Summarizing Multidimensional Data Streams: A Hierarchy-Graph-Based Approach

Yoann Pitarch, Anne Laurent, Pascal Poncelet

LIRMM - Univ. of Montpellier 2, CNRS
161 rue Ada, 34392 Montpellier, France
{pitarch,laurent,poncelet}@lirmm.fr

Abstract. With the rapid development of information technology, many applications have to deal with potentially infinite data streams. In such a dynamic context, storing the whole data stream history is unfeasible and providing a high-quality summary is required for decision makers. In this paper, we propose a summarization method for multidimensional data streams based on a graph structure and taking advantage of the data hierarchies. The summarization method we propose takes into account the data distribution and thus overcomes a major drawback of the Tilted Time Window common framework. Finally, we adapt this structure for synthesizing frequent itemsets extracted on temporal windows. Thanks to our approach, as users do not analyze any more numerous extraction results, the result processing is improved. Experiments conducted on both synthetic and real datasets show that our approach can be applied on data streams.

1 Introduction[1]

With the rapid development of information technology, many applications (web log analysis, medical equipment monitoring, etc.) have to deal with data streams. For instance, more than one billion of transactions are performed on the eBay website every day [1]. A data stream is defined as a potentially infinite sequence of precise and changing data arriving at an intensive rate. Due to the high-speed constraint, stream data can be read only one time (referred to as the one-pass constraint [2]) and storing the whole stream history is impossible. Nevertheless, the data stream history analysis would be helpful for decision makers. This naturally lead us to propose data stream summarization methods.

Moreover, most of stream data are multidimensional and can be considered at multiple levels of precision (referred as MD/MT data). For instance, a web log server registers visitor's IP, the date, the status code, ... Providing an on-line multidimensional and multilevel analysis on such data streams would be interesting in order to make profit of the OLAP technology in static datawarehouses.

To the best of our knowledge, only two approaches exist for summarizing multidimensional data streams thanks to OLAP technologies. The first of those

[1] Part of the MIDAS project founded by the french ANR agency (ANR-07-MDCO-008)

approaches is [3]. Since it is impossible to store the whole history of a stream, the temporal dimension is compressed thanks to the Tilted Time Windows [4] (TTW) (the most recent history is registered at the finest granularity and the older history is registered at coarser granularity). The authors made the most of the users habits in order to choose the materialized cuboids. In spite of an interesting architecture, the storage cost can be reduced. The authors of [5] overcame this drawback by introducing *precision functions* which define for each granularity level of every dimension the minimal interval of a TTW to avoid storing unqueried or computable data. Globally, the existing approaches focus on which cuboids must be materialized but none of them reconsider the use of TTWs. Even if this technique allows to reduce efficiently the temporal dimension, changing the time granularity at regular intervals can lead to an important loss of precision. Indeed, this mechanism does not take into account the data distribution. For instance, if an item rarely occurs in the stream, it could be useful to keep precise informations about its appearances. With the TTW mechanism, this information would be lost after the first aggregation.

In this paper, we propose a graph-based framework for summarizing MD/ML data streams which overcomes the major drawback of the TTW. If an item frequently appears in the stream, it is useless to conserve the precise history of its appearances. Thus, performing aggregations does not lead to an important loss of precision. Conversely, rare items should not be precociously aggregated because conserving their precise appearances could be useful for supporting decisions. Thereby, thanks to dynamic lists, aggregations are performed only if an item occurs in several close windows of the stream. On the contrary, non-close appearances are kept during a significant period.

Moreover, the frequent itemset mining on temporal windows can be considered as an interesting data stream summarization technique for two main reasons: the trend analysis and the context-awareness. Indeed, the trend analysis is one of the most frequent tasks performed by decision makers. Moreover, considering sets of items instead of isolated items allows the users to be more contex-aware. Thus, supported decisions are imporoved. Nevertheless the analysis of frequent itemsets extracted over temporal windows suffers from a major weakness: the independency of the extracted sets of itemsets. Thus, analyzing the stream evolution means observing these distinct sets of itemsets manually. The more thes number of separated sets to analyze, the more difficult it is. For instance, let us suppose that a website administrator hourly extracts the frequent itemsets from its network traffic. Then, if (s)he wants analyzing the yearly evolution of the visitor habits, (s)he has to consider the 8,760 separated sets of frequent itemsets. We show that our approach allows the unification of these separated and numerous sets and describe the minor modifications brought to the proposed framework in order to allow the synthesis of those independent sets of itemsets.

