A Hierarchy-Based Method for Synthesizing Frequent Itemsets Extracted from Temporal Windows

Yoann Pitarch, Anne Laurent, Pascal Poncelet

A Hierarchy-Based Method for Synthesizing Frequent Itemsets Extracted from Temporal Windows

Yoann Pitarch, Anne Laurent and Pascal Poncelet
LIRMM
UM2 - CNRS
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. A practical and consistent summarization method is the extraction of the frequent itemsets over temporal windows. Nevertheless, this method suffers from a critical drawback: results pile up quickly making the analysis either uncomfortable or impossible for users. In this paper, we propose to unify these results thanks to a synthesis method for multidimensional frequent itemsets based on a graph structure and taking advantage of the data hierarchies. We overcome a major drawback of the Tilted Time Window standard framework by taking into account the data distribution. Experiments conducted on both synthetic and real datasets show that our approach can be applied to 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 leads to propose data stream summarization methods.

One of the three data stream summarization categories is the use of data mining algorithm. In particular, the frequent itemsets are useful for users interested by the trend discovery. This technique is faithful to a human memory mechanism because memories are indeed inter-linked. Thus, events are not stored independently in the human memory. In the cognitive psychology domain, this idea is known as the *constructive memory theory* [3], [4]. For instance, the mention of a familiar person involves the reconstruction of several memories about this person (his appearance, the last time we saw him, ...). Frequent itemset mining algorithms on data streams are a stream summarization method and can be subdivided into two categories. In the former situation, those algorithms are applied on the whole data stream. The result is not really interesting because there is no way to analyze the evolution of the stream. More recently, frequent itemset mining approaches are applied on temporal windows to handle stream data. Thus, the analysis of the stream evolution is feasible but this increasing list of independent results made the analysis either uncomfortable or impossible. For instance, let us suppose that a website administrator extracts everyday the frequent itemsets from its network traffic. If (s)he wants analyzing the yearly evolution of the visitor habits, (s)he has to consider the 365 independent sets of frequent itemsets. An interesting proposal could be to synthesize those independent results in order to provide a meta-summary of the stream.

Most of data streams are composed by several dimensions which can be considered at multiple granularity levels. For instance, a chain store can store the visitors' purchases with the associated date and store address, ... All of these dimensions can be observed at coarser level of granularity (e.g., a product can be also considered as a product category). Taking advantage of these multidimensional and multilevel properties would be interesting in order to synthesize the frequent itemsets.

To the best of our knowledge, only two approaches take advantage of these properties in a stream context and both are used for summarizing multidimensional raw data. The first approach is [5]. Since it is impossible to store the whole history of a stream, the temporal dimension is compressed thanks to the Tilted Time Windows [6] (the most recent history is registered at the finest granularity while the older history is registered at coarser granularity). Then, since it is unrealistic to store all the cuboids, the

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

authors consider the users habits in order to choose the materialized cuboids. In spite of an interesting architecture, the storage cost can be reduced. Indeed, some materialized elements can be computed or are never queried. The authors of [7] overcame this drawback by introducing *precision functions*. These functions define for each granularity level of each dimension the minimal interval of a Tilted Time Windows to avoid storing unqueried or computable data. The major drawback of this approach is that the precision functions are fixed in the initialization step.
Globally, the existing approaches focus on which cuboids must be materialized but none of them reconsider the use of the Tilted Time Windows. 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 apparition. With the Tilted Time Window mechanism, this information would be lost after the first aggregation.

In this paper, we take benefit of the multilevel and multidimensional properties of almost data streams and propose a graph structure for synthesizing frequent itemsets. Thus, we provide a unique framework and facilitate heavily the result analysis process. We overcome the major drawback of the Tilted Time Windows thanks to dynamic lists. Aggregations are performed only if an itemset is frequent in several close windows of the stream. On the contrary, non-close occurences are kept during a significant period.
Experiments conducted on synthetic and real datasets show that our aproach is adapted to the data stream context.

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 [8] is extended to the frequent multidimensional itemset mining in data stream. Finally, a brief recall of the Tilted Time Window mechanisms is given.

