Speedy, Mini and Totally Fuzzy: Three Ways for Fuzzy Sequential Patterns Mining

Céline Fiot, Anne Laurent, Maguelonne Teisseire

Speedy, Mini and Totally Fuzzy: Three Ways for Fuzzy Sequential Patterns Mining

Céline Fiot, Anne Laurent, Maguelonne Teisseire
LIRMM - UMII, UMR CNRS 5506
161 rue Ada
34392 Montpellier Cedex 5, France
Email: {fiot, laurent, teisseire}@lirmm.fr

***Abstract*— Most real world databases are constituted from historical and numerical data such as sensors, scientific or even demographic data. In this context, algorithms extracting sequential patterns, which are well adapted to the temporal aspect of the data, do not allow processing numerical information. Therefore the data are pre-processed to be transformed into a binary representation which leads to a loss of information. Algorithms have been proposed to process numerical data using intervals and particularly fuzzy intervals. With regards to the search of sequential patterns based on fuzzy intervals, both existing methods are incomplete either in the processing of sequences or in the support computation. Therefore this paper proposes three methods to mine fuzzy sequential patterns (SPEEDYFUZZY, MINIFUZZY and TOTALLYFUZZY). We detail these algorithms by highlighting their different fuzzification levels. Finally we have assessed them through different experiments carried out on several datasets.**

***Index Terms*— Sequential patterns, numerical data, fuzzy intervals.**

I. INTRODUCTION

MOST real world databases are constituted from historical and numerical data (sensors, scientific, demographic data, ...). Within the context of large database mining, few works have been carried out to process this kind of data and most works are restricted to association rules [1]–[3]. A first proposal [3] processes quantitative data for association rules mining thanks to the attribute discretization into crisp intervals. However some frequent associations could be lost because of too restrictive bounds. Recently, the use of the fuzzy set theory has permitted less stark cuts between intervals leading thus to more relevant rules. [2] has presented a new definition of support and confidence based on fuzzy set theory, in the context of extracting association rules from quantitative attributes. Rules will go from "75% of people who buy butter also purchase bread" to the new type "60% of people who eat *a lot of* candies purchase *few* potato chips".

By contrast to association rule based approaches, sequential pattern algorithms take the temporal aspect of data (monitoring, evolution phenomena, ...) into account. They are thus well adapted to historical data. However they do not allow numerical data processing. Indeed such data must be pre-processed into a binary representation, which necessarily leads to a loss of information.
In this paper we present a complete and efficient fuzzy approach for sequential pattern mining which enables the processing of numerical data. Our approach is based on the definition of intervals and more precisely on the definition of *fuzzy* intervals. Obtained patterns are of the type "60% of people purchasing *a lot of* candies and *few* video games buy later *a lot of* toothpaste". These patterns are characterized by their support, which is by definition the percentage of clients who have bought these products. We define three approaches SPEEDYFUZZY, MINIFUZZY and TOTALLYFUZZY that differ one from each other by their definition of the support. The end-user is allowed to choose between speed of results extraction and accuracy of obtained frequent patterns. The implementation of the different solutions is based on an efficient and original data scanning, which extends the PSP algorithm proposed by F. Masseglia and al [4]. Experiments have been carried out on synthetic datasets. They highlight the feasibility of a fuzzy approach and its robustness.
This paper is structured as follows: Section II presents an introduction to sequential patterns and describes the lacks in the former proposals of fuzzy sequential patterns. Section III presents our three algorithms (SPEEDYFUZZY, MINIFUZZY and TOTALLYFUZZY) detailing the different support definitions and their involvement on the database scanning. TOTALLYFUZZY, the most complex algorithm is then detailed and illustrated through a short example. Section IV presents the implementation of the algorithms and the experiments. Section V concludes on the different perspectives associated to this work.

II. FROM SEQUENTIAL PATTERNS TO FUZZY SEQUENTIAL PATTERNS

In this section we briefly describe the basic concepts of sequential patterns then we take a look at the earlier proposals of fuzzy sequential patterns, highlighting their weaknesses.

A. Sequential Patterns

Let DB be a set of customers transactions where each transaction T consists of three informations: a customer-id, a transaction timestamp and a set of items in the transaction.
Let $I = \{i_1, i_2, ..., i_m\}$ be a set of items. An *itemset* is a non-empty set of itemsets, denoted by $(i_1 i_2 \dots i_k)$. It is a non-ordered representation. A *sequence* s is a non-empty ordered list of items, denoted $< s_1 s_2 ... s_p >$. A *n-sequence* is

a sequence of n items (or of size n).

Example 1: Let us consider purchases of products $1, 2, 3, 4,$ and 5 made by the customer Smith according to the sequence $s =< (1)\ (2\ 3)\ (4)\ (5) >$. It means that all items of the sequence were bought separately except the products 2 and 3 which were purchased at the same time. In this example, s is a 5-sequence.
One sequence $< s_1\ s_2 ... s_p >$ is a *subsequence* of another one $<\ s'_1\ s'_2\ ...s'_m >$ if there exist integers $l_1 < l_2 < ...l_j... < l_p$ such that $s_1\ \subseteq\ s'_{l_1},\ s_2\ \subseteq\ s'_{l_2}, ...,$
$s_p\ \subseteq\ s'_{l_p}$.