2 Problem Statement

In this section, the necessary concepts of our proposed method are defined. First, some definitions about the multidimensional and multilevel data are given. Then, the problematic of the frequent itemset mining [6] is extended to the frequent multidimensional itemset mining in data streams. Finally, a brief recall of the TTW mechanisms is given.

2.1 Dealing with MT/ML Data

Let $D = \{D_1, ..., D_M\}$ be a set of M dimensions. Every dimension D_i is defined over a (finite or not) set of values named $Dom(D_i)$. Generally, every dimension can be considered at several levels of granularity. These levels compose the hierarchy H_i of the dimension D_i with the following notations: max_i is the number of levels in H_i with $H_i^{max_i}$ the finest level and H_i^1 the coarsest. Note that for every dimension D_i we consider a wild-card value $*$ which can be defined as *all the values in* D_i. We note $x \in Dom(D_i^j)$ if x is defined on the level H_i^j. For instance, the hierarchy of a geographic dimension D_{Geo} could be $H_{Geo} = \{H_1 = ALL, H_2 = Continent, H_3 = Country, H_4 = City\}$ and we have $France \in Dom(D_{Geo}^3)$. With these notations, a (multidimensional) item t is defined as $t = (d_1, ..., d_M)$ so that for every $i = 1...M$, $d_i \in Dom(D_i)$. We say that t is a *Lowest Level Item* (LLI) if $\forall d_i \in 1 \ldots M$, $d_i \in Dom(D_i^{max_i})$. On the contrary, t is named an *High Level Item* (HLI). We note $father(t)$, the direct generalization of the item t. For instance, considering the hierarchies presented in the Figure 1, the item $t = (Water, LA)$ is a LLI and $father(t) = (Drink, USA)$.

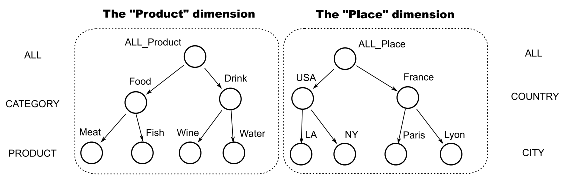


Fig. 1. The hierarchies used in our example

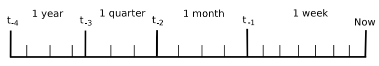
2.2 Mining Multidimensional Items in data stream

Initially introduced in [6], the frequent itemset mining problem in a database DB consists in the extraction of the multidimensional itemsets wich are supported by at least *minSupp* transactions. A *M-multidimensional k-itemset* is a set of k items described with M dimensions. The support of an itemset X, denoted by $supp(X) = count(X)/|DB|$ where $count(X)$ is the number of transactions in which X appears divided by the total number of transactions. An itemset X is frequent iff $supp(X) \geq minSupp$ where $minSupp$ is a user-defined numerical

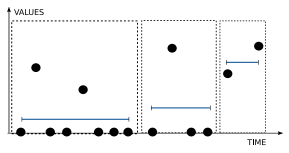
parameter.
To take the dynamic context into account, we adapt the problematic as follows. A data stream $S = B_0, B_1, ..., B_n$ is an infinite sequence of batches (temporal windows), where each batch is associated with a timestamp t, *i.e.* B_t, and n is the identifier of the most recent batch B_n. A batch B_i is defined as a set of transactions appearing over the stream at the i^{th} time unit. In such a context, the support notion must be redefined. $supp_{B_i}(X) = count(X)/|B_i|$ where $count(X)$ is the number of transactions of B_i in which X appears divided by $|B_i|$, the number of transactions of B_i.

