Modeling and Clustering Users with Evolving Profiles in Usage Streams

Chongsheng Zhang[‡], Xiangliang Zhang[*], Florent Masseglia[‡]

[‡]INRIA, AxIS Project-Team, Sophia Antipolis, France
chongsheng.zhang@inria.fr,florent.masseglia@inria.fr
[*]King Abdullah University of Science and Technology, Thuwal, Saudi Arabia
xiangliang.zhang@kaust.edu.sa

Abstract. Existing data stream models commonly assume that users' records or profiles in data streams will not be updated once they arrive. In many applications such as web usage, however, the users' records/profiles may evolve along time. This kind of streaming transactions are referred to as *bi-streaming* data - the data evolves temporally in two dimensions, the flowing of transactions as with the traditional data streams, and the evolving of users' profiles inside the streams, which makes *bi-streaming* data different from traditional data streams. The two-dimensional evolving of *bi-streaming* data brings difficulties on modeling and clustering for exploring the users' behaviours. This paper will propose three models to summarize *bi-streaming* data, which are the batch model, the Evolving Objects (EO) model and the Dynamic Data Stream (DDS) model. Through creating, updating and deleting user profiles, the models summarize the behaviours of each user as an object. Based on these models, clustering algorithms are employed to identify the user groups. The proposed models are tested on a real-world data set showing that the DDS model can summarize the bi-streaming data efficiently and effectively, providing better basis for clustering user profiles than the other two models.

1 Introduction

Data streams are increasingly growing in volume and diversity with the extensive use of web, sensors and mobile devices. These days, many companies need to handle or process large volume of streaming data. Moreover, they expect the streaming data to be timely analyzed such that the real-time results are reported to make prompt responses.

Several data stream models have been proposed to approximately represent the data for accelerating the mining of data streams [1, 8, 5, 4, 15]. The data streams managed by these models have a common characteristic that their transactions are deemed independent. That is to say, each arriving transaction is a complete description of the observed objects. In some real cases, however, the transactions/records may be related to others arriving before or after them. For instance, when users access internet through mobile phones, they are unable

to visit several pages within a very short period due to the operation inconvenience. Thus, their requests are separated into several transactions occurring in non-successive time steps. This kind of data streams is referred to as *Bi-streaming data*, differing from the *traditional streaming data* with independent transactions. We give the definition of *Bi-streaming data* and its summarization model in Definition 1 with the application goal of exploring the user behaviours.

Definition 1. ***Bi-streaming data and summarization model****. Bi-streaming data is a time-ordered sequence that consists of transactions/records, each of which was made by a user with unique identifier. Transactions having the same user can be grouped together and summarized in form of* (***ID, content***)*, where* ***ID*** *is the identifier of the user, and* ***content*** *is an abstract of all the transactions associated with the user.*

In bi-streaming data, every arriving transaction can be an incomplete description of a user. From the user's point of view, his/her behaviour described by the associated transactions is evolving along time. From the streaming's point of view, the usage of system is evolving by adding/updating/deleting users. Summarizing the bi-streaming data into user groups structures a 2-dimensional streaming data, the streaming groups $(ID, content)^t$ along t and the streaming abstraction $content^t$ for each user ID.

To be clear, we give an instance of a bi-streaming data and its summarization in Table 1.

Table 1. An example of bi-streaming data and its summarization. (a) is the streaming of user requested pages and (b)is the summary of navigation of each user

Transaction	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}
User Id	C_1	C_4	C_1	C_2	C_3	C_4	C_1	C_2	C_2	C_4	C_5	C_6
Time	t_1	t_1	t_2	t_3	t_3	t_3	t_4	t_4	t_5	t_6	t_7	t_7
Page	A	A	C	D	B	C	H	E	F	H	B	B

(a)

C_1	C_2	C_3	C_4	C_5	C_6
A	D	B	A	B	B
C	E		C		
H	F		H		

(b)

Example 1. Suppose that we are working with a data streams DS of online web usage. The set of transactions $\{T_1...T_{12}\}$ shown in Table 1 (a) is a proportion of DS. Each transaction T_x contains a time stamp t_i, a unique user identifier (ID) and a requested page. For instance, $T_1 < C_1, t_1, A >$ is a transaction which occurred at time t_1, when user C_1 visited the page A. Checking the transactions derived from user C_1 until $t_i (i = 12)$, we see that this user made a set of navigations $\{A, C, H\}$ by time order. Similarly, user C_2 has a set of navigations $\{D, E, F\}$. The summarization of navigation of each user is shown in Table 1 (b).

When more data are arriving at $t_i (i > 12)$, the set of navigation of users will be updated correspondingly. With the purpose of modeling the user behaviours and clustering the users for mining social groups, the summarization of users in

terms of navigations will be analyzed. However, the summarization of users at time t_i is incomplete which is an important characteristic stated in the definition of bi-streaming data. The incompleteness of the user's summarization makes it unsuitable to use the traditional data streaming models, which either treat the transactions one by one or wait until the descriptions of users' behaviours are complete. The former solution will cause the missing of interesting usage patterns, while the latter one requires the storage of a large amount of data, disabling the expected real-time analysis.