Example 2: The sequence s' = <(2) (5)> is a subsequence of s because (2) $\subseteq$ (2 3) and (5) $\subseteq$ (5). However, <(2) (3)> is not a subsequence of s since items were not bought together. All transactions from the same customer are grouped together and sorted in increasing order of their timestamp. They are called a data sequence. A customer *supports* a sequence s if it is included into the data sequence of this customer (s is a subsequence of the data sequence). The *support* of a sequence is defined as the percentage of customers supporting s. In order to decide whether a sequence is frequent or not, a minimum support value ($minSupp$) is specified by the user and the sequence is said to be frequent if the condition $supp(s) \geq minSupp$ holds. Given a database of customers transactions the problem of sequential patterns mining is to find all maximal sequences of which the support is greater than a specified threshold (minimum support) [5]. Each of these sequences represents a sequential pattern, also called a maximal frequent sequence. Note that items are processed using a simple binary evaluation: present or not present.

B. Fuzzy Sequential Patterns

In order to mine fuzzy sequential patterns, the quantity universe of each item is partitionned into several fuzzy sets. Then those fuzzy sets are used to mine frequent sequences.
T.-P. Hong and al [6] have presented the first proposal of a fuzzy sequential patterns mining approach. Their proposal is based on the discretization of data into fuzzy intervals. However to minimize the number of items being processed, they only keep, for each of them, the fuzzy set having the highest cardinal over the whole database (by Σ-count).

Customer	Date	$little$	$medium$	lot
C1	d1	0.6	0.4	0
C1	d2	0	0.7	0.3
C1	d3	1	0	0
Max(C1)		1	0.7	0.3
C2	d1	0	0	0
C2	d2	0	0.5	0.5
C2	d3	0	0.3	0.7
Max(C2)		0	0.5	0.7
C3	d1	0	0.6	0.4
C3	d2	0.9	0	0
C3	d3	0	0.2	0.8
Max(C3)		0.9	0.6	0.8
Count	$(\Sigma(Max))$	**1.9**	**1.8**	**1.8**

Fig. 1. Transactions for the fuzzy partition $Candy$

Example 3: Let us consider the database shown on Fig.1 describing the purchases of 3 customers for the attribute $Candy$.
In this case, the only column that will be taken into account in [6] for the item $Candy$ is $little$, whereas it is not the quantity that is the most present into the data sequences (it is in fact $medium$ or lot).
It is thus obvious that such a decision is reducing and can lead to erroneous knowledge. The consequences of this strategy on the discovery of sequential patterns will be pointed out in our experiments, section IV-D.
Y.-C. Hu and al [7], [8] have adopted a very theoretical approach without any algorithm or implementation. Moreover their proposal presents ambiguous formalism and notations for the support computation of a fuzzy itemset and so far a fuzzy sequence. It is notably hard to identify differences between the support computation of the sequences <(a)(b)(c)> and <(a)(b c)>. This point is however fundamental in the context of sequential patterns mining because the timestamps associated to the items play an important part. In conclusion previous works propose neither a complete approach nor clearly defined algorithms. Thus we claim that a new useful and efficient approach has to be defined: we propose three algorithms to mine fuzzy sequential patterns.

III. Proposal: from Low Fuzzy to High Fuzzy

In order to offset lacks of existing methods for extracting fuzzy sequential patterns, we firstly give a clear and non ambiguous definition of associated concepts (item, itemset, g-k-sequence and fuzzy support). The core of this paper is the proposal of three algorithms according to the adopted counting method.
We consider the four classical ways to compute fuzzy cardinality:

1) counting all elements for which the membership degree is not null;
2) considering only the elements for which the membership degree is greater than a defined threshold. This counting method is called *thresholded count*;
3) adding the membership degrees of each element. This counting method is called *sigma-count*;
4) adding the membership degrees greater than a defined threshold, this is called *thresholded sigma-count*.

Each algorithm corresponds to one precise support definition, thus allowing three levels of fuzzification during the mining of fuzzy sequential patterns.

A. Preamble: item, itemset, g-k-sequence

The concepts of item and itemset have been redefined compared to classical sequential patterns.

Definition 1: A **fuzzy item** is the association of one item and one corresponding fuzzy set. It is denoted by $[x, a]$ where x is the item (also called attribute) and a is the associated fuzzy set.

Example 4: $[candy, lot]$ is a fuzzy item where lot is a fuzzy set defined by a membership function on the quantity universe of the possible purchases of the item $candy$.

Definition 2: A **fuzzy itemset** is a set of fuzzy items. It can be denoted as a pair of sets (set of items, set of fuzzy sets associated to each item) or as a list of fuzzy items.

We use the following notation: (X, A) where X is a set of items and A a set of corresponding fuzzy sets.

Example 5: $(X, A) = ([candy, lot][soda, little])$ is a fuzzy itemset and can be also denoted by $((candy, soda)(lot, little))$.

One fuzzy itemset only contains one fuzzy item related to one same attribute. For example, the fuzzy itemset $([candy, lot][candy, little])$ is not a valid fuzzy itemset, because it contains twice the attribute $candy$.
Lastly we define a g-k-sequence.

Definition 3: A **g-k-sequence** $S =< s_1 \cdots s_g >$ is a sequence constituted by g fuzzy itemsets $s = (X, A)$ grouping together k fuzzy items $[x, a]$.