2.3 The Tilted Time Window Framework

Since the volume of data generated by data streams is too huge to be totally materialized, a lossy compression of data stream history must be performed. In stream data analysis, users are usually interested in recent changes at a fine granularity, but long term changes at coarse scale. Thus, the literature proposes a model which takes inspiration of this: the *Tilted Time Window model* [4]. In fact, time can naturally be registered at different levels of granularity. The most recent history is registered at the finest granularity and the older history is registered at coarser granularity. The level of coarseness depends on the application requirements and on how old the time point is from the current time. Figure 2(a) provides an example of TTW.



(a) A natural TTW illustration



(b) The TTW model confronting to sparse data

Fig. 2. TTW: illustration and drawback

This technique allows an efficient compression of the temporal dimension but suffers from major weakness. Changing the time granularity at regular intervals can lead to an important loss of precision. For instance, let us consider the appearances of the item i presented as the point in Figure 2(b). As we can see, the item i rarely occurs in the stream. With classical TTW, these appearances would have been quickly aggregated. Indeed, the older the appearances, the more the number of values aggregated together. Thus, assuming that the displayed lines represent the aggregated values inserted in the TTW, we see that the resulted TTW is not representative of the appearance history of the item i in the stream. As a consequence, TTWs are specially well-adapted for balanced data distribution but not for continously-changing data distribution. Since stream data have a continuously changing distribution, this drawback is really critical.

3 The Raw Stream Data Summarization

Initially, the hierarchies associated to the dimensions compose the base-structure. These nodes are structural and do not store anything. Then, as multidimensional items keep coming, nodes storing the summary at different levels of granularity are created, updated or deleted dynamically. As previously discussed, storing their history in TTWs leads to in an important loss of precision with distribution-changing data streams. To overcome this drawback, the history of each LLI is conserved in dynamic lists storing the precise appearances. Thus, aggregations are not performed at regular time intervals but only when some conditions (e.g., temporal proximity between elements of the list) are validated. We discuss this mechanism in the Section 3.2. Higher granularity nodes store classical TTWs. Figure 3 displays a simple example of structure.

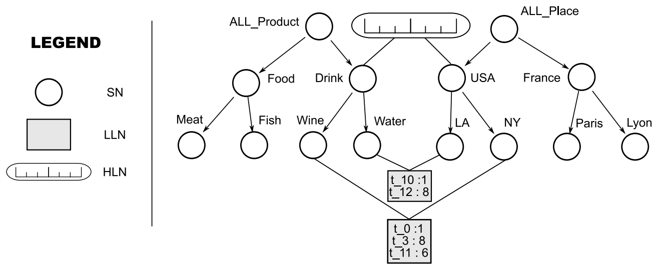


Fig. 3. A toy illustration of the proposed structure. Observing the list associated to the item $(Water, LA)$, we see that it occurs 1 time in the batch t_{10} and 8 times in the batch t_{12}. History of more general items (e.g., $(Drink, USA)$) are store in TTWs.

Due to the potentially infinite length of a stream, the simple accumulation of appearances in dynamic lists is impossible. Mechanisms for aggregating or merging data are proposed:

1. If the same item appears in close temporal windows, they are merged and the result of this merging is propagated along the item generalizations.
2. A maximum size for each list is fixed. When a list reaches its maximal size, the oldest element is deleted.

3.1 The Structure Description

Initially, the graph structure is composed by the dimension hierarchies. These nodes are name *structural nodes* SN. Histories of $LLIs$ are stored in nodes named the *Lowest Level Nodes* (LLN) defined as follows.

Definition 1. *Let $N_X = (X, Hist_X)$ be a LLN node so that X is a LLI and $Hist_X$ is a list containing pairs $< W : Count_W >$ where W is a time interval and $Count_W$ is the number of occurences of X_R in W. If W represent more than one time unit, we note respectively W_{beg} and W_{end} for the beginning and the end of the interval. On the contrary, the notation W is used.*

HLI are represented in our structure by *High Level Nodes* (HLN) defined as follows.

Definition 2. *Let $M_X = (X, T_X)$ be an HLN so that X is an HLI and T_X is a TTW storing the history of X.*

Now we described how these components interact.

