Speed Up Gradual Rule Mining from Stream Data! A B-Tree and OWA-based Approach

Jordi Nin$^{(1,2)}$, Anne Laurent$^{(2)}$ and Pascal Poncelet$^{(2)}$

$^{(1)}$LAAS, Lab. d'Analyse et d'Architecture des Systèmes
CNRS, Centre National de la Recherche Scientifique
7, Avenue du Colonel Roche
31077 Toulouse, France
jnin@laas.fr

$^{(2)}$LIRMM
Univ. Montpellier 2
CNRS UMR 5506
Montpellier, France
{laurent,poncelet}@lirmm.fr

Abstract

Gradual rules allow users to be provided with rules describing the ordering correlations among attributes. Such a rule is for instance given by *the higher the salary and the lower the number of cars, the higher the number of tourist travels*. Previously intensively used in fuzzy command systems, these rules were manually provided to the system. More recently, they have received attention from the data mining community and methods have been defined to automatically extract and maintain gradual rules from numerical databases. However, no method has been shown to be able to handle data streams, as no method is scalable enough to manage the high rate which stream data arrive at. In this paper, we thus propose an original approach to mine data streams for gradual rules. Our method is based on B-Trees and OWA (Ordered Weighted Aggregation) operator in order to speed up the process. B-Trees are used to store already-known gradual rules in order to maintain the knowledge over time, while OWA operators provide a fast way to discard non relevant data.

Keywords: Data Streams, Gradual Rules, OWA operators.

1 Introduction

Nowadays, data streams are gaining more attention as they are one of the most used ways of managing data such as sensor data that cannot be fully

stored. Such data, used in many domains such as telecoms, health, or system supervision (*e.g.* web logs), require novel approaches for analysis. Some methods have been defined to analyze this data, mainly based on sampling, for extracting relevant patterns [5, 10]. They have to tackle the problem of handling the high data rate, and the fact that data cannot be stored and has thus to be treated in a *one pass* manner [1]. For instance, recently new approaches were defined in order to extract association rules and sequential patterns over streams (e.g. [6, 24]). However, these methods do not allow users to be provided with the gradual rules that can be found in the data continuously arriving. Such rules point out the correlations between attribute orderings, and are thus referred to as multidimensional statistical correlations. For instance, such a rule could be of the form *the higher the salary and the lower the number of cars, the higher the number of tourist travels*, coming from the fact that for most of the data in the database, when the salary increases and the number of cars decreases, then the number of travels increases. The number of tuples from the database that assess this rule is considered as being the *support*, and can be computed by several manners [3, 17, 18].
It should be noted that extracting such rules is a very difficult task, as the search space grows exponentially. When mining data streams, this complexity is increased as the data must be handled on the fly and can be seen only once. In order to speed up the mining process, we thus propose to consider aggregation functions [32]. These functions are well-known techniques for data summarization, decision making, data mining, etc. In the context of gradual rule mining, we use them in order to reduce the number of attributes to consider. In a first approximation, we aggregate all the attribute values into a single datum using OWA (Ordered Weighted Averaging) operators [38, 39], which contrary to the arithmetic mean, allow us to weight the values as needed by the application (*e.g.* large values are more important than small ones). This fact is very important in our scenario because each attribute of the data stream has the same importance. By using OWA operators, it is possible to adjust the final aggregated value by taking into account the data distribution of each attribute. OWA operators have been used in many other scenarios where attribute correlations have a great impact on the final result of the data mining process. For instance, OWA operators have been applied to record linkage without common attributes [33] or schema matching [31].
After applying the OWA operator to reduce the number of attributes to be considered, tuples supporting a gradual rule are stored in a B-Tree structure to be retrieved very efficiently (the search cost is $O(log(n))$ where n

is the number of elements stored in the B-Tree). By using such B-Trees for ordering the tuples considering the aggregated OWA value as the key, we accelerate the process. Managing every new piece of data arriving on the stream indeed amounts to only look at the previous and next tuple in the B-Tree. From this operation, we can decide if a new tuple supports a given gradual rule (represented by the B-Tree) and must thus be inserted. Note that the insertion cost is also very low ($O(log(n))$). Furthermore, in order to maintain n as small as possible, *i.e.* in order that n could be stay in main memory, we use a well-known technique, called tilted-time window frames [20], for pruning the old B-Tree elements (tuples), thus, accelerating the B-Tree operations.
The rest of the paper is organized as follows. Section 2 recalls the problem statement by defining gradual rules (Section 2.1) and data streams (Section 2.2). Section 3 is devoted to the related work. Following in Section 4, we give some details about aggregation functions showing the reasons why OWA operators are the most suitable candidate. Then in Section 5, we provide a complete description of our approach. Experiments are described in Section 6, highlighting the relevance of our proposal. Finally, Section 7 draws some conclusions and describes some lines for future work.

2 Problem Statement

This section is devoted to the two main basic concepts that will be used in this paper: gradual rules and data streams.

2.1 Gradual Rules

Gradual rules allow us to define correlations between attribute orderings [16, 17]. Gradual rules are very close to action rules firstly introduced by Ras et al. in [26] and extended in [34, 35] in order to reclassify objects with respect to some distinguished atttribute (called a decision attribute). Intervention [11] or information changes [27] are also comparable to this framework. All these approaches could be useful in order to extract rules such as: *the higher the salary, the higher the number of cars.* They are defined over several attributes which are provided with an ordering relation. Each attribute is defined over a domain of values (mainly numerical, *e.g.* salary in euros) and a gradual item is defined as a pair (A, d) where A is the name of an attribute and d is the direction of the ordering. For instance $(Salary, +)$ stands for the gradual item *the higher the salary.* Then, a gradual itemset is defined as a set of gradual items $\{(A_1, d_1), \ldots, (A_k, d_k)\}$.