To solve the above problem, we propose 3 different approaches to model and summarize the bi-streaming streams, the Batch model, the Evolving Objects (EO) model and the Dynamic Data Stream (DDS) model. The 3 models provide the summarization of bi-streaming data by using different strategies and having different requirements on the computational resources. Based on the summarization, we cluster the users into social groups by Affinity Propagation that is a new clustering method with advantages in optimizing the clustering distortion [9], and by Streaming Affinity Propagation (StrAP) that is an efficient and effective online clustering algorithm [14]. The proposed models were applied on the real world 21.8 GB data set from *France Telecom*. The experimental results show that the summarization based on the DDS model is better than that of the other two models.

The rest of the paper is organized as follows. We discuss the related work of data stream models and algorithms in section 2. We introduce the Batch model, the EO model and the DDS model in section 3. The analysis of the models is presented in section 4. Section 5 describes the clustering algorithms used on top of the model. Section 6 gives the data sets and parameter setting in the validation. The application validation results are reported in section 7. Finally we conclude in section 8.

2 Related Work

In this section we make a short survey on existing data streams models. Also, we review related work on data stream mining.

Data stream models. There are five commonly adopted data stream models: batch processing model [10], sliding window model [1, 8], tilted-time window model [5], landmark window model [4], and damped stream model [15]. In batch model, the streaming data are divided orderly into several batches, then mining algorithms are run on each batch. The sliding window model only keeps a fixed size of recent data in memory, and replace the old transactions by the new arriving ones. The tilted-time window model provides different levels of resolutions compressing the old data and maintaining the new data in details. Unlike sliding window model which kicks the obsolete data out of the window as new data arrives, the landmark model keeps all the data which arrive after the landmark time. Instead of discarding the old data that live out of a window or arrived before the landmark time, the damped stream model decays the weights of the previous transactions according to their freshness.

Querying bi-streaming data. Regarding the query of bi-streaming data, Haghani et al. [11] investigated the issue of continuously monitoring the top-k individual objects. In this bi-streaming data, the numerical values of the same object are arriving intermittently in different transactions. To rank the objects, all the transactions related to one object within the current sliding window are added. The objects with k largest summation value are reported. In order to efficiently select the top-k objects, Haghani et al. focus on lowering the memory usage by pruning irrelevant objects.

Mining uncertain data streams. Unlike the incomplete but deterministic transactions in bi-streaming, transactions in uncertain data streams can have one of several possible values [7]. Assuming that the standard error of each data point is available, Aggarwal et al. in [2] develop a method for clustering uncertain data streams through the use of an uncertain micro-cluster model, which is comprised of a number of summary structures. When deciding the assignment of data points to clusters, the error level of uncertainty of the data points is included to measure the similarity between the point and the cluster centroid.

Out-of-order data streams. In out-of-order data streams, the arriving time of transaction does not necessarily follow their generation timestamps [6]. For time decaying aggregate computations, thanks to the *q-digest* data structure which summarizes the frequency distribution, the weighted value of the late arrivals are approximately estimated and accounted for in aggregation.

Mining evolving user profiles. Iglesias et al. proposed an online classifier that learns from streaming command-line data (the UNIX csh command), and recognizes the profile class of a computer user, e.g., novice programmers, experienced programmers, computer scientist, or non-programmers [12]. The behaviour of each user is described by a histogram distribution of n relevant subsequence of commands. When a new user brings in the new set of subsequences, n will be increased and all users are described in a higher dimension. This is so-called evolving user profiles. A user is classified into the class to which he/she has the closest similarity measured by cosine distance. The evolving user profiles in this data stream [12] are different from that of bi-streaming data. Each transaction of the former stream is a complete sequence of one user command, while one transaction of the bi-streaming data is only a part of one user's behaviour.

3 Modeling Bi-streaming Data

In this section, we will elaborate on the models for managing and mining the bi-streaming data. As stated before, bi-streaming data is a special data stream where it is common for several transactions to share the same identifiers. How to manage the bi-streaming data is an important issue which will influence the data stream mining results, e.g., clustering, frequent patterns mining. We hence propose 3 different models for handling the bi-streaming data, the Batch model, the Evolving Objects (EO) model and the Dynamic Data Stream (DDS) Model.

3.1 Batch Model

In order to summarize the behaviour of one user described in non-successive transactions, one intuitive way is to

1. collect the arriving transactions in a buffer until the buffer is full
2. for each user in the buffer
   - find all transactions related to this user
   - summarize these transactions
   - create an object for this user with key=UserID and content=summarized transactions
3. release the buffer
4. start to collect the transactions and go to step 1 again.

Figure 1 demonstrates the batch model applied on an example of bi-streaming data shown in Table 1.

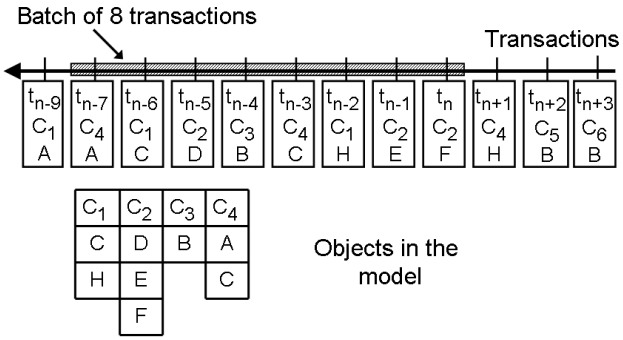


Fig. 1. Building Batch Model from the Bi-streaming Data

Example 2. Let the buffer size to be 8. The collection of transactions $(T_2...T_9)$ is summarized to be a list of objects, $< C_1, \{C, H\} >$, $< C_2, \{D, E, F\} >$, $< C_3, \{B\} >$, $< C_4, \{A, C\} >$. Each object consists of an identifier and the summarized content of his/her transactions.