2.1. Hierarchy

Let $D = \{D_1, ..., D_n\}$ be a set of N dimensions. Each dimension D_i is defined on a (finite or not) set of values called $Dom(D_i)$. Generally, every dimension can be considered at several levels of granularity. We call these levels 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 each dimension D_i we consider a wild-card value * which can be defined as *all the values in* D_i. For instance, the hierarchy of a geographic dimension D_{Geo} could be $H_{Geo} = \{H_1 = ALL, H_2 = Continent, H_3 = Country, H_4 = City\}$. With these notations, a (multidimensional) item *t* is defined as $t = (d_1, ..., d_n)$ such that for every $i = 1...m$ and $l_i = 1...max_i$, $d_i \in Dom(D_i^{l_i})$, $D_i \in D$. We denote $father(i)$, the direct generalization of the item i. For instance, if we consider the hierarchies presented in the Figure 1, we have $father((A_{12}, B_{21})) = (A_1, B_2)$.

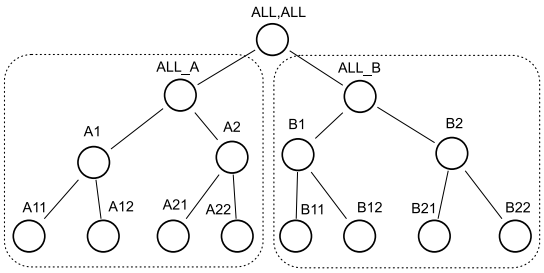


Figure 1. The hierarchy used in our example

2.2. Mining Multidimensional Itemsets in data stream

Initially introduced in [8], the frequent itemset mining problematic in a database D consists in the extraction of the itemsets wich are supported by at least $minSupp$ transactions. As we handle multidimensional items, this problematic can be adapted as follows. Let $\mathcal{I} = \{x^1, x^2, ..., x^m\}$ with $x^i = (x_1^i, ..., x_N^i)$ be a set of m distinct N-multidimensional items. A *k-itemset* is a subset of $\mathcal{I}$ with a cardinality equals to k. The support of an itemset X, denoted by $supp(X) = D_X/|D|$ where D_X is the number of transactions in which X occurs divided by the total number of transactions. An itemset X is frequent iff $supp(X) \geq minSup$ 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 tuples appearing over the stream at the i^{th} time unit. In such a context, the support notion must be redefined. $supp_{B_i}(X) = D_X/|B_i|$ where D_X is the number of transactions of B_i in which X appears divided by $|B_i|$, the number of transactions of B_i.

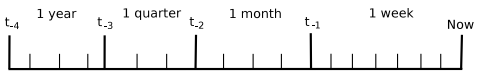


Figure 2. A natural *tilted-time window* model

2.3. Tilted Time Window

The volume of data generated by data streams is too huge to be totally materialized. 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 tis inspired by this idea: the *tilted-time window model* [6]. 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 provides an example of *tilted-time windows*.

This technique allows for efficient compression of the temporal dimension but suffers from a major weakness. Changing the time granularity at regular intervals can lead to an important loss of precision. For instance, let us consider the apparitions of the item i presented in the Table 1. With classical Tilted Time Model, it is impossible to detect that item i periodically appears in the stream. Indeed, old values would have been aggregated. Thus, the Tilted Time Window is specially well-adapted for balanced data distribution but not for continously-changing data distribution. Since stream date have a continuously changing distribution, this drawback is really critical.

i	T_0	10
	T_{100}	1
	T_{200}	6
	T_{300}	8
	T_{400}	4

Table 1. An illustration of the Tilted Time Window drawback

3. The Frequent Itemset Synthesis

Multidimensional and multilevel data stream summarization can be put together with the human memory mechanism. Indeed, human beings dayly receive a huge amount of information which are synthesized in memory. Thus, we naturally considered the cognitive psychology theory. A current and adjustable theory is known as the constructive memory [3], [4]. It means that memories are inter-linked. Indeed, events are not stored independently in the human memory. For instance, the mention of a familiar person involves the reconstruction of several memories about this person (his appearance, the last time we saw him, ...). The frequent itemset semantic is very close to this theory.