ID	Salary	Cars
t_1	2000	2
t_2	3000	3
t_3	3500	4
t_4	2500	4
t_5	1000	1
t_6	4000	3

Table 1: Database to be Mined: An Example

A gradual rule is given by a pair (s_1, s_2) referred to as $s_1 \rightarrow s_2$ where s_1 and s_2 are gradual itemsets.

The importance of a gradual itemset s is given by its *support* which is computed as the proportion of tuples from the database that support the gradualness. Supporting the gradualness means here that for every item (a, d) from s, for every pair of tuples (t, t') supporting s, $t.a \geq t'.a$ holds if $d = +$ and $t.a \leq t'.a$ holds if $d = -$. In this context, as shown in Section 3, many algorithms have been defined. It should be noted that, from a single database (set of tuples), several subsets of tuples can be found that support the gradualness. The support is thus defined as the *maximal* cardinality of the subsets: Let DB be the database, constituted of a set of n tuples $T = \{T_1, \ldots, T_n\}$ ($|DB| = n$), the support of a gradual itemset s is defined as $supp(s) = \frac{|T_s|}{|DB|}$ where $T_s \subseteq T$ is the maximal subset of tuples supporting s.

For instance, considering Table 1, the support of the itemset $\{(Salary, +)$ $(Cars, +)\}$ is 4/6 as tuples (t_1, t_2, t_3 and t_5) can be ordered so as to keep the increasing slope for both Salary and Number of Cars.

2.2 Data Streams

As data streams cannot be fully stored, one of the most important challenges is to find an efficient way for summarizing them. For this purpose, several methods have been proposed, mainly based either on data mining (association rules or sequential patterns [5, 10, 13]) or on sampling and aggregation (wavelets [12, 28], sketches [2, 4, 7], or sampling [37]).

When considering data mining approaches, the support computation is not the same as in the static context as it is not possible to divide by the total number of transactions since the stream is potentially infinite. The support of an itemset I is thus defined for every window W in the stream and is

denoted by $supp_W(I)$. More formally, $supp_W(I) = \frac{D_I}{|W|}$ where D_I is the number of transactions in W where I occurs and $|W|$ is the size of the window W. Similarly to the static environment, an itemset l is frequent if $supp_W(l) \geq minSupp$ where $minSupp$ is a user-defined parameter.
When mining data streams, three criteria must hold to ensure the quality of the data stream summary:

1. The construction and update time of the summary must be faster than the stream rate.

2. The closer the summary is compared to what a decision maker would retain from the stream, the better it is.

3. The memory consumption has to be bounded.

As explained in Section 1, the data mining community has recently paid attention to data the stream scenario. However, no method has been proposed for gradual rule mining in data stream due to scalability problems. For this reason, in this paper, we thus propose a very efficient gradual rule mining approach, and we show that it is scalable enough to fulfill the data stream criteria described above.

3 Related Work

Gradual itemset mining can be seen as a particular case of frequent pattern mining. However, the search space of gradual itemsets is larger than other frequent pattern mining scenarios, such as association rules. For this reason, most of the literature related to mining frequent patterns over the stream has been focused on mining frequent itemsets disregarding the concept of gradualness.
For instance, the first approach for finding frequent itemsets was proposed by [20] where they study the landmark model where patterns support is calculated from the start of the data stream. They also define the first single-pass algorithm for data streams based on the anti-monotonic property. Li et al. [19] use an extended prefix-tree-based representation and a top-down frequent itemset discovery scheme. In [29], authors propose a regression-based algorithm to find frequent itemsets in sliding windows. Chi et al. [8] consider closed frequent itemsets and propose the closed enumeration tree (CET) to maintain a dynamically selected set of itemsets. In [14], authors consider an FP-tree-based algorithm to mine frequent itemsets at multiple time granularities by a novel logarithmic tilted-time window technique.

More recently new approaches have been defined in order to extract more complex patterns such as sequential patterns over the stream. For instance, in [24], the authors propose a new approach, called SPEED (*Sequential Patterns Efficient Extraction in Data streams*), to identify maximal sequential patterns over a data stream. The main originality of this mining method is that a novel data structure is used to maintain frequent sequential patterns coupled with a fast pruning strategy. In [21], another approach is proposed, based on sequence alignment for mining approximate sequential patterns data streams.
Tree structures [9] are very common structures in association rule mining. It is well-known that B-tree [22, 23] and other structures as FP-tree [15], T-Tree or P-tree [36] achieve a good time performance when they are applied to compute the support of a set of association rules. However, to the best of our knowledge, they have never been applied to gradual rule mining in order to speed mining up and maintaining processes.
The research related to mining gradual itemsets has focused on the static databases environment. For instance, in [16] it is assumed that the entire database is known in advance. In [17] the database has to be sorted several times (once per attribute), making this approach unfeasible for real time scenarios. To the best of our knowledge, there is no related literature about mining gradual itemsets in data streams due to the high complexity of this problem and the strong requirements of data streams: algorithms has to be executed in real time scenarios and data has to be summarized over the time. Therefore, in the remaining of this paper we will provide some details about aggregation functions and their utility for our scenario, as well as a methodology for maintaining gradual itemsets in data stream scenarios.

4 Aggregation Functions

Aggregation functions [32] are functions used for information fusion. They typically combine N data values supplied by N data sources into a single datum. The simplest and most widely used aggregation functions are the arithmetic and the weighted mean. In [38], Yager introduced another function, the OWA operator, to model aggregation in intelligent systems. The definitions of these functions are given below. As they require a weighting vector, firstly we provide such definition. All definitions in this section assume that $a_1, \ldots, a_N$ are the N values to be fused.