3.2 Evolving Objects Model

Evolving Objects (EO) model is based on a window sliding along the bi-streaming data. We create and maintain an object for each user by observing the transactions included in the current sliding window. Instead of re-constructing all objects at next buffer in the batch model, we maintain the objects in evolving objects model when reading every new transaction.

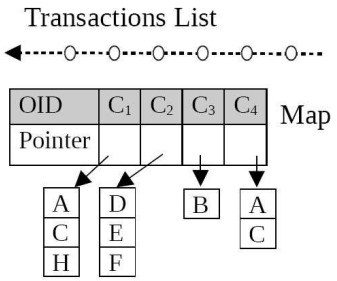


Fig. 2. Data Structure of Evolving Objects (EO) Model

Figure 2 presents the data structure of EO model. Besides the transactions list and the *map* with user ID as the key, extra data structures are used to store the summarized items of each user.

Building the evolving objects model includes the following steps:

1. slide the window forward along the bi-streaming data
2. delete the items/contents in the expired transactions from the corresponding objects in the EO model
3. for each new transaction, check whether the identifier of this transaction exists in the model
   - if yes, add the items/content of the new transaction to the existing object
   - otherwise, create a new object with the items of the new transaction then insert it to the EO model

Figure 3 is an example of applying EO model on bi-streaming data shown in Table 1.

Example 3. In Figure 3, the current sliding window with size 8 includes $(T_2...T_9)$. Before this, we created new objects for the two users when T_1 and T_2 were in the window. As the window was not full, there was no out-of-date transaction to be deleted. Then T_3 came, we checked the model and found that its identifier C_1 was already there. So we updated the content of C_1 from $\{A\}$ to $\{A, C\}$. Likewise, we update the model until we read transaction T_9 in the current sliding window. At this point the window is full and we have to remove the oldest transaction out of the sliding window. Transaction T_1 is the oldest transaction and removed. Consequently, we should update the object of its owner, by updating the content of C_1 from $\{A, C, H\}$ to $\{C, H\}$. Then we add $\{F\}$ to the object of current transaction's owner C_2. The final model obtained by EO model is the same as that built by the Batch model. Actually, EO model is an incremental implementation of the Batch model.

3.3 Dynamic Data Stream Model

Dynamic Data Stream (DDS) model has two main components: an *Objects List* (window) and a *Map* with user ID as the key. Figure 4 shows the data structure

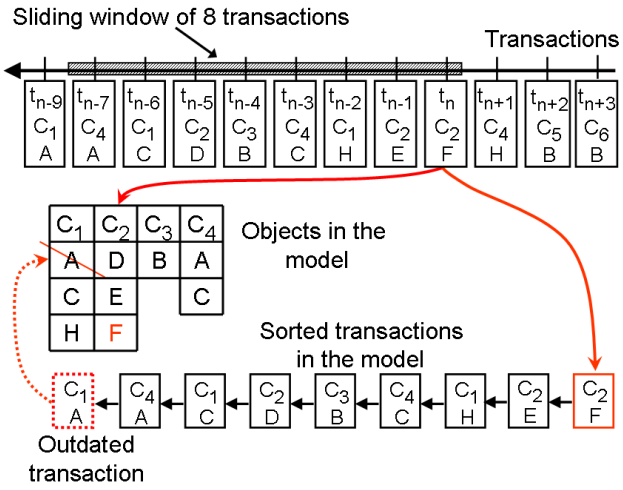


Fig. 3. Evolving Objects (EO) Model

of our proposed DDS model. The *Objects List* maintains the objects in a list, and each object in the list carries the item set for the corresponding user. For simplicity, we refer to the left end of list with old objects as the *head* of the list and the right end with recent objects as the *tail* of the list. The objects that come earlier and have no updates recently would be in the *head* of the list and those recently arrived or updated will be close to the *tail* of the list. When there is not enough space, old objects will be removed from the *head* of the list.

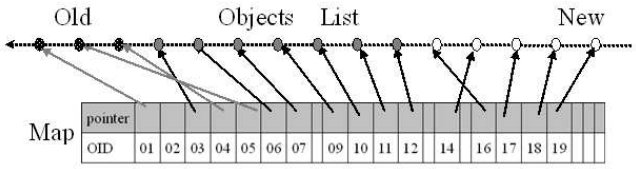


Fig. 4. Data Structure of the Dynamic Data Stream (DDS) Model

In order to quickly check whether the identifier already exists in the current list/window, we keep a map, whose keys are the identifiers of the according objects in the list. Since the identifiers are unique, we can promptly find the according object (with the help of the map) in the list without scanning the data again.

In DDS model, the new/recently updated objects will always be appended or moved to the *tail* of the list and be connected after the object which is currently at the *tail* of the list. Consequently, the *tail* should be the entry for us to add new objects and *head* is the exit for us to remove the old objects out of the list.