3.2 Updating the Structure

A distance measure between LLI Since stream data arrives at a very low level of granularity, the number of potential items can be huge. [3] proposes to tackle this problematic by electing the the lowest level of granularity which is interesting for the user (named the m-layer). Thus, data are systematically aggregated to this level of granularity. Sometimes, users need to keep a track of precise data. In such a context, two items could be different but semantically near. A solution for considering these items as equivalent is to define a distance measure between them. If this distance is lower than an user defined threshold, the items are considered identically. We propose an hierarchy-based distance measure. Let $A = \{X_1^A, X_2^A, ..., X_N^A\}$ et $B = \{X_1^B, X_2^B, ..., X_N^B\}$ be two LLI. We define $dist(A, B)$ as:

$$dist(A, B) = 1 - \frac{\sum_{1 \leq j \leq N} prox(x_j^A, x_j^B)}{N}$$

With:

- N is the number of dimensions
- $$prox(x_1, x_2) = \begin{cases} \frac{1}{lv(NCA(x_1,x_2))^2} & \text{if } lv(NCA(x_1, x_2)) \neq ALL \\ 0 & \text{otherwise} \end{cases}$$
- *lv(x)* is the level of granularity of x (with $lv(x) = 1$ if x is a sheet of the hierarchy)
- $NCA(x_1, x_2)$ is the nearest common ancestor of x_1 and x_2

Property 1 *The dist function is a distance measure and satisfies the following conditions: (1)the symmetry, (2) $\forall x, y$ $dist(x, y) = 0 \Leftrightarrow x = y$ and (3) the triangular inequality.*

Example 1 *Let A=(Wine,Paris) and B=(Wine,Lyon) be two LLI. We have $dist(A, B) = 1 - \frac{1+0.25}{2}$. Indeed, we have $NCA(Wine, Wine) = IdProduct$ ($prox(Wine, WIne) = 1$) and $NCA(Paris, Lyon) = Country$ ($Prox(Paris, Lyon) = \frac{1}{2^2}$).*

Two items A and B are semantically close if and only if $dist(A, B) < distMax$ where $distMax$ is a user defined threshold. In our experimentations, we considered $distMax = 0.5$.

When the node N_X (where N_X is a LLN) already exists, an update of this node must be performed if X reappears (in the batch t for instance). Indeed, the pair $< t, count_t >$ must be added to $Hist_X$. Due to the storage constraint, a merge mechanism is proposed.

The Merge Operation If an item occurs in close time interval, it is unecessary to store its distinct appearances. A naive solution would be to perform this merge operation without any check. This solution would lead to mistakes during the propagation on the HLN. For instance, let us consider the Table **??** and the TTW shown on Figure **??**. With this TTW, an agregation is performed every three time units. The problem is that the aggregated value corresponding to the time interval $[T_{10}; T_{13}]$ cannot be inserted in the second window of the TTW because it overlaps the first two windows. Indeed, each value in the second window represents three time units (more generally, each value stored in a window k represents the aggregation of $W_1 \times ... \times W_{k-1}$ time units). So, a merging operation can be performed if and only if the impacted interval represents one temporal granularity of the TTW.

The proposed method for merging pairs stored in the $Hist_X$ of a node N_X and propagating the aggregated values along the generalization of X is described as follows. Firstly, the pair f arising from the merge is computed (line 1) and the related pairs are deleted from $Hist_X$ (line 2). This process begins a propagation along the generalization of the concerned item. So, it is necessary to seek the nodes sharing the same generalization. Each $Hist$ is scanned in order to locate entries which must take part in the aggregation mechanism (line 3 and 4). To ensure the consistency of the algorithm, a pair cannot take part in two different merge operations. So, each pair participating to a merge operation is marked (line 5). At last, the aggregated value is inserted at the appropriate position in the TTWs corresponding to the generalizations of X.