To synthesize multidimensional and multilevel frequent itemsets, we need a structure which can manage different heterogeneous objects such as hierarchies, multidimensional items or itemsets. For this, we take advantage of a graph structure. Initially, the based-graph is composed by nodes representing the hierarchies of the dimensions. These nodes are structural and do not store anything. Then, as multidimensional itemsets keep coming, nodes storing the summary at different levels of granularity are created, updated or deleted dynamically. In the existing approaches which take advantage of the multilevel property, the history is stored in Tilted Time Windows. As previously discussed, this mechanism leads to an important loss of precision with distribution-changing data streams. We overcome this drawback thanks to dynamic lists contained in the nodes representing the lowest granularity itemsets. Each time t an itemset is frequent, the timestamp and the corresponding support are inserted in the associated list. Thus, aggregations are not performed at regular time intervals but only when it exists close elements in one list. We discuss this mechanism in the Section 3.2. Higher granularity nodes store classical Tilted Time Windows. The Figure 3 displays an example of structure.

Due to the potentially infinite length of a stream it is impossible to simply accumulate occurrences in each list. In order to ensure the bounded size of the summary, some mecanisms for aggregating, merging or forgetting data are proposed:

1) If the same itemset is frequent in close temporal windows, they are merged and the result of this merging is propagated along the itemset generalizations.
2) Storing all the occurences per itemset is infeasible. Thus, a maximum size for each list is fixed. When a list reaches its maximal size, the oldest element is deleted.
3) Forgetting the old and non-repetitive events is a human behavior. Thus, if an itemset does not appear in the stream for a long period, its associated node is deleted.

3.1. The Structure Description

Initially, the graph structure is composed by the dimension hierarchies. We note this set of nodes X_{init}. By definition, stream data are observable at a very low level of granularity. We note them the *Base Items* (*BI*). More formally, a k-item $X = (x_1, ..., x_k)$ is a BI if $\forall x_i$, we have $x^i \in Dom(D_i^{max_i})$. The frequent itemsets which must be inserted in the graph are composed of BI. These base frequent itemset histories are stored in *List Nodes* (LN).

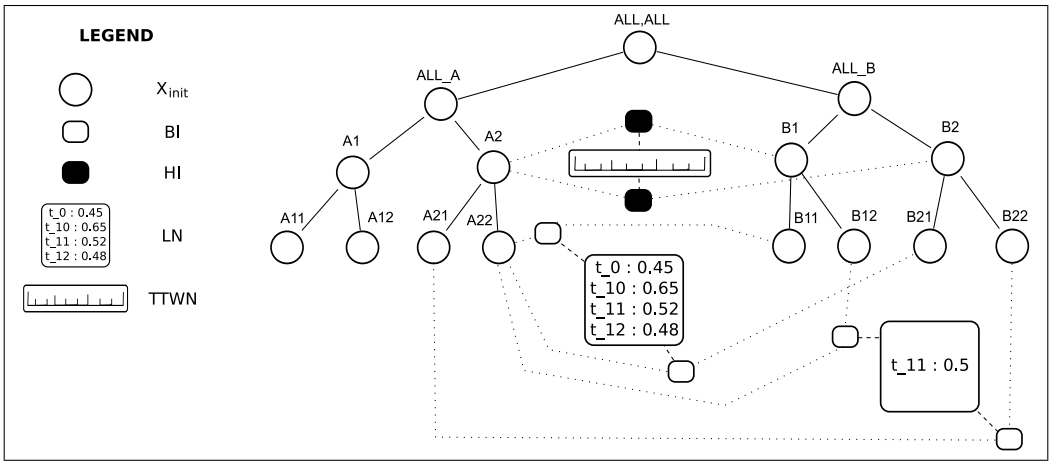


Figure 3. An illustration of the proposed structure

Definition 1. *Let $N = (X_N, SuppList_N)$ be a LN so that X_N is a* base frequent itemset *and $SuppList_N$ is a set of 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.*

In order to provide a multilevel vision of the itemsets, we also materialize the generalization of the BI and call them $Higher\ Item$ (HI). Thus, the base frequent itemset generalizations can be easily constructed thanks to the HI. The frequent itemset generalization history are stored in nodes called the *Tilted Time Window Nodes* ($TTWN$).