Example 6: The sequence $S =< ([soda, lot][candy, lot])$ $([video\ games, little]) >$ groups together 3 fuzzy items into 2 itemsets. It is a fuzzy 2-3-sequence.

In the next sections of this article we use the following notations: let $\mathcal{C}$ represent the set of customers and $\mathcal{T}_c$ the set of transactions for one customer c. Let $\mathcal{I}$ be the set of attributes and $t[i]$ the value of attribute i in transaction t.
Each attribute i is divided into fuzzy sets. For example, we use the dataset described on Fig.2 (an empty cell indicates that the product has not been purchased).

Customers	Date	Items: candy	toothpaste	soda	ball	videogame
C1	d1	2				
	d2	1	3	1		
	d3	4		1		
	d4			1	5	
	d5			2		2
C2	d1	2			1	
	d2			2		
	d3		4	1		
	d4	3				
C3	d1					3
	d2	3	1			
	d3				4	5
	d4			2		
	d5		2			
C4	d1					
	d2					
	d3	2				4
	d4			3		
	d5		2			
	d6			2		

Fig. 2. Transactions grouped by customers and ordered by their timestamp

First the quantitative database is converted into a membership degree database. Each attribute is partitionned into several fuzzy sets, as shown on Fig.3 which represents the membership functions for each attribute. These partitions are automatically built by dividing the universe of quantities into intervals. Each interval groups the same proportion of customers. It is then fuzzified in order to ensure a better generalization.

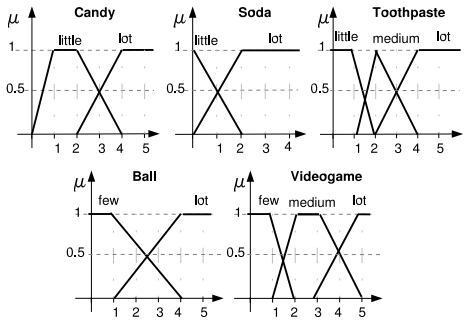

Fig. 3. Membership functions for the fuzzy sets of the database attributes

From these membership functions we get the membership degrees for each transaction and each fuzzy set. Fig.4 describes these values for customer 1.

D.	Items: candy li.	candy L.	toothpaste li.	toothpaste m	toothpaste L.	soda li.	soda L.	ball f.	ball L.	videogame f.	videogame m.	videogame L.
d1	0.75	0.25										
d2	1			0.5	0.5	0.5	0.5					
d3	0.25	**0.75**				0.5	0.5					
d4						0.5	**0.5**		1			
d5							1				1	

Fig. 4. Membership degrees for customer 1

Fig.4 is used below to illustrate the computation of the different supports for the sequence <([candy, lot])([soda, lot])>.

B. Fuzzy Supports

The **support of a fuzzy itemset** is computed as the percentage value of customers supporting this fuzzy itemset compared to the total number of customers in the database:

$$FSupp_{(X,A)} = \frac{\sum_{c \in \mathcal{C}} \Big[S(c, (X, A)) \Big]}{|\mathcal{C}|} \quad (1)$$

where the support degree $S(c, (X, A))$ marks whether the customer c supports the fuzzy itemset (X, A) or not.
As previously presented, the cardinality of a fuzzy set depends on the counting method. We transpose here three of those technics in the framework of fuzzy sequential patterns and we propose three definitions for the fuzzy support:

• SPEEDYFUZZY is based on the count "supports / does not support" (first counting method). Computing the support of a fuzzy itemset consists in counting all customers who purchased at least once the itemset. Whatever the membership degree of the purchase for the fuzzy itemset is, if it is greater than zero, the customer has the same weight:

$$S_{SF}(c, (X, A)) = \begin{cases} 1 & \text{if } \exists t \in \mathcal{T}_c | \forall [x, a] \in (X, A),\ \mu_a\big(t[x]\big) > 0 \\ 0 & \text{else} \end{cases} \quad (2)$$

Example 7: With the SPEEDYFUZZYcount, the first customer supports the sequence since one transaction is found containing the regarded itemset with a membership degree greater

than zero. It corresponds to the underlined fuzzy items in the Fig.4.

• MINIFUZZY is based on a thresholded count (second counting method). In this method, the number of customers supporting the fuzzy itemset is only incremented when each item of the candidate sequence has a membership degree greater than a specified threshold in the data sequence of the customer:

$$S_{MF}(c,(X,A)) = \begin{cases} 1 & \text{if } \exists t \in \mathcal{T}_c | \forall [x,a] \in (X,A),\ \mu_a\big(t[x]\big) > \omega \\ 0 & \text{else} \end{cases} \tag{3}$$

Example 8: With the second count, MINIFUZZY, we consider that customer 1 supports the sequence since a succession of transactions is found containing the items with a membership degree greater than the threshold (ω=0.49). These items are boldfaced in the Fig.4.