Example 2 *Let us consider the example displayed in Figure 3 and let us suppose that a merge has to be performed on the* (Water-LA) *node. Thus, the two pairs* $< t_{10} : 1 >$ *and* $< t_{12} : 8 >$ *are aggregated. Then, the search of nodes sharing the same generalization (i.e.,* (Drink-USA)*) find the node* Wine-NY*. Its list contains a pair* $< t_{11}, 5 >$*, which can participate to the aggregation.*

Limiting the Size of the Lists The merge mechanism allows for compression of lists but is insufficient to guarantee that the structure fits in main memory. Thus, additional methods must be proposed in order to avoid the memory overflow.

```
Algorithm 1: Merge
    Data: $N = (X, Hist_X)$ a $LLN$, $beg =< t_{beg} : count_{t_{beg}} >$,
          $end =< t_{end} : count_{t_{end}} >$
    Result: Performs the merge between $beg$ and $end$ and propagates along the
            generalization of $R$
1   $f =< t_f : count_f >=< [t_{beg}; t_{beg}] : agr(count_{t_{beg}}, ..., count_{t_{end}}) >$;
2   Removal of the merged pairs. ;
3   $M = \{p$ so that $p =< t_i : count_i >\in Hist_{X'}$ with $father(X) = father(X')$ and
    $t_i \in [t_{beg}; t_{end}]\}$;
4   $agr_{bro} = agr(count_i)$ (with $< t_i : counti >\in M$) ;
5   Mark those pairs ;
6   Insert agr($count_f$, $agr_{bro}$) in the TTW of $father(R)$ ;
```

Firstly, we previously said that a merging is performed if the interval represents one temporal granularity of the TTW but it is unrealistic to consider all the granularities. For instance, let us suppose that the TTW displayed in Figure 2(a) is used. Considering the whole TTW for the merging mechanism implies that we can potentially wait for 1 year before any aggregation and generalization. Due to the huge number of lists (i.e. the number of LLN), storing a so long history in each list is inconceivable. So, we introduce a user-defined numerical parameter, W_{MAX}, which means that the maximum size of the possibly merged interval is $W_1 \times ... \times W_{MAX} - 1$.

Secondly, the merging mechanism is not sufficient to limit the number of elements stored in a list. In fact, it is not possible to determine the data distribution in a stream and, consequently, it is impossible to predict the number of merging operations. So, in order to limit the size of the list, a user-defined numerical parameter, *MAX-SIZE*, is introduced. Intuitively, one may think that no more than *MAX-SIZE* elements per list are stored. This is not completely true. Indeed, since the *MAX-SIZE*th element in the list can be possibly merged in the future, we authorize *MAX-SIZE* $+(W_1 \times ... \times W_{MAX} - 1) - 1$ elements per list.

Example 3 *Let us consider that the TTW displayed in Figure 3 is used and that* MAX-SIZE= 3 *and* $W_{MAX} = W_3$. *With this parameter, the size on the list is at worst* $3 + (3 \times 3) - 1 = 11$.

The General Update Algorithm The general method for updating a LLN N can be described as follows. Firstly, the size of *Hist* is evaluated and compared to *MAX-SIZE*. If the size is smaller than *MAX-SIZE*, we check in the list if a merge operation is possible. If necessary, a merge operation is performed. Otherwise, the pair is inserted at the end of the *Hist*. If the size of the *Hist* is greater or equal to $MAX - SIZE$, we get the $MAX - SIZE^{th}$ element in *Hist* and we check if a merge operation is possible. Otherwise, we check if $t_c - t_m < (W_1 \times ... \times W_{MAX} - 1)$. This check allows us to verify if the $MAX{-}SIZE^{th}$ element could be merged in the future. Otherwise, it means that the list is full and that any element in the list can be merged. As a consequence, the oldest pair is deleted.

4 The Frequent Itemset Synthesis

As previously discussed, frequent itemsets extracted over temporal windows can be considered as an interresting data stream summarization technique. However, we discussed the difficulty for decision makers to analyze the numerous and independent set of results manually. In this section, the minor adjustments to perform in order to take into account such specific input are presented.

Storing frequent itemsets instead of items requires that dynamic lists (resp. TTWs) cannot be stored in LLN (resp. HLN). So, some definitions must be adapted. Indeed, LLN and HLN nodes are now considered as structural nodes and do not store any history.

Likewise $LLIs$, the history of each $LLIS$ is stored in nodes named the *Lowest Level Itemset Nodes* ($LLISN$).