Definition 2. *Let $N = (X, T)$ be a $TTWN$ so that X is a base frequent itemset generalization and T is a Tilted Time Window storing its history.*

3.2. Updating the Structure

In this section, we describe the different mechanisms for updating the structure and bounding the its size.

3.2.1. A distance measure between itemsets. One on the most important caracteristics of a data stream is that data arrives at a very low level of granularity. This can be critical due to the huge number of potential itemsets. [5] proposes to tackle this problematic by electing the minimal interest layer (*m-layer*). The m-layer is the lowest level of granularity which is interesting for the user. Thus, data are systematically aggregated to this level of granularity. This can be problematic if the user want to keep a track of precise data. In such a context, two itemsets could be syntaxically different but semantically near. A solution for considering those itemsets as identical is to define a distance mesure between them. Thus, when this distance is lower than an user-defined threshold, those itemsets are considered identically. We propose an hierarchy-based distance measure. Let $A = \{X_1^A, X_2^A, ..., X_L^A\}$ et $B = \{X_1^B, X_2^B, ..., X_K^B\}$ be two itemsets. Figure 4 presents the proposed distance measure and the associated notations.

Example 1. *Let $A = \{(A_{11}, B_{21})\}$ and $B = \{(A_{12}, B_{21}), (A_{22}, B_{11})\}$ be two base itemsets. We have $X^{max} = B$ and $dist(A, B) = 1 - \frac{1+0.25}{2\times 2}$. Indeed, the item of B which maximizes the numerator is (A_{12}, B_{21}). Thus, we have $NCA(A_{11}, A_{12}) = A_1$ (its implies $prox(A_{11}, A_{12}) = \frac{1}{2^2}$) and $NCA(B_{21}, B_{21}) = B_{21}$ (its implies $prox(B_{21}, B_{21}) = 1$).*

Propriety 1. *Dist is a distance mesure and satisfy the three following conditions :*

1) *Symetry*
   $\forall X, Y$ two itemsets, $dist(X, Y) = dist(Y, X)$
2) *Separation*
   $\forall X, Y$ two itemsets, $dist(X, Y) = 0 \Leftrightarrow X = Y$
3) *The triangular inegality*
   $\forall X, Y, Z$ three itemsets, $dist(X, Z) \leq dist(X, Y) + dist(Y, Z)$

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

When the node X_R corresponding to the itemset X already exists in the structure, an update of this node must be performed if X appears in the batch t. Indeed, the pair $< t, count_t >$ must be added to the SuppList of X_R. However, storing this increasing list without any compression technique is impossible due to the storage constraint. Thus, a merging mechanism is proposed.

3.2.2. The Merging Operation. The goal of such an operation is to compress the list stored in each LN. If an

$$dist(A,B) = 1 - \frac{\sum_{x \in X^{min}} max\Big(\sum_{1 \leq i \leq N} prox(x_i, x_i^j) \,|\, x^j \in X^{max} \Big)}{|X^{max}| \times N}$$

Such as:

- X^{max} is the itemset with the highest cardinality
- X^{min} is the itemset with the lowest cardinality
- N is the number of dimensions
- $$prox(x_1, x_2) = \begin{cases} \frac{1}{l^2} & \text{if } l = level(NCA(x_1, x_2)) \neq ALL \\ 0 & \text{else} \end{cases}$$
- $level(x)$ is the level of granularity of x (with $level(x) = 1$ if x is a sheet in the hierarchy)
- $NCA(x_1, x_2)$ is the nearest common ancestor of x_1 and x_2

Figure 4. The distance measure definition

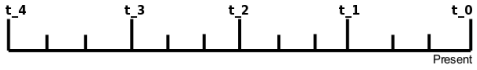


Figure 5. The Tilted Time Window used in the example

itemset occurs in close time interval, it is unecessary to store the distinct supports. A naive solution would be to perform this merge operation with any check but this solution would lead to mistakes during the propagation on the $TTWN$. For instance, let us consider the Table 2 and the Tilted Time Window shown on the Figure 5. With this Tilted Time Window, an agregation is performed every three time units. The aggregated value corresponding to the time interval $[T_{10}; T_{13}]$ cannot be inserted in the second window 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. As a consequence, a merging operation can be performed if and only if the impacted interval represents one temporal granularity of the Tilted Time Window.

