

# From Sequence Mining to Multidimensional Sequence Mining

Karine Zeitouni<sup>1</sup>

<sup>1</sup> PRISM Laboratory, University of Versailles-St-Quentin  
45 Avenue des Etats-Unis, 78035 Versailles, France

[Karine.Zeitouni@prism.uvsq.fr](mailto:Karine.Zeitouni@prism.uvsq.fr)

**Abstract.** Sequential pattern mining has been broadly studied and many algorithms have been proposed. The first part of this chapter proposes a new algorithm for mining frequent sequences. This algorithm processes only one scan of the database thanks to an indexed structure associated to a bit map representation. Thus, it allows a fast data access and a compact storage in main memory. Experiments have been conducted using real and synthetic datasets. The experimental results show the efficiency of our method compared to existing algorithms. Beyond mining plain sequences, taking into account multidimensional information associated to sequential data is for a great interest for many applications. In the second part, we propose a characterization based multidimensional sequential patterns mining. This method first groups sequences by similarity; then characterizes each cluster using multidimensional properties describing the sequences. The clusters are built around the frequent sequential patterns. Thus, the whole process results in rules characterizing sequential patterns using multidimensional information. This method has been experimented towards a survey on population daily activity and mobility in order to analyze the profile of the population having typical activity sequences. The extracted rules show our method effectiveness.

**Keywords:** Sequential data mining, data structures, algorithms, optimization.

## 1 Introduction

The problem of mining sequential patterns was first introduced in the context of market basket analysis [2]. It aims to retrieve frequent patterns in the sequences of products purchased by customers through time ordered transactions. Several algorithms have been proposed in order to improve the performances and to reduce required space in memory [20] [26] [11] [13] [10]. Other works have concerned mining frequent sequences in DNA [9] or Web Usage Mining [19] [22]. In general, it could apply to any database containing a collection of item sequences. Our target application was based on a time-use survey, more precisely the database reports the daily activities and displacements carried out by each surveyed person in a household.

After having tested the most cited algorithms, we have observed their weakness to scale with large size datasets. Indeed, most of them perform multiple scans of the

database, which is the main bottleneck in mining. Others require too large memory space to load the data when the database size increases.

This leads us to propose, in the first part of this chapter, a new algorithm which aims at enhancing the performances of mining sequential association rules while reducing resource consumption. We make the following contributions:

1. This algorithm only makes one scan of the database;
2. It is based on a highly compact main memory data structure, saving the required storage resources;
3. It allows a fast access to the data thanks to index structure;
4. The experimental results show that our algorithm outperforms existing ones.

Mining *sequential patterns* has many interesting applications as it is. In addition to performance issue, many works have proposed new features, such as incremental sequential pattern mining [5] [12], restriction by constraints [14] or dealing with new types of data, such as query plans [26]. Among interesting extensions, *multidimensional sequence mining* is a major issue [16]. In fact, it allows discovering rules that links between sequences (e.g. transaction history) and regular attributes data (such as those in client file). Such rules may describe customer profiles, e.g. to which category of individuals a given purchase (or a given path traversal pattern) corresponds, or discover to which category of individuals correspond a given path traversal pattern. This is the subject of the second part of this chapter.

Our approach consists in mining individual profiles - based on attributes - for the most frequent sequential patterns. At this end, we propose a characterization based approach where a whole sequence is considered as a complex attribute. Thus, it makes sense to integrate reasoning on sequences (frequent patterns, similarity, grouping) while other dimensions are considered as descriptive of each sequence group. Briefly, our approach is based on two steps. The first gathers all database sequences around the most similar *sequential pattern* in order to derive classes of sequences represented by their *sequential patterns*. The second step describes these classes (and their sequential patterns) by their *multidimensional* attributes values characterizing them.

The characteristic rules express which attribute properties are typical to frequent sequential patterns. The sequential patterns should fulfill a given support threshold, and the rule should be satisfied with a given confidence threshold. The extraction of such rules raises three main questions:

1. How to determine that a sequence or a subsequence is similar to another?
2. How to group multidimensional sequences with a given sequential pattern?
3. How to determine the most characteristic properties for a group of sequences?

We have adopted different solutions that we detail afterward.