• TOTALLYFUZZY carries out a thresholded Σ-count (fourth counting method). In this approach the importance of each fuzzy itemset in the data sequence is taken into account in the support computation. To do so the threshold membership function α is defined as:

$$\alpha_a\big(t[x]\big) = \begin{cases} \mu_a\big(t[x]\big) & \text{if } \mu_a\big(t[x]\big) > \omega \\ 0 & \text{else} \end{cases} \tag{4}$$

The support counting formula becomes:

$$S_{TF}(c,(X,A)) = \underline{\perp}_{j=1}^{\theta_c} \overline{\top}_{[x,a]\in(X,A)} \Big[\alpha_a\big(t_j[x]\big)\Big] \tag{5}$$

where $\overline{\top}$ and $\underline{\perp}$ are the generalized t-norm and t-conorm opertators.
The Σ-count (third counting method) is actually a thresholded Σ-count, with a threshold ω=0, so it is not worse a particular case.

Example 9: With TOTALLYFUZZY, customer 1 supports the sequence if a following of transactions is found containing the fuzzy items of the sequence, with a membership degree greater than the threshold ω. The best value for the sequence is kept. In our example, these items are underlined twice in the Fig.4.
The **support of a fuzzy g-k-sequence** is computed as the percentage value of customers supporting this fuzzy sequence compared to the total number of customers in the database:

$$FSupp_{(X,A)} = \frac{\sum_{c\in\mathcal{C}} \Big[S(c,gS)\Big]\Big]}{|\mathcal{C}|} \tag{6}$$

where the support degree $S(c,gS)$ indicates if the customer c supports the fuzzy sequence gS. This support degree is computed using the algorithms *CalcSpeedySeq, CalcMiniSeq or CalcTotallySeq* described in subsections III-C and III-D.

C. SpeedyFuzzy and MiniFuzzy Algorithms

The way the algorithms SPEEDYFUZZY and MINIFUZZY work is quite similar. For each customer it is a matter of scanning the transactions set to find the candidate sequence. For each itemset in the sequence it is necessary to check whether the membership degree is not zero for SPEEDYFUZZY or greater than the threshold ω for MINIFUZZY. As soon as the candidate sequence has been validated (the ordered set of items is supported by the customer), the scanning of the customer's transactions is stopped and the support of the sequence is incremented.

The algorithm SPEEDYFUZZY (resp. MINIFUZZY) is based on two functions: *CalcSpeedySupp* (resp. $CalcMiniSupp$) computes the support while $FindSpeedySeq$ (resp. $FindMiniSeq$) searches for a candidate sequence in the transaction set of a customer.

D. TotallyFuzzy

The TOTALLYFUZZY algorithm is more complex than the two previous ones because it is based on a thresholded Σ-count. Thus for each customer and each sequence, the best membership degree must be considered. This degree is computed as the aggregation of the itemset supports. The order of the fuzzy items must also be taken into account. That leads to an exhaustive scanning of the transaction set, as performed for association rule mining. We present here an efficient implementation of such a scanning through the notion of *path*. One path corresponds to a possible instantiation of the candidate sequence itemsets into the customer's transaction set. Several paths may be initialized for one customer. For the global support computation we only keep the complete one having the best degree.

As an illustration we run TOTALLYFUZZY to compute the support of the candidate sequence *g-S = <([candy, little])([soda, lot])>* for customer 1 from figure 4, with a threshold ω= 0.2. One path is a triplet containing the already found sequence, the following itemset and the membership degree for each found itemset.
To initialize the process, one first empty path *pth1 = (∅, ([candy, little]), 0)* is created corresponding to the already found sequence *seq*, the currently searched itemset *curIS* and the current membership degree *curDeg*. For transaction d1, *curIS = d1[1]*, the path *pth1* is so updated with *<([candy, little])>, ([soda, lot]), 0.75)*.
Then for transaction d2, a new path *pth2*, *pth2* ← *(<([candy,little])>, ([soda, lot]), 1)* is created because d2 contains the item *[candy, little]*, first itemset of the candidate sequence. *pth1* is updated because *[soda, lot]* is included into d2. *pth1* is closed because it contains all the elements of the *g-S* sequence. Transaction d3 is then checked. It contains *pth2.curIS*, the path *pth2* is so modified to *(<([candy, little])([soda, lot])>, ∅,0.75)*. This path is closed since it contains all the itemsets from gS. Nevertheless the scanning is optimized and only keeps, for two paths at the same step, the one with the best membership degree. Thus *pth1* is deleted

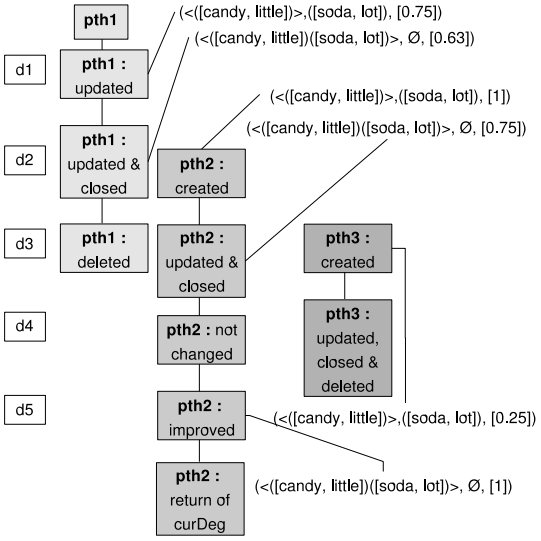

Fig. 5. Example of scanning for customer 1

of customer 1's list of paths because of a membership degree lower than *pth2.curDeg*. The scanning then continues as shown on Fig.5.