R	T_0	0.3
	T_{10}	0.3
	T_{11}	0.6
	T_{12}	0.8
	T_{13}	0.4

Table 2. A SuppList illustration at T_{13}

Algorithm 1 presents the method used for merging some pairs in the $SuppList$ of a LN R and how this aggregated value is propagated along the generalizations of R. Firstly, the pair f arising from the merge is computed (line 1) and the related pairs are deleted from the SuppList (line 2). This process begins a propagation along the generalization of the concerned node. So, the nodes sharing the same generalization have to be sought. Each *SuppList* 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 Tilted Time Windows corresponding to the generalization of R.

Example 2. *Figure 3 illustrates the proposed algorithm. Suppose that a merging operation has to be performed on the* $\{(A_{22}, B_{11})(A_{22}, B_{21})\}$ *node. Thus, the three pairs* $< t_{10} : 0.65 >$, $< t_{11} : 0.52 >$ *and* $< t_{12} : 0.48 >$ *are aggregated. Then, the search of node sharing the same generalization (*$\{(A_2, B_1)(A_2, B_2)\}$*) find the node* $\{(A_{22}, B_{12})(A_{21}, B_{22})\}$*. In its SuppList, it exists a pair,* $< t_{11}, 5 >$*, which can be aggregated with the three pairs.*

Algorithm 1: Merge

Data: $R = (X_R, SuppList_R)$ a *LN*,
$beg =< t_{beg} : count_{t_{beg}} >$,
$end =< t_{end} : count_{t_{end}} >$
Result: Performs the fusion 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. ;
`// Searching pairs occurring in the list of low level siblings`
3 $M = \{p | p =< t_i : count_i > \in SuppList_{R'}$ with $father(R) = father(R')$ 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($supp_f$, agr_{bro}) in the TTW of $father(R)$;

3.2.3. Limiting the Size and the Number of the Lists. The merging mechanism allows for compression of the

lists but is insufficient to guaranty that the structure fit in main memory. As a consequence, some methods have to be proposed in order to avoid the memory explosion.

Firstly, as previously discussed a merging operation is performed if the interval represents one temporal granularity of the Tilted Time Window but it is unrealistic to consider all the granularities. For instance, let us suppose that the Tilted Time Window displayed in the Figure 2 is used. Considering the whole Tilted Time Window 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 LLR), 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 Tilted Time Window displayed in the Figure 5 is used and that* MAX-SIZE$= 3$ *and* $W_{MAX} = W_3$*. With this parameter, the size on the list is at worst* $3 + (3 \times 3) - 1 = 11$*. Figure 6 illustrates such a list.*

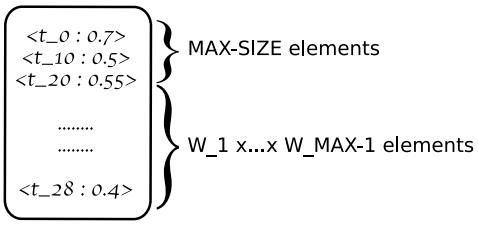


Figure 6. A dynamic list illustration

Finally, it is a human behavior to forget old and non repetitive events. Based on this idea, we propose to delete lists which are not updated for a long period. We call this parameter the *FORGETTING* parameter.

3.2.4. The General Update Algorithm. We present the general algorithm for updating a LLR (algorithm 2). Firstly, the size of *SuppList* is evaluated and compared to *MAX-SIZE* (line 1). 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 *SuppList*. If the size of the *SuppList* is greater or equal to $MAX - SIZE$, we get the $MAX - SIZE^{th}$ element in *SuppList* 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 first pair is deleted.