Definition 1 *A vector $v = (v_1, \ldots, v_N)$ is a weighting vector of dimension N if $v_i \in [0,1]$ and $\sum_{i=1}^{N} 1$.*

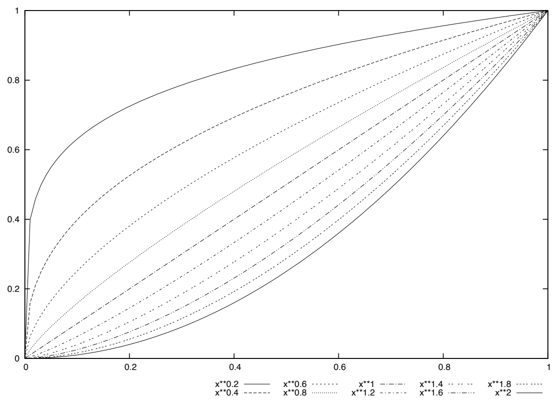


Figure 1: Graphical representation of Yager α-quantifiers.

Definition 2 *A mapping AM:* $\mathbb{R}^N \rightarrow \mathbb{R}$ *is an arithmetic mean of dimension* N *if* $AM(a_1, \ldots, a_N) = (1/N)\sum_{i=1}^{N} a_i$.

Definition 3 *Let* p *be a weighting vector of dimension* N*. A mapping WM:* $\mathbb{R}^N \rightarrow \mathbb{R}$ *is a weighted mean of dimension* N *if* $WM(a_1, \ldots, a_N) = \sum_{i=1}^{N} p_i a_i$.

Definition 4 *Let* w *be a weighting vector of dimension* N*. A mapping OWA:* $\mathbb{R}^N \rightarrow \mathbb{R}$ *in an ordered weighting average of dimension* N *if* $OWA\ (a_1, \ldots, a_N) = \sum_{i=1}^{N} w_i a_{\sigma(i)}$*, where* σ *is a permutation such that* $\forall i \in [1, N-1] a_{\sigma(i)} \geq a_{\sigma(i+1)}$.

While WM aggregates values considering the importance (or reliability) of the data sources, OWA operators permit the user to aggregate the values giving importance to large (or small) values. This fact is very important in our scenario because each attribute of the data stream has the same importance. But by using the weighting vector of OWA operators, it is possible to adjust the final aggregated value taking into account the data distribution of each attribute. Of course, it could be possible to get a scenario where each attribute of the data stream has different importance. We can model this scenario using the WOWA operator [30], a linear combination of the WM and OWA.
Another important drawback of WM is that a different weighting vector must be defined for each gradual itemset to mine. Of course, with the OWA definition provided above we have the same problem, *i.e.* we have to define a weighting vector for mining gradual itemset of two attributes, another

for three attributes, and so on. However, as OWA operator is symmetric, we can define it in a different way when the number of data sources is not known in advance. This definition is based on fuzzy quantifiers.

Definition 5 *A function* $Q : [0,1] \to [0,1]$ *is a* regular monotonically non-decreasing fuzzy quantifier *(non-decreasing fuzzy quantifiers for short) if it satisfies: (i)* $Q(0) = 0$*; (ii)* $Q(1) = 1$*; (iii)* $x > y$ *implies* $Q(x) \geq Q(y)$*.*

An example of family of fuzzy quantifiers Q is given below. Such family corresponds to Yager α-quantifiers.

$$Q^{\alpha}(x) = x^{\alpha} \text{ for } \alpha > 0$$

A graphical representation of this fuzzy quantifier is given in Figure 1 for some particular α ($\alpha \in 0.2, 0.4, \ldots, 1.8, 2.0$). We can observe that for small α values, the function increases quickly near $x = 0$, whereas the increase is smoother for larger values of α. In the former case, the quantifier gives a larger importance to the small values and thus it is more sensitivity to changes in such values. On the contrary, in the latter case, the quantifier gives more importance to the large values and its sensitivity increases for such values.
Using fuzzy quantifiers, the OWA operator [39] is defined as follows.

Definition 6 *Let* Q *be a non-decreasing fuzzy quantifier, then* OWA_Q*:* $\mathbb{R}^N \to \mathbb{R}$ *is an* Ordered Weighted Averaging (OWA) operator *if* $OWA_Q\ (a_1, ..., a_N) = \sum_{i=1}^{N}(Q(i/N) - Q((i-1)/N))a_{\sigma(i)}$ *where* σ *is a permutation such that* $a_{\sigma(i)} \geq a_{\sigma(i+1)}$*.*

In our experiments we will use this latter OWA definition with the Yager α-quantifiers.

5 Mining Gradual Rules Over Data Streams

In this section, we firstly give an overview of our approach focusing on the requirements needed in a data stream mining scenario. Following, we explain in detail the gradual rule mining algorithms used in our approach. Finally, we provide the complexity of the most costly algorithm.

ID	Salary	Cars	OWA (B-Tree Key)
t_1	0.4	0.4	0.4
t_2	0.6	0.6	0.6
t_3	0.7	0.8	0.75
t_4	0.5	0.8	0.65
t_5	0.2	0.2	0.2
t_6	0.8	0.6	0.7

Table 2: Normalized Database (min_Salary=0, max_Salary=5,000, min_Cars=0, max_Cars=5), and OWA value.

5.1 Basic Idea

We consider databases such as described in Table 1 where attribute values have been normalized in $[0, 1]$, as shown by Table 2. For this purpose, we consider a minimum and maximum value for every attribute[1].
In order to have a global idea of each tuple for ordering them, we compute a summary using an OWA operator. Note that it is very easy to manage both increasing and decreasing gradualness. Indeed, as OWA weights give importance to small or large values, it is not the same to mine a gradual itemset like $\{(Salary, +)(Cars, +)\}$ or $\{(Salary, +)(Cars, -)\}$. For the former case, we have to compute the OWA value as $OWA_Q(a_{salary}, a_{cars})$, whereas, for the latter one, we have to compute the OWA value as $OWA_Q(a_{salary}, (1-a_{cars}))$. For instance, the last column of the Table 2 reports the value computed by giving the same weight to every attribute value. Example 1 shows the process for mining the gradual itemset $\{(Salary, +)(Cars, +)\}$.

