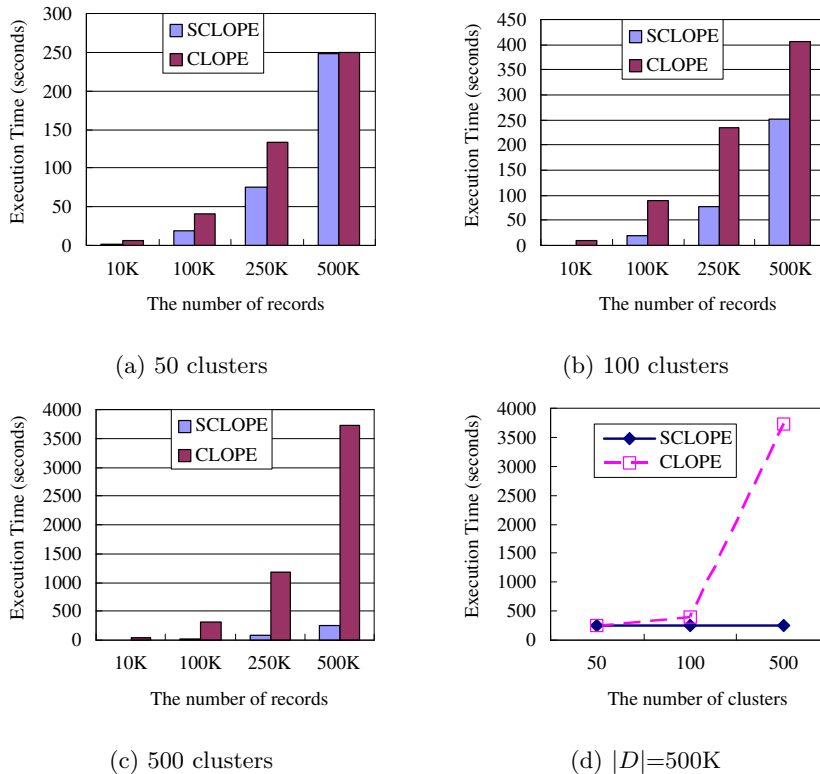


Fig. 4. Scalability test on the size of the data set using synthetic data.

There are two parameters affecting the clustering quality of SCLOPE: the width of each window w_s , and the number of micro-clusters φ in each window. In our tests, we experimented with different combinations of w_s and φ to evaluate the accuracy of SCLOPE. This occurs in two ways: (1) testing the online component to gain insights into the precision of the micro-clusters — important since it will affect the final clustering result; (2) and testing SCLOPE as a whole — to check the accuracy of its final cluster results.

Table 2 and 3 shows the accuracy in terms of purity for different combinations of w_s and φ . In Table 2, the purity for different values of w_s and φ is consistently close to 1. In other words, our online micro-clustering component produces high quality micro-clusters that are suitable for offline analysis. We then use the micro-clusters produced as inputs, and perform clustering over the entire lifetime of the stream using different values of r . The best scores are then tabulated for both SCLOPE and CLOPE in Table 3.

From this table, we see that SCLOPE achieves very good performance and cluster accuracy. Regardless of the combinations for w_s and φ , the final cluster results remain faithfully close to the results of CLOPE. The small difference in

Table 3. Cluster accuracy (*purity*) of the mushroom data set for SCLOPE using different w_s and φ values, and against CLOPE.

	SCLOPE			CLOPE
	$w_s = 10$	$w_s = 20$	$w_s = 30$	
$\varphi = 10$	0.866 ($r = 2.7$)	0.861 ($r = 2.7$)	0.866 ($r = 1.2$)	0.888 ($r = 3.6$)
$\varphi = 20$	0.848 ($r = 2.7$)	0.838 ($r = 1.5$)	0.857 ($r = 0.8$)	
$\varphi = 30$	0.853 ($r = 1.5$)	0.853 ($r = 1.5$)	0.864 ($r = 0.6$)	

the purity of the final result is apparently due to the loss of accuracy during the online micro-clustering phase. Nevertheless, this loss is acceptable when we recall that SCLOPE operates in a stream environment.

To avoid overfitting the algorithm, we also tested the accuracy of SCLOPE on another data set (see details in our technical report: [14]). Briefly, the data is a text collection of science and engineering papers containing 5559 documents with 7811 distinct attributes, and 5 class labels. Instead of the purity, we employed the *F-Measure*, and we compared the accuracy of our algorithm against CLOPE and a few others in the CLUTO toolkit. The results proved that our algorithm is indeed comparable to specialized text clustering algorithms.

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 [3], BIRCH [17], CLARANS [12], and CLIQUE [2].

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 [7], ROCK [9], STIRR [8], 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 [13], FC [5], 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 than to specify the amount of space. Therefore, an immediate future work would be to extend `SCLOPE` to handle both situations.

References

- [1] C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A Framework for Clustering Evolving Data Streams. In *Proc. VLDB*, Berlin, Germany, September 2003.
- [2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. In *Proc. SIGMOD*, Seattle, Washington, USA, June 1998.
- [3] MacQueen J. B. Some Methods for Classification and Analysis of Multivariate Observations. In *Proc. Berkeley Symp. on Math Statistics and Probability*, 1967.
- [4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In *Proc. ACM Symp. PODS*, June 2002.
- [5] D. Barbara. Requirements for Clustering Data Streams. *SIGKDD Explorations*, 2(3), 2002.
- [6] P. S. Bradley, J. Gehrke, R. Ramakrishnan, and R. Srikant. Philosophies and Advances in Scaling Mining Algorithms to Large Databases. *Communications of the ACM*, August 2002.
- [7] V. Ganti, J. Gehrke, and R. Ramakrishnan. CACTUS: Clustering Categorical Data Using Summaries. In *Proc. SIGKDD*, San Diego, California, USA, 1999.
- [8] D. Gibson, J. M. Kleinberg, and P. Raghavan. Clustering Categorical Data: An Approach Based on Dynamical Systems. In *Proc. VLDB*, New York City, New York, USA, August 1998.
- [9] S. Guha, R. Rastogi, and K. Shim. ROCK: A Robust Clustering Algorithm for Categorical Attributes. In *Proc. ICDE*, Sydney, Australia, March 1999.
- [10] G. Hulthen and P. Domingos. 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.
- [11] J. Han and J. Pei and Y. Yin. Mining Frequent Patterns without Candidate Generation. In *Proc. SIGMOD*, Dallas, Texas, USA, May 2000.
- [12] R. Ng and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. In *Proc. VLDB*, Santiago de Chile, Chile, September 1994.
- [13] L. O'Callaghan, A. Meyerson, R. Motwani, N. Mishra, and S. Guha. Streaming Data Algorithms for High Quality Clustering. In *Proc. ICDE*, San Jose, CA, February 2002.

- [14] K.-L. Ong, W. Li, W.-K. Ng, and E.-P. Lim. SCLOPE: An Algorithm for Clustering Data Streams of Categorical Attributes. Technical report, Nanyang Technological University, February 2004.
- [15] K. Wang, C. Xu, and B. Liu. Clustering Transactions Using Large Items. In *Proc. CIKM*, Kansas City, Missouri, USA, November 1999.
- [16] Y. Yang, X. Guan, and J. You. CLOPE: A Fast and Effective Clustering Algorithm for Transactional Data. In *Proc. SIGKDD*, Edmonton, Canada, July 2002.
- [17] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: AN Efficient Data Clustering Method for Very Large Databases. In *Proc. SIGMOD*, Canada, June 1996.