The algorithm TOTALLYFUZZY uses the function *FindTotallySeq* to carry out an ordered scanning in one customer's transaction set. When the first itemset of the sequence is found, one path is created with the itemset support. The next transactions are checked to find whether the following part of the sequence or once again the beginning of the sequence or an improvement of the paths, already created. All the possible paths are thus completed step-by-step at each transaction. The support degree of the best path for the whole sequence is then returned.

The $Update$ function allows the update of each path. The $Optimize$ algorithm, not presented in this paper, enables to delete unecessary paths. The function $CalcTotallySupport$ computes the support for one candidate sequence by adding for each customer the aggregation value of the optimal path for this sequence.

***CalcTotallySupport* - Input:** gS, candidate g-k-sequence ;
Ouput: $FSupp$ fuzzy support for the sequence gS ;
$FSupp, nbSupp, m \leftarrow 0$;
For each customer client $c \in \mathcal{C}$ **do**
  $m \leftarrow$ FindTotallySeq(g-S, $\mathcal{T}_c$);
  [the customer's support degree is aggregated to the current support]
  $nbSupp$ += m ;
end For
$FSupp \leftarrow nbSupp/\Gamma$;
return $FSupp$;

Fig. 7. CalcTotallySupport

***FindTotallySeq* - Input:** g-S, candidate g-k-sequence;
T, transaction set to run
Ouput: m, support degree of the best g-S representation instanciated in the transaction set T
$Paths$: **list of paths** → **(seq, curIS, curDeg)**
[seq is the subsequence of g-S, already found,
curIS is the following itemset in g-S,
curDeg is the list of membership degrees for the itemsets of seq]
$Paths \leftarrow$Path(Ø, gS.first, 0)
For each transaction $t \in T$ **do**
  For each path $pth \in Paths$, not updated at t **do**
    If (pth not closed) **then**
      If (pth.curIS $\in t$) **then**
        [It is considered that t contains the itemset if the degree of each item in the itemset is greater than the threshold ω]
        pth.curDeg $\leftarrow pth$.curDeg — $\overline{\top}_{[x,a]\in pth.\text{curIS}} \alpha_a(t[x])$;
        Update(pth);
      end If
    end If
    For j **from** 2 **to** pth.curIS -1 **do**
      [A possible improvement of the current path is searched for]
      If ((gS.get(j) $\in t$) & ($\overline{\top}_{[x,a]\in gS.get(j)} \alpha_a(t[x]) > pth$.curDeg[j])) **then**
        nCurIS $\leftarrow$ gS.get(j) ;
        For i **from** 1 **to** j-1 **do**
          nSeq $\leftarrow$ nSeq — gS.get(i) ;
          nCurDeg $\leftarrow$ nCurDeg — pth.curDeg[i];
        end For
        nCurDeg $\leftarrow$ nCurDeg — $\overline{\top}_{[x,a]\in gS.get(j)} \alpha_a(t[x])$;
        $Paths \leftarrow Paths \cup$ update((nSeq, nCurIS, nCurDeg)) ;
      end If
    end For
  end For
  If ((gS.first $\in t$) & (not(FirstPass))) **then**
    [a new path is created if the first itemset of the sequence is found]
    $pth \leftarrow$Path(Ø, gS.first, $\overline{\top}_{[x,a]\in gS.first} \alpha_a(t[x])$);
    $Paths \leftarrow Paths \cup pth$;
    Update(pth) ;
  end If
  Paths.Optimize() ; *[deletion of the less pertinent paths]*
end For
For each path $pth \in Paths$ **do**
  If (pth not closed) **then**
    Cut(pth); *[deletion of the paths not containing the whole sequence]*
  end If
end For
$pth \leftarrow Paths.first$; *[$Paths$ only contains the best complete path]*
$m \leftarrow \odot(pth$.curDeg) ; *[Aggregation to return the support degree]*
return m;

Fig. 6. FindTotallySeq

***Update* - Input:** pth, path to update

pth.seq $\leftarrow$ pth.seq $\cup pth$.curIS ;
If (pth.curIS.next $\neq$ Ø) **then**
  pth.curIS $\leftarrow$ pth.curIS.next ;
else
  Close(pth); *[Once the whole sequence is found the path is closed]*
end If
Optimize(pth) ;

Fig. 8. Update

Our approach extends the level-wise approach generate-prune within the context of sequential patterns and more particularly uses the prefix-tree structure described in [4], in order to improve the support computation process. For example the tree

on figure 9 represents the sequences *<([candy, little][soda, lot])([soda, lot])>*, *<([candy, little])([toothpaste, medium])>* and also the subsequences *<([toothpaste, medium])>* and *<([soda, lot])([soda, lot])>*.

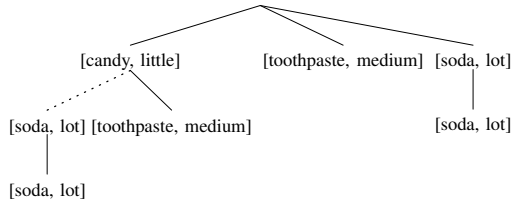

Fig. 9. Storage of sequences as a prefix-tree

Nevertheless, generating fuzzy candidates requires to check whether two fuzzy items in one fuzzy itemset are not related to the same attribute. Indeed in our approach it is not necessary to search for transactions containing both *little candy* and *a lot of candy*. Consequently such candidates are not generated.