Example 1 *Consider each attribute of the normalized database of Table 2 as the data coming from the data stream, and suppose that the gradual itemset to mine is GI: $\{(Salary, +)(Cars, +)\}$. At the beginning the corresponding B-Tree for GI is empty. At time ts_1, the tuple t_1 arrives and $OWA_Q(t_1)$ is computed. As the B-Tree is empty, t_1 is inserted in the B-Tree root node, this is illustrated in Figure 2.(a). At time ts_2, the tuple t_2 arrives and $OWA_Q(t_2)$ is computed. As $OWA_Q(t_1) \leq OWA_Q(t_2)$, we have to check that $t_1.Salary \leq t_2.Salary$ and $t_1.Cars \leq t_2.Cars$ hold. As both conditions are fulfilled t_2 is inserted in the B-Tree (as it is shown in Figure 2.(b)). This process is repeated at time ts_3 with tuple t_3 checking that t_3 attributes*

[1]If a tuple has an attribute lower or higher than the minimum or maximum value predefined, such value can be normalized as 0 (for the minimum) or 1 (for the maximum).

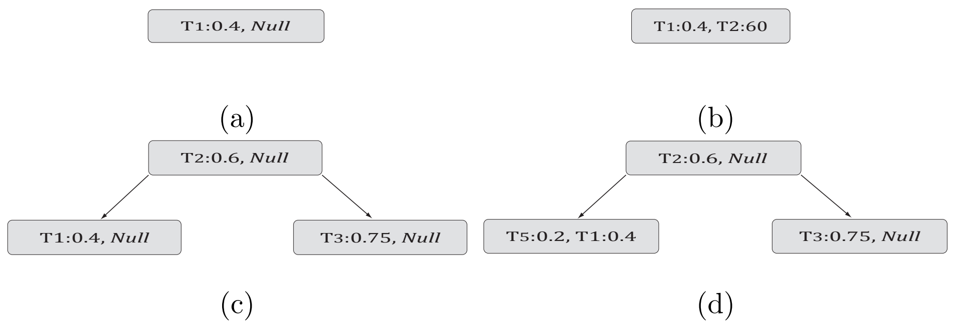


Figure 2: B-Tree Insertion Example.

are larger than t_2 attributes (Figure 2.(c)). Then, at time ts_4, the tuple t_4 arrives and the $OWA_Q(t_4)$ is computed. As $OWA_Q(t_2) \leq OWA_Q(t_4) \leq OWA_Q(t_3)$, we have to check that $t_2.Salary \leq t_4.Salary \leq t_3.Salary$ and $t_2.Cars \leq t_4.Cars \leq t_3.Cars$ hold. As $t_2.Salary = 0.6$ and $t_4.Salary = 0.5$ the first condition does not hold and tuple t_4 cannot be inserted in the B-Tree. After that, tuple t_5 is inserted at time ts_5 because $OWA_Q(t_5) \leq OWA_Q(t_1)$ and $t_5.Salary \leq t_1.Salary$ and $t_5.Cars \leq t_1.Cars$ hold, this is depicted in Figure 2.(d). Finally, at time ts_6 tuple t_6 is discarded in a similar way as t_4.

The **key idea** of our approach is based on the property reported below.

Property 1 *Let s be a gradual itemset, b be the B-Tree containing tuples from the data stream holding s. Let Q be the fuzzy quantifier used in the OWA operator. Let t_{new} be a new tuple arriving on the stream. Let pos be the insertion position of t_{new} in b considering the OWA summary of t_{new}. Then, whatever the OWA weights, if t and t' are previous and subsequent tuples stored in b respectively, such that $OWA_Q(t) \leq OWA_Q(t_{new}) \leq OWA_Q(t')$, we have: if there exists $a \in s$ such that $t.a \leq t_{new}.a \leq t'.a$ does not hold, then there does not exist pos' in b that t_{new} can support s considering the relative order established by b. Therefore, t_{new} can be discarded. This property holds due to the fact that all attributes of the tuples stored in the B-Tree are sorted in a non-decreasing order to hold s.*

Proof 1 *Let o be the $OWA_Q(t_{new})$ summary for the tuple t_{new}. Let t and t' be previous and subsequent tuples stored in b. Now, let us assume that t_{new} can be inserted in b. Then, o gives us the unique possible position in*

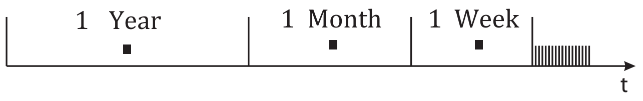


Figure 3: Tilted-Time Window Frames.

the B-Tree order for the tuple t_{new}. To prove this, let us imagine the most difficult case, where t and t' attributes are equal to t_{new} except one that has to be smaller (larger) for t (t'). As we are computing the OWA summaries using the same weights, we have $OWA_Q(t) \leq OWA_Q(t_{new}) \leq OWA_Q(t')$ because there is at least one attribute in t (or t') smaller (or larger) than t_{new}. Moreover, we can ensure that t_{new} can only be inserted between t and t'. Note that, if t_{new} cannot be inserted in the provided position by the OWA summary, then t_{new} cannot be inserted in any other place because the order of the B-Tree tuples. This statement holds due to the order inside the B-Tree, i.e. *the previous tuple of t has at least one attribute smaller than t, and of course, smaller than t_{new}. The same reasoning can be applied to the subsequent tuple of t'. However, this proof only holds if all OWA weights are different from 0.*