Generally, when a new transaction arrives, we update the DDS model by the following process:

1. check whether the identifier of this new transaction exists in the object list
   - if yes, add the items/content of the new transaction to the content of the existing object
   - otherwise, create a new object
2. move the updated (or add the new) object to the *tail* of the list
3. check whether there is no enough space
   - if yes, delete the object in the *head* of list.

Figure 5 illustrates the process of building DDS models from the bi-streaming data shown in Table 1.

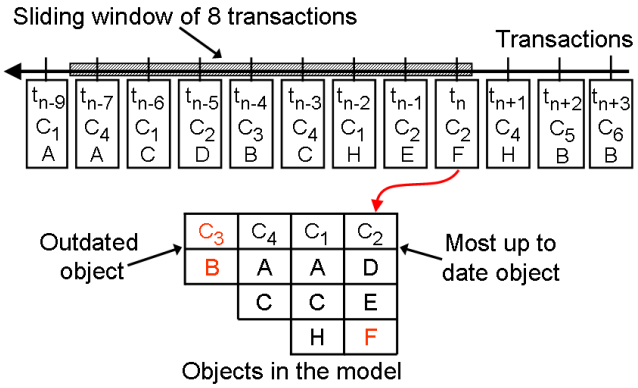


Fig. 5. Building DDS Model from the Bi-streaming Data

Example 4. After reading the first transaction $T_1 < C_1, t_1, A >$ from the bi-streaming data in Table 1, we insert the first object $O_1 < C_1, \{A\} >$ at the *tail* of the list. Similarly, after T_2 arrives, we insert another object $O_2 < C_4, \{A\} >$ after O_1. So far, object O_2 is the latest object in the list and object O_1 is the *head* of the list. When T_3 comes, we will not make a new object, because with the help of the map we find the identifier of T_3 is C_1, the same as the key of object $O_1 < C_1, \{A\} >$. For taking T_3 into the list, we update the content of the O_1 from $O_1 < C_1, \{A\} >$ to $O_1 < C_1, \{A, C\} >$, and move it from its current position (*head* of the list) to the *tail* of the list. After this updating, object O_1 is the latest object in the *tail* of list and object O_2 is oldest and becomes the *head* of the list.

Given limited memory, elder objects in the head of list will be removed. Let the memory limit for the model be 8 transactions. After reading T_9, we have to remove an old object C_3, which is currently the *head* (oldest one) of the list. The final objects in the list will be, in accordance with the order from new objects to old objects, $< C_2, \{D, E, F\} >$, $< C_1, \{A, C, H\} >$, $< C_4, \{A, C\} >$.

4 The Analysis of Bi-streaming Models

In this section we analyze the advantages and disadvantages of the 3 candidates of bi-streaming models, the batch model, EO model and DDS model, in terms of pattern preservation, CPU cost and memory usage.

4.1 Preservation of Usage Patterns

Let us review the data managed in Figures 1, 3 and 5. Given the same transaction data (Table 1) and parameters (buffer size or window size to be 8), we obtain the same object list and content from batch and EO model, which are different from the results in DDS model. Looking into the content of user C_1, we see that his/her behaviours are better preserved in DDS model than the other two models, as DDS model describes the behaviour of C_1 as $\{A, C, H\}$, whereas the batch and EO models describe C_1 as $\{C, H\}$. In addition, user C_3, an out-of-date user, is not kept in DDS model but it is held by batch and EO models.

Comparing to the batch and EO models, DDS model specializes in keeping more complete profiles of frequently updated users. This property is due to the fact that DDS model moves the updated objects to *tail* of the list and pushes the old and seldom updated objects to the *head* of the list for deleting. DDS model can accumulate the users' patterns and keep users with longer patterns (active users with more dynamic behaviours). These patterns are useful in many applications. For instance, they can be used to analyze the web usage data with high bounce rate[1].

The batch model has a very simple data structure, but it may truncate the intrinsic patterns hidden between batches by segmenting of the data in buffer size. The same drawback exists in EO model. However EO model offers dynamically organized objects while the batch model does not neaten the data until the batch is ready.

4.2 CPU Cost

To analyze the efficiency of each candidate model, we consider the price to pay for the operations of *search*, *insertion*, *deletion*, and *update* on the model as computational overhead.

Batch model waits until the buffer is full of transaction. Whenever reading a new transaction, we insert it to the transaction list in buffer. When the buffer

[1] $http://en.wikipedia.org/wiki/Bounce\_rate$

is full, we build a map such that we can merge the transactions with same identifiers. On average for each transaction, the *total cost* is the *search cost* plus the *insertion cost* plus the *updating cost*.

EO model keeps the transactions in a sliding window, meanwhile maintains and updates the summarized objects in extra data structures on the fly. On the one hand, the advantage is that the arranged objects are always available for analysis. On the other hand, every time when a new transaction comes, we should not just list it in the sliding window, but we also search the map and update the corresponding objects. For any operation, we *insert* the transaction in window list. Next we *search* the map to check whether the same identifier already exists in the map. If it is a new object in EO model, we *insert* a new key in the map and create a new associated object for the key. If the identifier exists in EO model, we *update* the corresponding object in the map. EO model requires frequent *deletion* when the sliding window is full and new transaction arrives. Moreover, we have to *search* the according object before deletion and *update* it after deletion.