Algorithm 2: UpdateRule

Data: $c =< t_c : count_c >$ the pair to insert,
$R = (X_R, SuppList_R)$ a *LN*, W_{MAX}

```
1  if size(ListSupp_R) < MAX − SIZE then
2      if (∃l ∈ SuppList_R sa (t_c − t_l) ∈
       {W_1, .., W_1 × ... × W_MAX − 1}) then
3          Merge(R,l,c);
4      else
5          Add c at the end of SuppList_R;
6  else
7      m =< t_m : count_m > the pair of SuppList_R at
       the MAX − SIZE^th position ;
8      if (t_c − t_m) ∈ {W_1, .., W_1 × ... × W_MAX − 1})
       then
9          Merge(R,m,c);
10     else
11         if (t_c − t_m) < (W_1 × ... × W_MAX − 1) then
12             Add c at the end of SuppList_R;
13         else
14             Delete the first pair of SuppList_R;
```

4. Experiments

In this section, we present the experiments we conducted in order to evaluate the feasability and the performances 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.

4.1. Synthetic Datasets

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 *minSupp* parameter, the number of dimensions, the fan-out factor of the hierarchies, their depths, the MAX-SIZE parameter and the FORGETTING parameter. Due to lack of place, all our experimental results are available on a website [2]. Here, we present only the most representative ones. In the same way, analyzing precisely the impact of each parameter is infeasible. As a consequence, we synthesize the result analysis.

The data stream is generated by a multidimensional random data generator designed for testing cube computation and data mining algorithms on data streams. The convention for the data sets is as follows: D10L3C5I10S5SM10FP10 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 10K potential items in the stream, minSupp=5% and the proper-approach parameters are SIZE-MAX=10 and the FORGET-TING parameter equals 10. These parameters are the default parameter of our experimentations. The methodology used for extracting frequent itemsets is the following one. We arbitrarily build the transactions. The average number of items per transaction is 20 and the average number of transactions per batch is 1000. Then, we applied a frequent itemset mining algorithm [3] [**?**].

Figure 7 presents a representative result obtained during the experimentations. On Figure 7(a), we can observe 3 distincst behaviors. Firstly, the memory consumption quickly increases. This can be explained by the fact that during that time, none list is removed (because of the FORGETTING parameter). During the second phase, the RAM consumption fairly increases. Indeed two concurrent phenomena occur: the list suppressions (FORGETTING parameter) and the creation of new nodes in the structure. Finally, the memory usage stabilizes because all the potential LN are created and the list update is balanced by the list suppression.
Regarding the update time per window (Figure 7(b)), 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 suppressions and merging mechanisms (approximately 20ms).

Due to the Random Uniform Distribution of data, paramaters which impact directly on the number of potential itemsets to store have a logical influence on both time and memory consumption performances. Indeed, the more the number of itemsets, 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 others experiments, results show the feasibility of the proposed method.

2. http://www.lirmm.fr/~pitarch/SOCPAR09/index.html
3. We use the implementation provided by the Illimine project.

4.2. Real Dataset

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

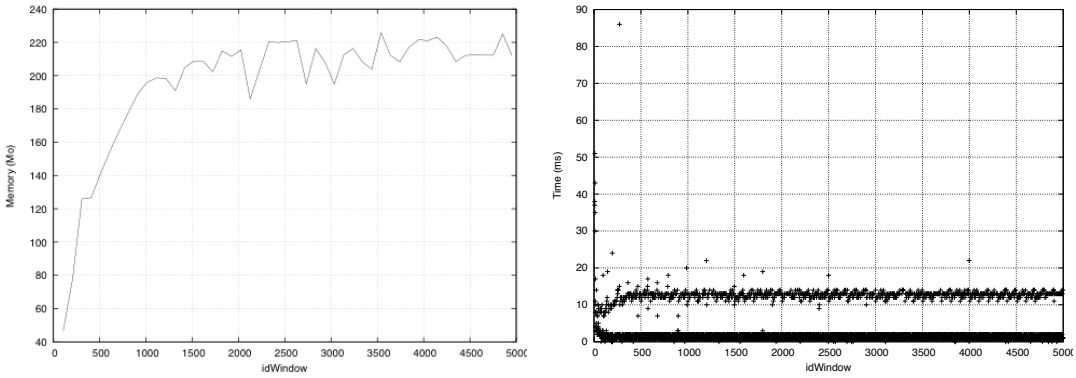
The methodology used for extracting frequent itemsets is the following one. We arbitrarily build the transactions. The average number of items per transaction is 25 and the average number of transactions per batch is 100. Then, we applied [?] with a minSupp=10%. The average number of frequent itemsets per batch is approximatively 100. Finally, we run our algorithm on those frequent itemsets. The Figure 8(a) 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.