Thanks to Property 1, we know that if the ordering among the new tuple and the previous and subsequent one does not hold for at least one item, the tuple can be immediately discarded, thus speeding up the process. Otherwise, the tuple can be inserted in the B-Tree. The complexity of our proposal is related with the cost of the B-Tree operations, because the cost of such operations depend on the number of tuple whereas the complexity of the OWA summary depends on the number of attributes (much smaller than the number of tuples). For this reason, we can consider the OWA summary cost as a constant value. Then, as B-Tree operations have a cost equal to $O(log(n))$ for each gradual itemset to mine. We can say that this process is very efficient and it can be affordable in a real time scenario. Therefore, the first stream mining requirement holds.
As explained in Section 2.2, we have to ensure that our approach is able to manage every new tuple faster than the stream rate using a bounded

amount of memory and maintaining in memory a detailed information of the most recent tuples. Yet, if we repeat this process a large number of times, the number of tuples stored in the B-Trees will grow and we will need a large amount of memory. In order to avoid this fact, B-Trees are pruned by compressing old tuples. Such a compression is done computing an attribute wise summarization using an OWA operator. This summary is inserted in the B-Tree and it will be preserved over the time. This compression technique, also known as tilted-time window, was proposed by [20] and it has been used in many other works as [8, 14, 25] using different summarization procedures. Figure 3 shows a tilted-time windows table: the most recent tuples are preserved, then, in another level of granularity, the last week, the last month and the last year are summarized. Based on this model, one can compute data analysis in the last week with a maximum precision, in the last month with the precision of a week, and so on.
The rationale of this process is that old tuples are removed from the B-Trees, saving memory space, but partial information of those tuples is preserved in the compressed tuple. Using this technique, we ensure that the amount of memory needed for our approach is bounded and the most recent tuples are preserved in the B-Trees. Therefore, using this compression technique, the second and the third stream mining requirements are also hold.
It should be noted that, in a stream mining scenario, it may be the case that we accept a tuple arriving at the stream that complies with the previous tuples although it would be better to discard it because in the future we will have to discard a large amount of tuples decreasing in this way the final support. For this reason, the approach used to compute the support has to be considered as a heuristic.

5.2 Algorithm Definition

In this section, we provide the algorithms and all the details of our proposal. We assume that tuples coming from the stream are already normalized. Algorithm 1 is in charge of initializing the gradual itemsets and B-Trees lists (lines 2-3). Basically, such initialization consists in creating all the possible gradual itemsets taking into account the number of attributes of tuples, and creating for each one an empty B-Tree. Following, we have to compute when the first pruning process will be executed (line 6). As we are interested in maintaining in memory all tuples belonging to a certain period of time (the most recent frame of the tilted-time window), we have to prune the B-Trees for the first time when we have in memory tuples of two window frames. In the main loop of Algorithm 1, we process the data

```
Algorithm 1: Data Stream Mining
    Data: s: Stream, q: Quantifier, w: Window
1   begin
2       g = Gradual itemset list;
3       b = B-Tree list;
4       InitializeGradualRule(g);
5       InitializeBTree(b);
6       np = now() + 2w;
7       while (!s.empty()) do
8           GradualRuleProcessing(g,b,s,q);
9           if (now() ≤ np) then
10              BTreePruning(b,w);
11              np = now() + w;
12  end
```

stream (line 8) and every tilted-time window frame we execute the B-Tree pruning process (line 9-11). This last part of this algorithm is also used to recompute the next period of time when B-Trees will be pruned again (line 11).

Algorithm 2 manages the gradual rule mining process. Initially, it reads the data stream, storing in the array a all attribute values (line 3). Next, for each gradual itemset to mine we have to do the following process: Firstly, we have to reverse all attribute values corresponding to the case (a_i, -), *i.e.* we have to compute the value $a_i = 1 - a_i$. Then, $OWA_q(a)$ is computed using the fuzzy quantifier q (line 6), such quantifier is a user parameter and it has to be selected based on data stream value distribution as we have explained in Section 4. Afterwards, the previous and the subsequent B-Tree nodes are recovered (lines 7-8) and, as we have explained in Property 1, each attribute the of new tuple is checked with the previous and subsequent tuples (line 9). Then, if the new tuple can be added in the B-Tree, it is added and the gradual itemset support is updated (line 11), if not it is discarded.

Finally, Algorithm 3 is devoted to the B-Tree pruning process, it is executed at each tilted-time window frame. In this algorithm, for each tuple, if it is out-of-date of the current window frame (line 4), the corresponding B-Tree node is removed. If not it is preserved until the next pruning process.

The most costly algorithm is the Algorithm 2. The complexity of this algorithm is related to the number of gradual itemsets (s) and to the number of tuples that we have to process in a window frame (n). B-Tree operations

Algorithm 2: Gradual Rule Processing

```
Data: g: Gradual itemset list, b: B-Tree list, s: Stream, q:
      Quantifier
1  begin
2      while (!s.empty()) do
3          a=read(s);
4          for $i \leftarrow 1$ to g.Size do
5              ReverseValues(g.get(i), a);
6              o = ComputeOWA(a, q);
7              prev = PreviousNodeSearch(o, b.get(i));
8              susbseq = SubsequentNodeSearch(o, b.get(i));
9              if (checkRule(g.get(i), b.get(i), a, prev, subseq)) then
10                 b.get(i).addNode(o, a, timestamp);
11                 UpdateSupport(g.get(i));
12 end
```

used in this algorithm (search and insertion) have a complexity equal to O(log (n)). In the worst scenario, for each new tuple, we have to execute two B-Tree operations (one search and one insertion) for each gradual itemset to mine. Therefore, the complexity of this algorithm is equal to O(2s log (n)). As Algorithm 2 has a sub-linear complexity and the tilted-time window technique allows us to keep n as small as possible, we argue that it is affordable for most of real time scenarios.

6 Experiments

In this section we describe the experiments that we have been carried out to test the efficiency of our approach.