When reading a new transaction in DDS model, we first look up the map to check whether the according identifier already exists in the map. As each identifier is unique, the cost for the search over the map is O(1). Next, we will have 3 possible operations on the object associated with the identifier: *insertion, update* and *deletion*. If it is a new identifier, we will insert a key in the map. Moreover, we will create a new object and insert it in the *tail* of the list. The *total cost* for insertion is the *search cost* plus the *insertion cost*. If it is an existing identifier, then we will update the according object in the list and move it to the *tail* of the list. If we are going to remove an object from the list, we simply delete it from the list, but we do not need to search the map, because we always remove the oldest one from the *head* of the list. Therefore, comparing to EO model, DDS model requires less *deletion* and the *deletion* is much more economical with no additional actions of *look up* and *update*.

As a conclusion, DDS model has lower CPU cost than the batch model and EO model.

4.3 Memory Usage

The memory cost for each model is listed in Table 2. The memory usage of batch model has two parts. One part is the memory for keeping the batch transactions. The other is the memory required for establishing a mapping table to identify then merge the transactions with the same identifiers. In EO model, we not only have to keep the data in the sliding window, but we also need to maintain a map and extra arranged objects for all the identifiers. As a result, the memory usage is two times of the data size added by the map size. In DDS model, the objects are directly linked in the list. Thanks to the map we can easily reach the objects through the corresponding key. As we do not have to keep other extra data structures, the total memory usage is the size of the data added by that of the map.

The conclusion is that Batch and DDS models have the same memory usage. EO model, however, uses almost two times more memory than them.

Table 2. Memory cost of 3 candidate models

Candidates	memory for modeling	memory for data	Overall memory usage
Batch	map size	data size	map size + data size
EO	map size+data size	data size	map size + 2*data size
DDS	map size	data size	map size + data size

5 Clustering Users based on the Bi-streaming Model

The above introduced 3 models, the batch, EO and DDS models, manage the bi-streaming data through organizing the transactions by their identifiers. Based on the summarization by models, the common data mining tasks can be conducted, e.g., clustering, frequent pattern mining. In this paper, we apply the models for clustering applications. Note that we do not intend to evaluate the pros and cons of the clustering algorithms. Rather, we aim at evaluating different bi-streaming models for providing the best summarization, which is the foundation of mining clusters or frequent patterns.

The clustering methods we used are Affinity Propagation (AP) [9] and Streaming Affinity Propagation (StrAP) [14]. AP is a clustering method based on messages-passing. Comparing with other clustering methods, e.g., k-medoids, it has the advantages in terms of clustering quality, stability and annulling of setting the number of clusters k. StrAP is an online clustering method. It has the merits of (1) seamlessly updating the clustering model; (2) adapting to changes of data distribution; (3) intelligible compressed data model.

StrAP has been successfully used in clustering the traditional data streams [14]. However, it cannot be directly used for clustering the bi-streaming data, because each transaction is not independent and not complete. Our proposed bi-streaming models summarize the flow of transactions to be the flow of objects, which consist of an UserID and the related patterns of this user. In order to cluster the user behaviours, we can apply StrAP on the flow of objects that produced in EO and DDS model. We can also apply AP on the current set of objects at a given time stamp.

Clustering algorithm AP and StrAP require to measure the similarity between pairs of objects. Intuitively, we use the percentage of common elements between two objects as their similarity. AP and StrAP do not need the setting of the number of clusters before using. The only parameter for users to set is the "preference" of each object to be the center of a cluster. In the experiments, we empirically set this parameter to be the median value of all pair-wise similarities. For all different bi-streaming models, we use the same clustering algorithm with the same parameter.

6 Data Sets and Experimental Settings

In this section, we will describe the real-world data set we used, the experimental setting and the evaluation criteria in the application of the proposed 3 bi-streaming models and clustering.

6.1 Data Sets

The real-world data sets we employed are the usage records of mobile users from *France Telecom*. The full data set is of 21.8 GB size and contains nearly 60,000,000 lines of mobile users' browsing records from May to July of 2008. Each record (transaction) has 7 fields, including 3 numerical variables, 3 categorical variables and one date time variable. In our experimental validation, we used three relevant fields: 1) the identifier of the user; 2) the category of page that the user visited; and 3) the corresponding time when the user requested the page.

Besides the full data set, we generated a smaller data set by filtering out the records, whose corresponding users requested less than 2 pages in the real data set. After the preprocessing, the final trimmed data is about 4.7 GB. And the attributes used are the same as the real data set.

As we stated in Definition 1 in the Introduction, this browsing records of mobile users are typically bi-streaming data. Each record has an identifier of the user, and a visited page that is only a part of this user's request. Ordering all the records by their timestamps, we apply the proposed 3 candidate models to build the up-to-date model of this bi-streaming data.

6.2 Experimental Settings

The main parameter in proposed models is the *window size* (*buffer size* in the batch model), which represents how many transactions we will keep in memory. To ensure the fairness of evaluation, we use the same window size for all the 3 models. Thus, the number of transactions kept in each model is the same. In our experiments, the setting of *window size* varies from 10,000 to 120,000. The sensitivities of *window size* is analyzed in section 7.1.