First the fuzzy support of all fuzzy items is computed and only the items with a support greater than $minSupp$ are stored as frequent ones of size 1. Then candidates of size k are the g-k-sequences obtained by combining frequent sequences of size $k-1$. They are stored with their support into a tree. The fuzzy support of the g-k-sequences is then computed. The process stops when it is not possible anymore to generate candidate sequences of size $k+1$.

Example 10: As shown on Fig. 10, we obtain the following 1-frequent ones for TOTALLYFUZZY: *[candy, little]*, *[soda, lot]*, *[toothpaste, medium]* and *[video game, medium]*. These 1-frequent patterns enable us to generate candidates of size two such as *<([candy, little])([soda, lot])>* or *<([toothpaste, medium][video game, medium])>*.

candy		**toothpaste**			**soda**		**ball**		**videogame**		
li.	lot	li.	med.	lot	li.	lot	few	lot	few	med.	lot
75	50	25	**62.5**	37.5	25	**100**	25	50	0	**62.5**	37.5

Fig. 10. Fuzzy support $S_{TotallyFuzzy}$ (%) for each fuzzy item

Fig. 11 shows the sequential patterns (maximal frequent sequences) respectively extracted by SPEEDYFUZZY, MINIFUZZY and TOTALLYFUZZY.

Sequential patterns with SPEEDYFUZZY	<([video game, medium])>	75%
	<([candy, little])([toothpaste, medium])>	75%
	<([candy, lot])([toothpaste, medium])>	75%
	<([candy, little])([soda, lot])([soda, lot])>	75%
	<([candy, lot])([soda, lot])([soda, lot])>	75%
Sequential patterns with MINIFUZZY	<([video game, medium])>	75%
	<([candy, lot])>	75%
	<([candy, little])([toothpaste, medium])>	75%
	<([candy, little])([soda, lot])([soda, lot])>	75%
Sequential patterns with TOTALLYFUZZY	<([video game, medium])>	62.5%
	<([candy, little])([toothpaste, medium])>	56.3%
	<([candy, little])([soda, lot])([soda, lot])>	62.5%

Fig. 11. Extracted sequential patterns

It should be noted that the frequent items are the same for all counting methods. The difference stands in the number and length of the sequences. For a same $minSupp$, the number and length of the mined patterns are indeed greater with MINIFUZZY or SPEEDYFUZZY than with TOTALLYFUZZY. It is due to the thresholded Σ-count.
This reduction of the number of patterns can be used for a database containing really much frequent patterns to find the most pertinent ones. The advantage of this method is to be more selective and so to find the most relevant sequential patterns for the mined database. The user will thus be provided with a selection of patterns, and not only have to expertise a selection of patterns and not a really large quantity of them.

IV. EXPERIMENTS

In this section, we present a comparison of performances between the three algorithms SPEEDYFUZZY, MINIFUZZY and TOTALLYFUZZY and the algorithm PSP [4]. These experiments have been carried out on a PC - Linux 2.6.7 OS, CPU 3GHz and 512MB of memory. All the algorithms have been implemented in Java on the principle of PSP. In particular the Prefix-Tree structure is used to store the candidate and frequent sequences.

A. Datasets

We use synthetic datasets which have been created in several steps. In a first step, quantitative databases have been generated using an enhancement of DatGen [9]. Then an automatic partitionning is carried out by a tool based on a module of Weka [10] using the customer-timestamp-item-quantity file. This module, *DiscretizeFilter*, enables us to divide a given interval into several smaller intervals of equivalent number of elements (equi-depth). Lastly the membership degree database is created.

T	Number of transactions in the database
C	Number of customers in the database
I	Average number of items by transaction
X	Number of possible items in the database
Q	Maximum quantity value
P	Number of fuzzy sets by item

Fig. 12. Parameters for Synthetical Databases Generation

Fig. 12 gives the list of parameters used in the data generation method and Fig.13 shows the databases used and their properties. For these experiments the number of fuzzy sets by attribute has been fixed to 3, the maximal quantity by item to 10 and the number of different items in the database to 100. These databases are rather dense.

Name	C	T	I
cl5tr100I10	5,000	100,000	10
cl5tr200I10	5,000	200,000	10
cl5tr250I10	5,000	250,000	10
cl5tr350I10	5,000	350,000	10
cl5tr450I10	5,000	450,000	10
cl5tr500I10	5,000	500,000	10
cl10tr250I10	10,000	250,000	10
cl10tr500I10	10,000	500,000	10

Fig. 13. Parameter values for synthetic datasets in experiments

B. Operators

We have chosen to implement our version of TOTALLYFUZZY with *min* as a t-norm operator and *max* as a t-conorm operator.
Let consider an item. To be said frequent, this item must be relevant enough for enough customers (regarding the value of $minSupp$). It means that its degree must be greater than ω. Now let IS be an itemset. If we consider that each item of IS is relevant, we should be allowed to say that this itemset is relevant. The support of this itemset should thus not be lower than the minimum support from the items. The operator for computing this support must thus be idempotent. For this reason, we have chosen the min operator as t-norm. As aggregation operator we have chosen the average operator. Each itemset in the sequence will so be considered equivalent to the others. However we could have used the min operator to compute the support of sequences. A sequence is indeed the intersection, taking into account the temporal order, of itemsets. It can be computed using the min as explained above.