6.1 Dataset and Parameter Description

In our experiments, we have used a dataset coming from a train sensor network installed in some trains of a national railway company. We consider here that each sensor provides a new value every 100 miliseconds (*i.e.* 10 new values per second), in total we manage more than 80,000 tuples. In the experiments, three subsets of 5, 8 and 10 temperature sensors installed in the bogie wheels have been selected. The number of gradual itemsets

Algorithm 3: B-Tree Pruning

```
   Data: b: B-Tree List, w: Window
1  begin
2      for $i \leftarrow 1$ to $b.size$ do
3          for $j \leftarrow 1$ to $b.get(i).size$ do
4              if $(b.get(i).get(j).timestamp \leq now() - w)$ then
5                  RemoveNode(b.get(i), j);
6  end
```

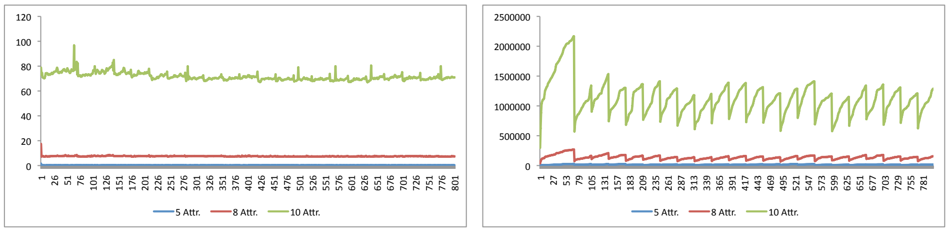
5 sensors	8 sensors	10 sensors
116	3,272	29,512

Table 3: Number of gradual itemsets for each experiment.

that we mine in each case is depicted in Table 3. We have normalized the values provided by each sensor within the $[0, 1]$ interval. Usually, the values provided by the temperature sensors are small, and only when the brakes are working they increase their value. Therefore, we are interested in having a very low granularity in the small values, because they are the most frequent ones. For this reason, we have used in our experiments a Yager α-quantifier with a low α, $Q^{0.5}(x)$. Of course, other α values can be considered, for instance, we can consider a quantifier with a large α ($Q^2(x)$). In this case the OWA summary will be very sensitive to small changes in large values, but such values are infrequent in our data, then the support of the gradual itemsets will be underestimated.
In our software, all algorithms are implemented as independent threads. Therefore, in our experiments, we have simulated a real time environment where tuples are arriving at a high rate and gradual rule mining and B-Trees pruning processes are executed in the same processor at the same time.

6.2 Results Analysis and Discussion

In this section, we present a complete analysis of the experiments carried out in this work, as well as, a short discussion about why the results obtained show that our approach is suitable for the stream mining scenario. The experiments were performed using a Sun Java Virtual Machine and a Debian Linux operating system in a Dell Precision 390 workstation with only one 32 bits CPU and one giga of RAM memory.



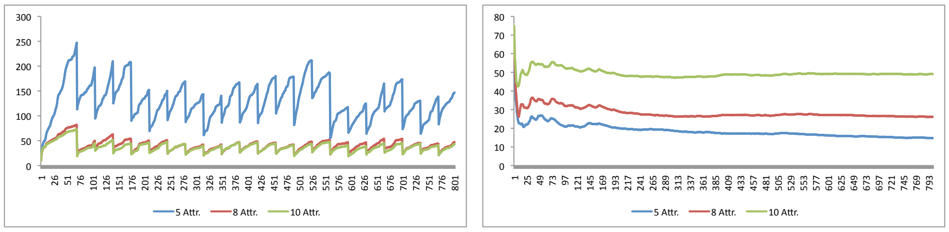
(a) (b)

Figure 4: (a) CPU usage. (b) Memory usage.

All the charts presented in this section were calculated in the same way. After processing each new tuple the following statistics were computed: the total CPU usage for mining all the graduals itemsets (Figure 4.(a)), the total number of nodes stored in the B-Trees (Figure 4.(b)), the average size of B-Trees (Figure 5.(a)) and the average support of five more supported itemsets (Figure 5.(b)). As we have obtained a very large log file, we have computed a smaller one computing the average of these values in groups of 100 elements. The group number is shown in the horizontal axis of all the charts, this gives us a time reference.

If we observe in detail the results presented in Figure 4.(a), we can see that the CPU time is constant over the time, then it is clear that our approach is able to work in real time scenario. The peaks are due to the B-Tree pruning process executed regularly following the tilted-time window frames, of course, as we have accelerated the stream rate this process is repeated more frequently. The highest peak in the 10-attribute scenario is produced at the beginning, just when the first pruning process is executed. This is produced because at the beginning all B-Trees are empty and most of new tuples are kept. Once B-Trees contain a higher number of nodes the insertion of a new tuple is more difficult and B-Tree growth is kept under control. We can see this fact in more detail in Figure 4.(b).

In Figure 4.(b) we show the amount of memory needed by our software. Of course, when the amount of attributes increases, the number of gradual itemsets and therefore the number of B-Trees also increase. This is the reason why the 10-attribute scenario uses more memory than the other two. As we can see, at the beginning of the stream the amount of memory needed increases very fast, but after the first pruning the amount of necessary memory is bounded. Obviously, it increases over the time but it is reduced in each pruning process. Therefore, we can say that our approach is able to work in scenarios where memory is a limited resource, as usual in most of



(a) (b)

Figure 5: (a) B-Tree average size. (b) Average support of five most supported itemsets.

the stream mining applications.

In Figure 5.(a) we present the average B-Tree size evolution over the time. As in the previous chart, at the beginning all B-Trees increase their size very fast, but after some pruning processes, the average B-Tree size is bouned. Again, the 10-attribute scenario has higher memory consumption than 5 and 8 attributes scenarios, this fact is related with the support values shown in Figure 5.(b). The higher the number of attributes, the easier the obtention of gradual itemsets having high support. If the support of the itemsets is high, the average B-Tree size will be also large.