In order to evaluate the performance of the proposed models and the clustering results obtained thereafter, we set up a reference model, which approximates the ideal situation of modeling bi-streaming data. Since we cannot afford to keep and analyze all the data in memory ideally, the next best way is to build a compromised reference model by holding as more data as possible in the Batch model, e.g., setting the buffer size as 3 times the *window size* of EO and DDS model. For instance, suppose that the results are going to be evaluated at transaction T_n and the sliding window capacity is S in EO and DDS model, the number of transactions kept in reference model will be $S \times 3$, counting back from T_n until it has 3 times of the transactions.

6.3 Evaluation Criteria

The bi-streaming models are firstly measured in *efficiency*, i.e., the algorithm running time on each model. The effectiveness of bi-streaming models is assessed by the quality of clustering results obtained on top of the summarization models. The performance of proposed models are compared to that of the reference model, which uses 3 times more transactions in the batch model as we introduced in section 6.2. We adopted the following validation criteria for measuring the clustering quality:

1. **Cluster Stability** [13]. Cluster stability can be considered as a kind of precision. It measures the "goodness" of a clustering with respect to the reference. A larger stability value implies a better clustering.
   Suppose that we have a clustering C_a with K_a clusters and a reference clustering C_r with K_r clusters from the same set of N objects. We construct a $N \times K_a$ probability matrix Ma for clustering C_a. Each element $Ma(i,j) = 0$ if object i is not in cluster $C_a(j)$ and $Ma(i,j) = 1/\sqrt{|C_a(j)|}$ if object i is in cluster $C_a(j)$ which has $|C_a(j)|$ members. Similarly, we construct a $N \times K_r$ matrix Mr for clustering C_r.
   Let M be the multiplication of Ma^T (the transpose of Ma) and Mr, $M = Ma^T \times Mr$. The stability value of clustering C_a w.r.t. reference C_r is defined as:
   $$\textbf{StabilityValue}(C_a, C_r) = trace(M * M')/min(K_a, K_b)$$
   where trace is the sum of diagonal elements of the matrix. The larger the *StabilityValue* is, the closer the clustering C_a and the reference are, therefore the better the clustering C_a is.
2. **Entropy** [3]. This measure is based on information theory. The entropy value addresses the consistency level of clustering with regard to the reference. The larger the entropy, the worse the clustering compared to the reference.
   For example in the above clustering C_a and reference clustering C_r, if the objects which originally belong to the same cluster in $C_a(i)$ are scattered across the clusters in C_r, the entropy of this cluster will be high.
   We define the entropy of $C_a(i)$ (i^{th} cluster of C_a) to be
   $$E(C_a(i)) = -\frac{1}{log(K_r)} \sum_{j=1}^{K_r} \frac{|C_a(i) \cap C_r(j)|}{|C_a(i)|} log\left(\frac{|C_a(i) \cap C_r(j)|}{|C_a(i)|}\right)$$
   The entropy of clustering C_a w.r.t. reference C_r is defined as the sum of the entropy values of all the clusters in C_a:
   $$E(C_a) = \sum_{i=1}^{K_a} \frac{|C_a(i)|}{N} E(C_a(i))$$
3. **Average length of objects**. The average length of objects shows the average number of accumulated patterns over all objects. A longer accumulated patterns means a more intact preservation and summarization of users' behaviours.

7 Experimental Results of Modeling and Clustering the Real-world Data

This section will report the experimental results comparing the performance of the proposed models. We implemented all the models and algorithms in C++ and conducted every experiment on the same computer with one Intel 2.33 GHz processor, 3 GB main memory, running Fedora 7 operating system.

7.1 Clustering user behaviours by AP based on bi-streaming model

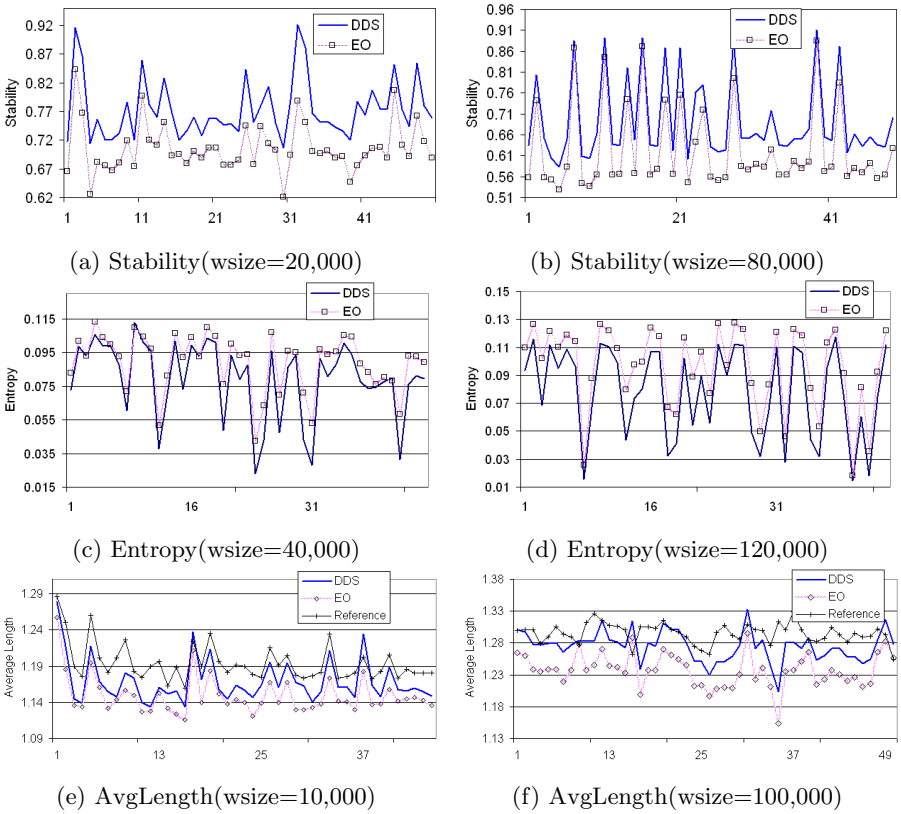
Due to the large volumes of the real-world data, we cannot afford to output all the clusters and validation details at each step. Instead, we randomly select several steps to evaluate the clustering results and details. From each bi-streaming model, we randomly select about 160-200 clusters and evaluate them by comparing with the reference model. As analyzed in section 4, the modeling results of the EO model and Batch models are the same. Hence, we will only compare the clustering results of DDS model and EO model with the reference model in the following experiments.