Both methods have been experimented using a real dataset related to population daily activity and mobility survey. It aims at mining frequent patterns of activity sequences, then at analyzing the profile of the population having those typical activity sequences. In addition, other experiments have been conducted to test the scalability of the sequential pattern mining algorithm, and use synthetic data and public available data widely used.

This chapter combines and extends two previously published papers, namely [17] [18]. It is organized as follows: a background section will provide an overview of the

state of the art, before stating the concepts and definitions used further, and finally, it introduces the use case. Our contributions are then presented in two parts; the first is related to sequential pattern mining, then the second proposes an approach for multidimensional sequences mining. Each part details the implementation and the experimental results. The conclusion outlines the main contribution of the paper and discusses its perspectives.

## 2 Background

### 2.1 State of the art

Most works related to mining frequent sequences are in the field of customer transaction analysis. Early work on frequent patterns -*Apriori* algorithm- only considered transactions, not sequence of transactions [1]. This algorithm is costly because it carries out multiple scans of the database to determine frequent subsets of items. Three algorithms dealing with sequence of transactions are presented and compared in [2], and [20]: *AprioriAll*, *AprioriSome* and *DynamicSome*. *AprioriAll* algorithm is an adaptation of *Apriori* to sequences where candidate generation and support are computed differently. *AprioriAll*, and *AprioriSome* only compute maximal frequent sequences. Their principle is to jump to candidates of size  $k+next(k)$  in the next scan, where  $next(k)>1$ . Maximum frequent sequences of lower size that have not been calculated are given in the backward phase. The value of  $next(k)$  increases with  $P_k = |L_k|/|C_k|$ , where  $L_k$  stands for frequent sequences of size  $k$ , and  $C_k$  the whole generated candidates of size  $k$ . *DynamicSome* algorithm is based on *AprioriSome* but uses a jump by a multiple of user defined step. *SPAM* algorithm [10] uses a bitmap representation of transaction sequences once the entire database has been loaded in a lexicographic tree. But this algorithm considers that the entire database and all used data structures should completely fit into main memory, and then do not adapt for large datasets. *GSP* algorithm [20] utilizes the property that all contiguous subsequences of a frequent sequence also have to be frequent. As *Apriori*, it generates frequent sequences, then candidate sequences by adding one or more items. *PrefixSPAN* [14] first finds the frequent items after scanning the database once. The sequence database is then projected, according to the frequent items, into several smaller databases. Finally, all sequential patterns are found by recursively growing subsequence fragments in each projected database. Employing a divide-and-conquer strategy with the *PatternGrowth* methodology, *PrefixSPAN* efficiently mines the complete set of patterns.

As for *multidimensional sequences mining*, it has been studied recently and partially. Main works deal with frequent patterns that mix sequence items and dimensions. The main contribution is from [16]. They have defined three algorithms for mining *multidimensional sequential patterns*: *UniSeq* uses *PrefixSPAN* [14] to mine *multidimensional sequential patterns* with sequences extended with *dimensional information*. *Dim-Seq* uses the BUC-like algorithm [4] to first mine *multidimensional patterns*, then *PrefixSPAN* is used to mine the *sequential patterns* associated to the *multidimensional patterns*. *Seq-Dim* first mines *multidimensional patterns* using

BUC-like algorithm, then uses *PrefixSPAN* to mine the associated *sequential patterns*. Notice that the three algorithms produce the same result. However, they do not make a real distinction between *multidimensional values* and *sequential items*. Moreover, mining *multidimensional sequential patterns* do not allow extracting characteristic rules. Finally, they do not use specific methods to select the sequences while some multidimensional patterns are likely to be shared by groups of similar sequences.

## 2.2 Concepts and Definitions

In the proposed approach, we consider a database composed of sequences  $s$  and attributes  $A_i$  (with value  $a_i$ ) describing the sequences (table 1).

**Definition 1:** Let  $I$  be a set of items. A sequence  $s = \langle s_1, s_2, \dots, s_i, \dots, s_n \rangle$  is defined as an ordered list of elements  $s_i \in I$ .  $s_i$  is also called a sequence element.

More general definition was initially proposed in [2], where each sequence element is rather an item-set. However, we argue that basic sequences composed of single items are sufficient for many applications, and adopt this definition.