Definition 3. *Let $N_X = (X : Hist_X)$ be a LLISN node so that X is a LLIS and $Hist_X$ is a set of pairs $< W : Supp_W >$ where W is a time interval and $Supp_W$ is the support of X in W. If W represents more than one time unit, we note W_{beg} (resp. W_{end}) the beginning (resp. the end) of the interval. On the contrary, the notation W is used.*

$HLIS$ are stored in nodes named the *High Level Itemset Nodes* ($HLISN$).

Definition 4. *Let $R = (X : T)$ be an HLR so that X is an HLI and T is a TTW.*

Methods presented in the Section 3 are easily transposable to the frequent itemset synthesis. Due to both the lack of space and the extreme proximity with the above-written algorithms, we do not present them.

5 Experiments

In this section, we present experiments to evaluate the feasability and performance of our approach. Throughout the experiments, we answer the following questions inherent to scalability issues: *Does the algorithm update its data structure before the arrival of the next batch? Does the mining process over a data stream remain bounded in term of memory ?* The experiments were performed on a Intel(R) Xeon(R) CPU E5450 @ 3.00GHz with 2GB of main memory, running Ubuntu 9.04. The methods were written in Java 1.6. Due to lack of space, all our experimental results are stored on a website [2]. Here, we present and discuss only the most representative.

[2] http://www.lirmm.fr/~pitarch/PAKDD10/experiments.html

5.1 Synthetic Datasets

The data stream is simulated using a multidimensional random data generator (following a Random Uniform Distribution). The convention for the data sets is as follows: D10L3C5W20T100SM10FP10 means that there are 10 dimensions, 3 granularity levels per dimension (except level *), the node fan-out factor (cardinality) is 5 (i.e., 5 children per node), there are 20K temporal windows of 100 tuples and the proper-approach parameters are SIZE-MAX=10 and the forgotten parameter equals 10. We performed extensive experiments in order to evaluate the approach. For this purpose, we tested the influence of numerous parameters: the number of potential items in the stream, the number of dimensions, the fan-out factor of the hierarchies, their depths and the MAX-SIZE parameter.

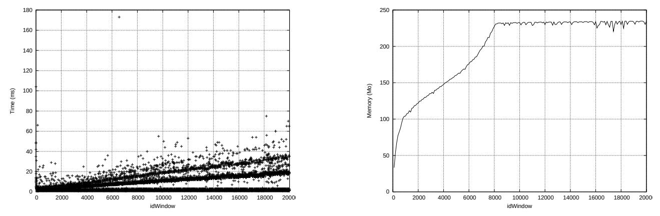
Results Figure 4 presents a representative result obtained during the experimentations. On Figure 4(b), we can observe 3 distincst behaviors. Firstly, the memory consumption quickly increases. Indeed, during that time, no merge is performed. During the second phase, the RAM consumption fairly increases. Indeed two concurrent phenomena occur: merges and *les listes qui se remplissent*. Finally, the memory usage stabilizes because all the potential LN are created and insertions in lists are balanced by merges.
Regarding the update time per window (Figure 4(a)), we notice 3 distinct time scales. The lowest one corresponds to a node creation or to a simple insertion in a list (performed almost instantaneously), the second one corresponds to merging and aggregation mechanisms (approximately 15ms) and the highest one is explainable by both insertionw and merges (approximately 20ms).

Due to the Random Uniform Distribution of data, paramaters which impact directly on the number of potential items to store have a logical influence on both time and memory consumption performances. Indeed, the more the number of items, the longer the time to perform a merge operation. Nevertheless, we notice that the performances become critical when the parameter values are extreme (e.g., when the depth of the hierarchies equals 7). In other experiments, results show the feasibility of the proposed method.

5.2 Real Dataset

The dataset used for these experiments comes from industrial pumps which transmit physical informations about pressure, external temperature, ... An item is described over 10 dimensions. We built arbitrarily hierarchies on these dimensions with the following characteristics. Every dimension has 3 levels of granularity and the average fan-out factor is 100. the dataset is dense. The input file were divided by windows containing 100 tuples.