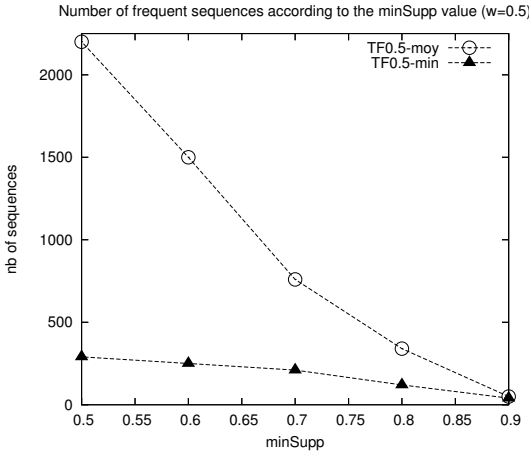

Fig. 14. Number of frequent sequences according to $minSupp$ depending on the aggregation operator for cl5tr100I10

Results described on Fig.14 show that the aggregation by the min is too selective and that it prunes too much sequential patterns.

C. Fuzzy Algorithms vs. PSP Algorithm

The aim of these experiments is to compare the fuzzy algorithms with PSP, a sequential pattern algorithm working with the same efficient structure as our fuzzy algorithms. Fig.15 shows the mining time according to $minSupp$. It can be noted that the extraction time increases as $minSupp$ decreases. The number of frequent patterns and generated candidates is indeed higher for a low $minSupp$. So the algorithms scan the dataset more times.

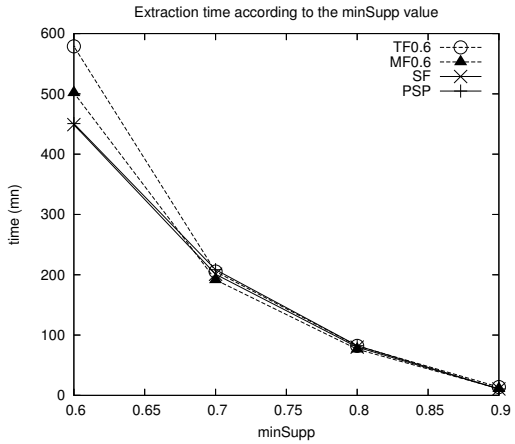

Fig. 15. Extraction time according to $minSupp$ (cl5tr100I10)

It can also be noted that SPEEDYFUZZY is as fast as PSP despite the fact that it scans is three times more items. One should indeed take into account that each PSP item has been splitted into three (it could have been more) fuzzy items to answer our algorithms needs. TOTALLYFUZZY and MINIFUZZY are slightly slower.

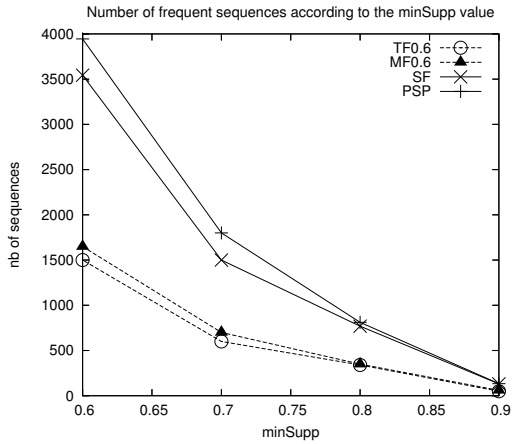

Fig. 16. Number of frequent sequences according to $minSupp$ (cl5tr100I10)

If we compare the number of frequent sequences for a same minimum support, Fig.16 shows that MINIFUZZY and TOTALLYFUZZY extract definitely less frequent sequences than SPEEDYFUZZY and PSP. It is due to the support definition.
In fact those two fuzzy algorithms, MINIFUZZY and TOTALLYFUZZY, only keep the items which have a degree greater than ω and so which are considered as relevant by the user. The number of frequent sequences is then necessarily reduced compared to SPEEDYFUZZY or PSP.
Finally Fig.17 shows the extraction time according to the number of transactions in a database describing transactions for 5000 customers, for $minSupp = 0.8$. It can be noted that the fuzzy algorithms have the same behavior as PSP.

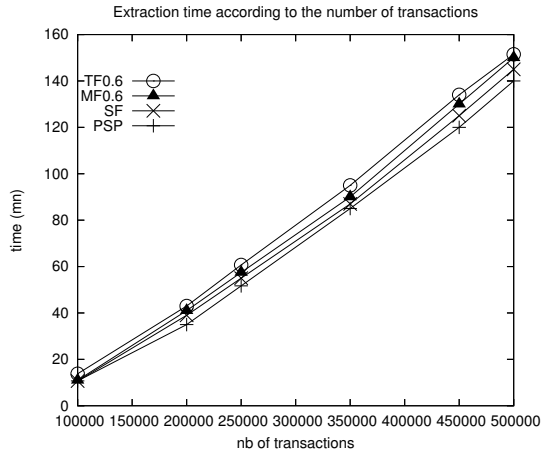

Fig. 17. Extraction time according to the number of transactions

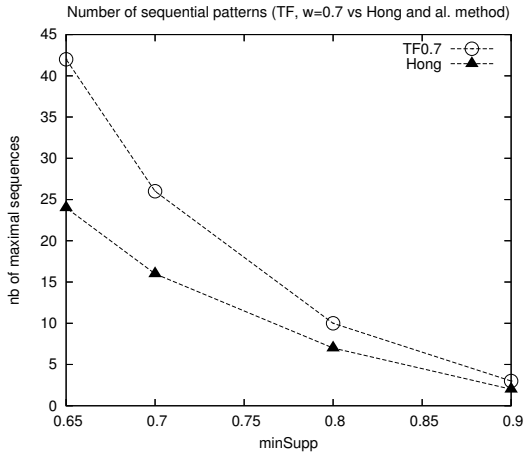

Fig. 19. Number of sequential patterns extracted by Totally Fuzzy (ω = 0.7) and by [6] for cl5tr100I10