Figure 6 shows the user behaviours clustering results obtained by using AP on top of the DDS model and EO model, which were used to modeling the real-world bi-streaming data of mobile users. The X-axes of Figure 6 are the randomly selected clustering results. The clustering results are evaluated by the criteria as we introduced in section 6.3, *Stability* (Figure 6 (a) (b)), *Entropy* (Figure 6 (c) (d)) and the *Average length of objects* (Figure 6 (e) (f)). The performance of DDS model and EO model is tested on a wide range setting of *window size* (noted as *wsize*) parameter, from 10,000 to 120,000.

Figure 7 shows the results applied on the trimmed bi-streaming data, which delete the records, whose corresponding users requested less than 2 pages in the real data set. Similar to Figure 6, the clustering results of user behaviours are evaluated by *Stability* (a), *Entropy* (b) and *Average length of objects* (c). The *window size* of DDS model and EO model is set to a medium value 80,000.

In Figure 6 (a)(b) and Figure 7 (a), we see that the clustering stability based on DDS model is always better than that based on EO model. Figure 6 (c)(d) and Figure 7 (b) show that the clustering results based on DDS model has lower entropy (better performance) than that based on EO model. Comparing the average length of objects obtained in DDS model, EO model and reference model shown in Figure 6 (e)(f) and Figure 7 (c), we find that DDS model preserves longer user behaviour patterns than EO model, and as expected has shorter patterns than the reference model that used 3 times more transactions.

As a conclusion from Figure 6 and Figure 7, DDS model preserves longer user behaviour patterns, guarantees better modeling of bi-streaming data and supports more stable and well-partitioned clustering results of user behaviours.



(a) Stability(wsize=20,000) (b) Stability(wsize=80,000)

(c) Entropy(wsize=40,000) (d) Entropy(wsize=120,000)

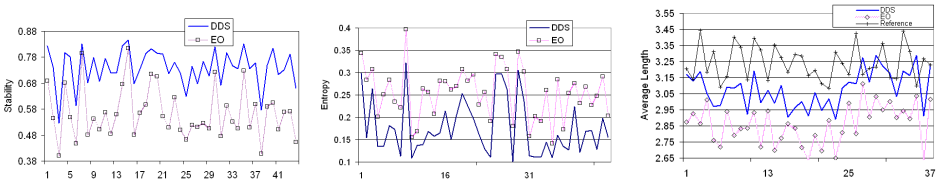
(e) AvgLength(wsize=10,000) (f) AvgLength(wsize=100,000)

Fig. 6. Evaluation of user behaviours clustering results in terms of Stability (a,b), Entropy (c,d) and Average length of objects (e,f), when AP was used on top of the DDS model and EO model applied on the **real-world** bi-streaming data of mobile users

7.2 Clustering user behaviours by StrAP based on bi-streaming model

In the above section, we cluster the user behaviours by randomly selecting several steps due to the huge volume of data. In order to cluster the user behaviours on the whole streaming process, we employ an online clustering method called StrAP, as we introduced in section 5.

Table 3 shows the comparison of the stability and entropy values of evaluating StrAP online clustering results based on DDS model and EO model. Instead of showing the curves of stability and entropy, we give the number of steps when the stability and entropy based on DDS model is larger (or smaller) than that based on EO model. To investigate the sensitivities of DDS and EO model w.r.t.



(a) Stability (b) Entropy (c) AvgLength

Fig. 7. Evaluation of user behaviours clustering results in terms of Stability (a), Entropy (b) and Average length of objects (c), when AP was used on top of the DDS model and EO model applied on the **trimmed** bi-streaming data of mobile users with wsize=80,000

the window size, the parameter *wsize* is set to various values from 40,000 to 120,000.

Table 3. Comparison of the stability and entropy of StrAP clustering results based on DDS model and EO model

	Stability: Number of steps		Entropy: Number of steps	
wsize	$S(DDS) > S(EO)$	$S(DDS) < S(EO)$	$E(DDS) > E(EO)$	$E(DDS) < E(EO)$
40,000	140	49	60	129
60,000	153	27	19	161
100,000	145	13	7	151
120,000	156	6	3	158

Comparing the stability and entropy in Table 3, we see that DDS model has higher stability (second column) and lower entropy (fifth column) than EO model at most of the steps. We then can conclude that the DDS model supports better than EO model on clustering the user behaviours by online fashion. This conclusion is consistent with that we made in previous section 7.1.