5. Conclusion

In this paper, we tackle the problem of synthesizing the results of multidimensional frequent itemset mining over data streams. Thanks to a graph structure, we take advantage of the multilevel property of most of data and provide efficient algorithm for updating this structure. Moreover, thanks to dynamic lists, we overcome the major drawback of the Tilted Time Window: taking into account the data distribution. Our experimental 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. Firstly, it would be interesting to consider a summarization approach based on sequential patterns in order to take into account the sequentiality between the events. Then, a promising way would be to capture both frequent and uncharacteristic events in the stream. Finally, an efficient querry system would be suitable to fully exploit the proposed structure.

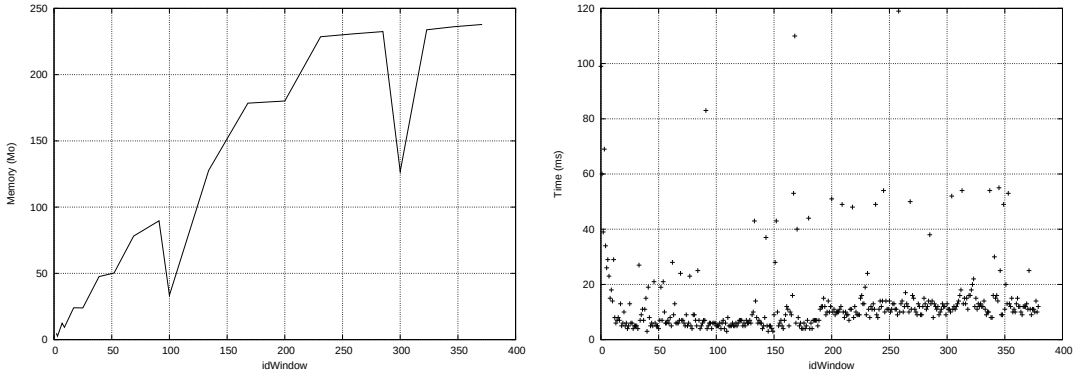
Acknowledgment

We would like to warmly thank Maximilien Servajean for the quality of the algorithm implementation.



(a) RAM consumption (b) Insertion time/window

Figure 7. Representative insertion time/window and RAM consumption experimental results



(a) RAM consumption (b) Insertion time/window

Figure 8. Experiments conducted on frequent itemsets

References

[1] eBay, "2006 annual report," 2006.

[2] C. C. Aggarwal, *Data Streams: Models and Algorithms*, ser. Advances in Database Systems, Springer, Ed., 2007. [Online]. Available: http://www.springer.com/west/home/default?SGWID=4-40356-22-107949228-0

[3] G. Miller, "Human memory and the storage of information," *Information Theory, IEEE Transactions on Information Theory*, vol. 2, no. 3, pp. 129–137, 1956. [Online]. Available: http://dx.doi.org/10.1109/TIT.1956.1056815

[4] D. L. Schacter and D. R. Addis, "The cognitive neuroscience of constructive memory: remembering the past and imagining the future." *Philos Trans R Soc Lond B Biol Sci*, vol. 362, no. 1481, pp. 773–786, May 2007. [Online]. Available: http://dx.doi.org/10.1098/rstb.2007.2087

[5] J. Han, Y. Chen, G. Dong, J. Pei, B. W. Wah, J. Wang, and Y. D. Cai, "Stream cube: An architecture for multi-dimensional analysis of data streams," *Distributed Parallel Databases*, vol. 18, no. 2, 2005.

[6] C. Giannella, J. Han, J. Pei, X. Yan, and P. Yu, "Mining frequent patterns in data streams at multiple time granularities," 2002.

[7] Y. Pitarch, A. Laurent, M. Plantevit, and P. Poncelet, "Multidimensional data streams summarization using extended tilted-time windows," in *FINA*, 2009.

[8] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB 94)*, J. B. Bocca, M. Jarke, and C. Zaniolo, Eds., 12–15 1994, pp. 487–499.