D. Totally Fuzzy vs. Hong and al. approach

The aim of this section is to show the benefits of our method in comparison to Hong and al. method [6] presented in section II-B. In particular we study the loss of information resulting from their approach. So in our experiments we have compared our algorithm Totally Fuzzy (the closest from their approach) with an implementation of Hong and al. approach.

Frequent items	TOTALLYFUZZY(ω=0.7)	Hong and al.
9601	65.12%	65.12%
9701	74.32%	74.32%
9801	85.36%	85.36%
9803	**73.18%**	**not found**
9901	97.24%	97.24%
9902	**70.28%**	**not found**
9903	**91.46%**	**not found**
10001	95.94%	95.94%
10002	**66.76%**	**not found**
10003	**88.44%**	**not found**
Nb of frequent items	10	5

Fig. 18. Number of frequent items extracted by TOTALLYFUZZY (ω=0.7) and Hong and al. method, for $minsup$ = 60% on cl5tr100I10 database.

It can be noted that some items (e.g. *10003* and *9903*) with a very relevant support (resp. *88.44%* and *91.46%*) are found by TOTALLYFUZZY and not by Hong and al, they are in fact deleted during the first step as a peprocessing.
As a consequence, the number of maximal frequent sequences found by TOTALLYFUZZY is higher than by the Hong and al. method. The frequent sequences resulting from the five ignored frequent items are indeed never generated and so cannot be found.
Fig.19 shows this loss of information. In fact for $minSup$=70%, Hong and al. approach only finds two thirds of the sequential patterns mined by TOTALLYFUZZY.

V. CONCLUSION AND PERSPECTIVES

In this paper we present a complete and efficient fuzzy approach for sequential pattern mining which enables us to process historical numerical data such as demographic or sensors data. Extracting sequential patterns in such databases is highly interesting for event sequence detection. However existing algorithms do not allow numerical data processing. Our proposal clearly and completely defines the different concepts and principles associated to fuzzy sequential patterns. The algorithms, SPEEDYFUZZY, MINIFUZZY and TOTALLYFUZZY, have been implemented and tested, showing the interest of our novel approach and the feasibility for the fuzzy methods described. Thus the user can handle three fuzzification levels thanks to our three different algorithms. This choice allows the extraction of frequent sequences by making a trade-off between relevancy and performance. Experiments have highlighted that this work could be applied to different kinds of data and build many perspectives. More particularly we plan to extend this work for fuzzy generalized sequential patterns.

REFERENCES

[1] A. Fu, M. Wong, S. Sze, W. Wong, , and W. Yu, "Finding fuzzy sets for the mining of fuzzy association rules for numerical attributes," in *the First International Symposium on Intelligent Data Engineering and Learning (IDEAL)*, 1998, pp. 263–268.
[2] C. M. Kuok, A. W.-C. Fu, and M. H. Wong, "Mining fuzzy association rules in databases," *SIGMOD Record*, vol. 27, no. 1, pp. 41–46, 1998.
[3] R. Srikant and R. Agrawal, "Mining quantitative association rules in large relational tables," in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, Montreal, Quebec, Canada, 4–6 1996, pp. 1–12.
[4] F. Masseglia, F. Cathala, and P. Poncelet, "The PSP approach for mining sequential patterns," in *Principles of Data Mining and Knowledge Discovery*, 1998, pp. 176–184.
[5] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Eleventh International Conference on Data Engineering*. Taipei, Taiwan: IEEE Computer Society Press, 1995, pp. 3–14.
[6] T. Hong, K. Lin, and S. Wang, "Mining fuzzy sequential patterns from multiple-items transactions," in *Proceedings of the Joint 9th IFSA World Congress and 20th NAFIPS International Conference*, 2001, pp. 1317–1321.
[7] R.-S. Chen, G.-H. Tzeng, C.-C. Chen, and Y.-C. Hu, "Discovery of fuzzy sequential patterns for fuzzy partitions in quantitative attributes," in *ACS/IEEE International Conference on Computer Systems and Applications (AICCSA)*, 2001, pp. 144–150.
[8] Y.-C. Hu, R.-S. Chen, G.-H. Tzeng, and J.-H. Shieh, "A fuzzy data mining algorithm for finding sequential patterns," *International Journal of Uncertainty Fuzziness Knowledge-Based Systems*, vol. 11, no. 2, pp. 173–193, 2003.
[9] G. Melli, "Synthetic classification data set generator."

[10] I. H. Witten and E. Frank, “Data mining: Practical machine learning tools with java implementations,” 2000.