We also see that DDS model is becoming better and better than EO model when the *window size* is increasing (the number of steps in the second and fifth column of Table 3 is increasing). This is because DDS model will take more transactions into account thus the users have more room and opportunity to accumulate their complete usage patterns when the *window size* increases.

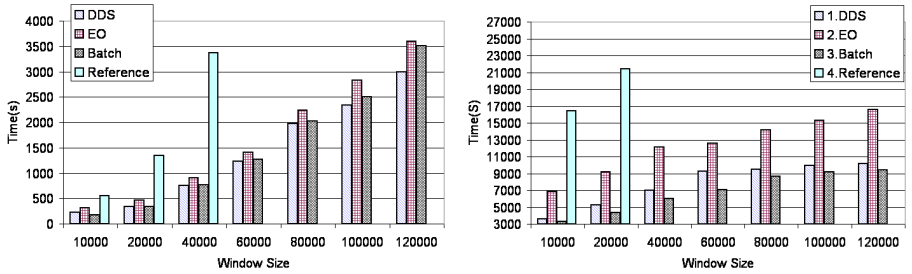
7.3 Efficiency of clustering user behaviours based on bi-streaming model

For the large volume streaming data, the efficiency of processing is very important as it determines whether we can get a timely summarization and identify

the useful patterns from the data. Figure 8 exhibits the running time of clustering user behaviours (using AP (a) and using StrAP (b)) on different models, the Batch, RO, DDS and reference model.

The x-axis is the *window size* increasing from 10,000 to 120,000. Generally, the running time of clustering increases as the *window size* becomes larger. DDS model always runs faster than EO model in both Figure 8 (a) using AP and (b) using StrAP.

In Figure 8 (a), although batch model uses slightly less time than DDS model when the window size is small, DDS model works better than Batch model as the window size is increasing. This phenomena can be explained on two aspects. First, in DDS model we prefer the up-to-date objects to the out-of-date ones. Therefore there will be less objects in DDS model than in batch model. It then takes less time for the clustering. Second, batch model has to wait until the buffer is ready. By contrast, DDS model deals with new transactions on the fly.



(a) AP (b) StrAP

Fig. 8. Running time of clustering user behaviours based on different models, (a) using AP and (b) using StrAP

Figure 8 (b) is the CPU time of using StrAP on different models. It should be noted that when AP is used for clustering the user behaviours, only several randomly selected steps are considered in both the DDS and batch model. Conversely, when StrAP is applied, the data managed by DDS model is always under processing; but the data managed by batch model is only processed at the end of each batch AP. Therefore, as shown in Figure 8 (b), StrAP costs more time on DDS model than that AP spends on Batch model.

In total, all these results verify that clustering on DDS model is more efficient than clustering on EO model.

8 Conclusion

In this paper, we have investigated the problems of modeling and clustering a special type of streaming data - bi-streaming data. In order to summarize the

bi-streaming data for clustering the users, we proposed three different models, the batch model, the EO model and the DDS model. Based on the models of bi-streaming data, clustering was carried out to discover user groups through employing a clustering method AP and an online clustering method StrAP. We tested the proposed models on a real-world data set which are the records of page request from mobile users of *France Telecom*. Experimental results showed that the DDS model outperformed the batch and EO models in both clustering quality and efficiency. Future work will be directed towards the classification and frequent pattern mining over bi-streaming data.

References

1. D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, 2003.
2. C. C. Aggarwal and P. S. Yu. A framework for clustering uncertain data streams. In *ICDE*, pages 150–159, 2008.
3. R. M. Aliguliyev. Performance evaluation of density-based clustering methods. *Information Sciences*, 179(20):3583 – 3602, 2009.
4. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, pages 1–16, 2002.
5. Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang. Multi-dimensional regression analysis of time-series data streams. In *VLDB*, pages 323–334, 2002.
6. G. Cormode, F. Korn, and S. Tirthapura. Time-decaying aggregates in out-of-order streams. In *PODS*, pages 89–98, 2008.
7. N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, pages 1–12, 2007.
8. M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002.
9. B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, February 2007.
10. C. Giannella, J. Han, J. Pei, X. Yan, and P. Yu. *Mining Frequent Patterns in Data Streams at Multiple Time Granularities*. In H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.), Next Generation Data Mining. AAAI/MIT, 2003.
11. P. Haghani, S. Michel, and K. Aberer. Evaluating top-k queries over incomplete data streams. In *CIKM*, pages 877–886, 2009.
12. J. A. Iglesias, P. Angelov, A. Ledezma, and A. Sanchis. Creating evolving user behavior profiles automatically. *IEEE Transactions on Knowledge and Data Engineering*, 99, 2011.
13. M. Meilă. The uniqueness of a good optimum for k-means. In *ICML*, pages 625–632, 2006.
14. X. Zhang, C. Furtlehner, J. Perez, C. Germain-Renaud, and M. Sebag. Toward autonomic grids: analyzing the job flow with affinity streaming. In *SIGKDD*, pages 987–996, 2009.
15. Y. Zhu and D. Shasha. Statstream: statistical monitoring of thousands of data streams in real time. In *VLDB*, pages 358–369, 2002.