Summarizing Items Here, we apply the multidimensional item summarization on this real dataset. Results are displayed in Figure 5. Regarding to the memory

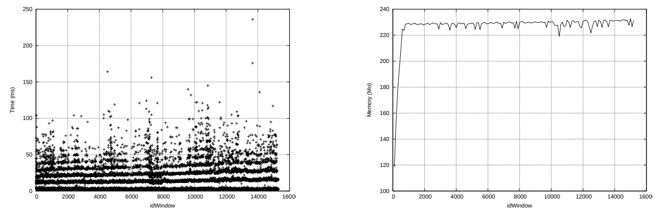


(a) Insertion time/window (3 dimensions)

(b) RAM consumption (3 dimensions)

Fig. 4. Influence of the number of dimensions(L3C5W20T100SM10FP150)

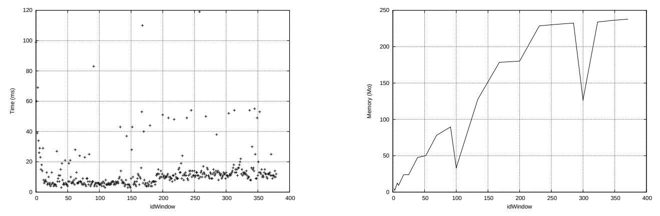
usage 5(b), we observe that is rapidly bounded. This can be explainable by the high density of the dataset. Concerning the insertion time , we can observe that (1) the average time is 50ms, (2) distinct time scales are observable, (3) the time is relatively stable and (4) this time is at worst 230ms.



(a) Insertion time/window

(b) Memory consumption

Fig. 5. Experiments conducted on raw data



(a) Insertion time/window

(b) Memory consumption

Fig. 6. Experiments conducted on frequent itemsets

Synthesizing Frequent Itemsets Here, we report results about multidimensional frequent itemset synthesis on this real dataset. The methodology used is the following one. We arbitrarily build multidimensional itemsets and customer sequences. The average number of items per itemset is 25 and the average number of customers per client is 100. Then, we applied a frequent itemset mining algorithm [3] with a minSupp=10%. The average number of frequent itemsets per window is approximatively 100. Finally, we run our algorithm on those frequent itemsets. Figure 6(b) displays our results about the memory consumption. We observe that the memory consumption stabilizes quickly because the frequent itemsets are globally the same on the whole data stream. Altough two off-peaks are observable, this can be explain by the garbage collector. Regarding the insertion time, we observe that the simple insertions or list creations are a little slower than with items. This is explainable by the higher complexity of itemsets in comparison to items. We can also observe several merging and generalization mechanisms.

6 Conclusion

In this paper, we tackle the problem of summarizing multidimensional and multilevel data stream thanks to a graph structure and provide efficient algorithm for updating this structure. Moreover, thanks to dynamic lists, we overcome the major drawback of the TTW: taking into account the data distribution. Finally, we show how frequent itemsets can be synthesized in order providing a comfortable solution for decision support. Our experiment study on both synthetic and real datasets shows that our summarization structure is efficient in both critical aspects in such context: time and space. Those results allow us to consider numerous possible extensions such as the adaptation of this structure to the sequential patterns.

References

1. eBay: 2006 annual report (2006)
2. Aggarwal, C.C.: Data Streams: Models and Algorithms. Advances in Database Systems. (2007)
3. Han, J., Chen, Y., Dong, G., Pei, J., Wah, B.W., Wang, J., Cai, Y.D.: Stream cube: An architecture for multi-dimensional analysis of data streams. Distribed Parallel Databases **18**(2) (2005)
4. Giannella, C., Han, J., Pei, J., Yan, X., Yu, P.: Mining frequent patterns in data streams at multiple time granularities (2002)
5. Pitarch, Y., Laurent, A., Plantevit, M., Poncelet, P.: Multidimensional data streams summarization using extended tilted-time windows. In: FINA. (2009)
6. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In Bocca, J.B., Jarke, M., Zaniolo, C., eds.: Proceedings of the 20th International Conference on Very Large Data Bases (VLDB 94). (12–15 1994) 487–499

[3] We use the implementation of FP-Growth provided by the Illimine project.