Finally, in Figure 5.(b) we can observe the average support of five most supported gradual itemsets. At the beginning, the support is unstable since we have processed few tuples. However, when the number of tuples processed increases the support becomes stable. The support values provided in this last chart are consistent with the statement explained before: the more attributes we have, the higher the support of the most supported itemsets, because we are mining more gradual itemsets and the chances of finding gradual itemsets with high support increase.

It is important to note that the results obtained in the dynamic scenario do not vary in any kind of static scenario. This statement holds because the most supported rules in a static scenario are always the same and the time needed to process all the possible gradual rules is more or less constant. Of course, we have to prune the b-trees over the time, if not both, time and memory performances measures, will increase due to the large amount of tuples stored inside the b-trees.

With these results we can say that our approach is able to work with a real dataset in a real environment with strong resource constraints. We recall that we have used a workstation with a single 32 bits processor and only with one giga of main memory. Note that, nowadays most of servers or workstations have more than one CPU and a larger amount of available memory. In this section, we have shown that under these strong constrains we are able to mine in real time (10 new values per second) a high number of gradual itemsets, specifically, in the 10-attribute scenario we manage 29,512 gradual itemsets.

6.3 Static Gradual Rule Mining Comparison

In order to study the efficiency obtained by our new approach, we compare the results obtained in the previous section with some others obtained by the static algorithm defined in [17]. Such algorithm works as follows: Firstly, the dataset is sorted by all the a attributes in the dataset, then we obtain a different versions of the dataset. After that, the algorithm computes for each tuple a conflicting list, such list is a set of discarded tuples if the tuple is included in the list of tuples supporting a given gradual rule. Finally, tuples not supporting the gradual rule are discarded taking into account the size of the conflicting list, the larger the conflicting list, the earlier the tuple is discarded.
As such algorithm is defined only for static datasets, we have applied it over a bounded sliding window of 100 tuples. The CPU usage of this algorithm is depicted in Figure 6. If we observe the time results obtained for the scenario where we mine 10 attributes, we can see that the static version processes one tuple each 600 milliseconds, whist our new algorithm only spends 80 millieconds, one magnitude order less. These results show the improvement obtained by the application of the OWA operator, such application avoids the multiple sorting step needed in the static algorithm. Also using the Property 1 defined in the Section 5.1, we can omit the conflicting list computation step because we only need to check the previous and subsequent record in the B-tree to discard or maintain a tuple in the list of supporting tuples.
Unluckily, the support values obtained by both approaches are not so easy to compare. Whereas, the support in our approach is computed dividing the B-tree size supporting a gradual rule by the number of processed tuples, the support computation for the static algorithm has to be calculated by mixing the results of all the window processed at a certain period of time. This last computation is unfeasible for a very large number of tuples. For this reason, we have only compared the support values obtained in each window for both algorithms. The support values obtained by the static approach are slightly better, around 2.3%-8% better depending on the considered gradual rule. This fact can be easily explained: the static algorithm sorts the dataset

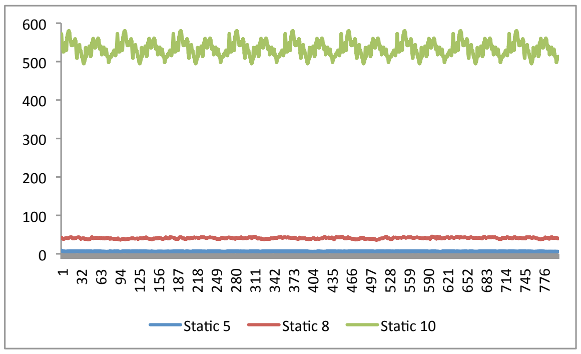


Figure 6: CPU usage of the static gradual rule mining algorithm

in the most convenient way for the support calculation, while the dynamic algorithm processes the tuples in the order they arrive at. However, we should say that the most supported rules are exactly the same in both cases. For this reason, we really believe that such difference is not significative considering the time improvement we obtain.

7 Conclusions

In this paper, we define an original method to mine data streams for gradual rules. Gradual rules are indeed of great interest to get knowledge from the databases managed by decision makers. However data streams, even if they become more and more present in the context of data management, are difficult to cope with as they are transmitted continuously at a potentially very high rate. Our approach is thus based on OWA operators and B-Trees so as to speed up the process. Our approach is shown to be efficient both in terms of time and memory consumption.
Future work include the possibility of selecting gradual rules. Moreover, we aim at further comparing the tree structures that may be used and their properties (either to speed up the insertion, or the initialization, etc.). Also, we would like to study the application of this algorithm to a very large static database because traditional methods cannot be applied in this case.

Acknowledgements

Partial support by the Spanish MEC (projects ARES – CONSOLIDER INGENIO 2010 CSD2007-00004 – and eAEGIS – TSI2007-65406-C03-02) is acknowledged. This work was also done in the context of the French ANR Project MIDAS (AND-07-MDCO-008).

References

[1] C. Aggarwal. *Data Streams: Models and Algorithms.* Springer, 2007.

[2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proc. of the 28th annual ACM Symposium on Theory of Computing (STOC'96)*, pages 20–29, 1996.

[3] F. Berzal, J. Cubero, D. Sanchez, M. Vila, and J. Serrano. An alternative approach to discover gradual dependencies. *Int. Journal of Uncertainty, Fuzziness and Knowledge-Based Systems (IJUFKS)*, 15(5):559–570, 2007.

[4] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[5] T. Calders, N. Dexters, and B. Goethals. Mining frequent itemsets in a stream. In *Proc. of the IEEE Int. Conference on Data Mining (ICDM'07)*, pages 83–92, 2007.

[6] T. Calders, N. Dexters, and B. Goethals. Mining frequent items in a stream using flexible windows. *Journal of Intelligent Data Analysis*, 12(3):293–304, 2008.

[7] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Sciences*, 312(1):3–15, 2004.

[8] Y. Chi, H. Wang, P. Yu, and R. Muntz. Moment: Maintaining closed frequent itemsets over a stream sliding window. In *Proc. of the IEEE Int. Conference on Data Mining (ICDM'04)*, 2004.

[9] F. Coenen, G. Goulbourne, and P. Leng. Tree structures for mining association rules. *Data Mining and Knowledge Discovery*, 8(1):25–51, 2004.

[10] G. Giannella, J. Han, J. Pei, X. Yan, and P. Yu. Mining frequent patterns in data streams at multiple time granularities. In *Next Generation Data Mining, MIT Press*, 2003.

[11] S. Greco, B. Matarazzo, N. Pappalardo, and R. Slowinski. Measuring expected effects of interventions based on decision rules. *Journal of Experimental and Theoretical Artificial Intelligence*, 17(1-2), 2006.

[12] M. Greenwa and S. Khanna. Space-efficient online computation of quantile summaries. In *Proc. of the ACM Int. Conference on Management of data (SIGMOD'01)*, pages 56–66, 2001.

[13] J. Han, Y. Chen, G. Dong, J. Pei, B. Wah, J. Wang, and Y. Cai. Stream cube: An architecture for multi-dimensional analysis of data streams. *Distributed and Parallel Databases*, 18(2):173–197, 2005.

[14] J. Han, J. Pei, B. Mortazavi-asl, Q. Chen, U. Dayal, and M. Hsu. Freespan: Frequent pattern-projected sequential pattern mining. In *Proc. of ACM Int. Conference on Knowledge Discovery and Data Mining (KDD'00)*, 2000.

[15] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation. *Data Mining and Knowledge Discovery*, 8:53–87, 2004.

[16] E. Hüllermeier. Association rules for expressing gradual dependences. In *Proc. of the 6th Eu. Conf. on Principles of Data Mining and Knowledge Discovery (PKDD'02)*, pages 200–211, 2002.

[17] L. D. Jorio, A. Laurent, and M. Teisseire. Fast extraction of gradual association rules: A heuristic based method. In *IEEE/ACM Int. Conference on Soft Computing as Transdisciplinary Science and Technology (CSTST)*, 2008.

[18] L. D. Jorio, A. Laurent, and M. Teisseire. Mining for gradualness over time using sequential patterns. In *Intelligent Decision Technologies (IDT'09)*, 2009.

[19] H.-F. Li, S. Lee, and M.-K. Shan. An efficient algorithm for mining frequent itemsets over the entire history of data streams. In *Proc. of 1st Int. Workshop on Knowledge Discovery in Data Streams*, 2004.

[20] G. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proc.of 28th Int. Conf. Very Large Databases*, 2002.

[21] F. Masseglia, P. Poncelet, M. Teisseire, and A. Marascu. Web usage mining: extracting unexpected periods from web logs. *Data Mining and Knowledge Discover*, 16(1):39–65, 2008.

[22] R. J. Miller and Y. Yang. Association rules over interval data. In *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 452–461, 1997.

[23] B. Nag, P. M. Deshpande, and D. J. DeWitt. Using a knowledge cache for interactive discovery of association rules. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 244–253, 1999.

[24] C. Raissi, P. Poncelet, and M. Teisseire. Speed : Mining maximal sequential patterns over data streams. In *Proc. of the 3rd IEEE Int. Conference On Intelligent Systems (IS2006)*, 2006.

[25] C. Raïssi, P. Poncelet, and M. Teisseire. Towards a new approach for mining maximal frequent itemsets over data stream. *Journal of Intelligent Information Systems (JIIS)*, 28(1):23–36, 2007.

[26] Z. Ras and A. Wieczorkowska. Action rules: how to increase profit of a company. In *Proceedings of the Principles of Data Mining and Knowledge Discovery (PKDD 00)*, pages 587–592, Lyon, France, 2000.

[27] A. Skowron and P. Synak. Planning based on reasoning about information changes. In *Rough Sets and Current Trends in Computing)*, pages 165–173, 2006.

[28] E. J. Stollnitz, T. D. Derose, and D. H. Salesin. Wavelets for computer graphics: theory and applications. In *Morgan Kaufmann Publishers Inc.*, 1996.

[29] W.-G. Teng, M.-S. Chen, and P. Yu. A regression-based temporal patterns mining schema for data streams. In *Proc. of 29th Int. Conf. Very Large Databases (VLDB'03)*, pages 93–104, 2003.

[30] V. Torra. The weighted owa operator. *Int. Journal of Intelligent Systems*, 12:153–166, 1997.

[31] V. Torra. Owa operators in data modeling and re-identification. *IEEE Trans. on Fuzzy Systems*, 12(5):652–660, 2004.

[32] V. Torra and Y. Narukawa. *Modeling decisions: information fusion and aggregation operators.* Springer, 2007.

[33] V. Torra and J. Nin. Record linkage for database integration using fuzzy integrals. *Int. Journal of Intelligent Systems*, 23(6):715–734, 2008.

[34] L.-S. Tsay and Z. Ras. Action rules discovery system dear, method and experiments. *Journal of Experimental and Theoretical Artificial Intelligence*, 17(1-2):119–128, 2005.

[35] A. Tzacheva and Z. Ras. Action rules mining. *International Journal of Intelligent Systems*, 20(7):719–736, 2005.

[36] K. Verma, O. Vyas, and R. Vyas. Temporal approach to association rule mining using t-tree and p-tree. In *Machine Learning and Data Mining in Pattern Recognition*, volume 3587 of *Lecture Notes in Computer Sciences*, pages 651–659. Springer, 2005.

[37] J. S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, 1985.

[38] R. R. Yager. On ordered weighted averaging aggregation operators in multi-criteria decision making. *IEEE Trans. on Systems, Man and Cybernetics*, 18:183–190, 1988.

[39] R. R. Yager. Families of owa operators. *Fuzzy Sets and Systems*, 59:125–148, 1993.