**Definition 2:** A *sequences database*  $S$  is composed of a set of tuples  $(sid, s)$  where  $sid$  denotes the identity of the sequence  $s$ .

**Definition 3:** A sequence  $s = \langle s_1, s_2, \dots, s_m \rangle$  is called a subsequence of another sequence  $t = \langle t_1, t_2, \dots, t_n \rangle$ , denoted  $s \subseteq t$  if and only if there exist integers  $1 \leq j_1 < j_2 < \dots < j_n \leq m$  such that  $s_1 = t_{j_1}, s_2 = t_{j_2}, \dots, s_n = t_{j_n}$ .

In other words, a sequence  $s$  is *included* in a sequence  $t$  if the ordered list of elements of  $s$  is included in the ordered list of elements of  $t$ .

**Definition 4:** The support of a sequence  $\alpha$  in a sequence database  $S$  is the number of tuples in the database containing  $\alpha$  i.e.,  $\text{support}(\alpha) = |\{ \langle sid, s \rangle \mid \langle sid, s \rangle \in S \wedge (\alpha \subseteq s) \}|$ .

**Definition 5:** Given a positive integer *min\_support*, a sequence  $\alpha$  is called a *sequential pattern* in sequence database  $S$  if  $\text{support}(\alpha) \geq \text{min\_support}$ .

Another concept is the similarity of sequences in order to compare them. Sequence similarity is a well known problem in the fields of bio-informatics. It aims at determining if a DNA sequence is similar or not to another. The most popular algorithms are BLAST [3], FASTA [15] and LCSS (*Longest Common Subsequence*) [6]. We use this last method in our definition of sequence similarity. It measures the minimum number of insertions and deletions to transform  $s_1$  into  $s_2$ . This method is widely implemented and used for this purpose.

**Definition 6:** Given two sequences  $s = \langle s_1, s_2, \dots, s_i, \dots, s_m \rangle$  and  $t = \langle t_1, t_2, \dots, t_j, \dots, t_n \rangle$  such that  $(i \in [1, m], j \in [1, n])$ . Let  $\text{lcs}(s, t)$  the size of the longest common subsequence. The *dissimilarity distance* between  $s$  and  $t$  is defined as:  $d(s, t) = m + n - 2 * \text{lcs}(s, t)$ .

**Definition 7:** Given a positive integer *DT* called *dissimilarity threshold*, sequence  $s$  is said similar to a sequence  $t$  if their dissimilarity distance is lower than *DT*.

**Definition 8:** A *multidimensional sequences database* is of schema  $(RID, S, A_1, \dots, A_m)$ , where *RID* is a primary key,  $A_1, \dots, A_m$  are dimensions, and  $S$  the domain of sequences. A *multidimensional sequence* is defined as  $(rid, s, a_1, \dots, a_m)$  where  $a_i \in A_i$ , for  $1 \leq i \leq m$  and  $s$  a sequence (see table 1).

**Definition 9:** We define a *Class of a Sequence*  $S$ , denoted  $S_C$ , as the cluster composed of multidimensional sequences whose sequences are similar to  $S$ , where  $S$  is a *sequential pattern*.

In order to define characteristic rules, we adopt and formalize the definition given in [9]. They define *characterization* of a sub-set as the property descriptions specific to this sub-set, comparing to all objects in the database.

**Definition 10:** We denote  $se$  a subset of the database  $DB$ ,  $prop$  a multidimensional property  $(a_{i1}, \dots, a_{ik})$ ,  $freq^{se}(prop)$  the number of objects in  $se$  that meet the property  $prop$ ; and  $card(se)$  the cardinality of  $se$ . The *significance* of  $prop$  in the subset  $se$  is defined as:  $F^{DB}_{se}(prop) = (freq^{se}(prop)/card(se)) / (freq^{DB}(prop)/card(DB))$

**Definition 11:** Given a real  $R$  standing for the significance threshold.  $prop$  is said characteristic of  $se$ , and denoted as:  $prop \rightarrow se$  [*significance*], if and only if:  $F^{DB}_{se}(prop) = significance \geq R$ .

**Definition 12:** Let  $S_c$  be the class of a *sequential pattern*  $SP_c$ . We define a *multidimensional sequential rule* as:  $prop \rightarrow SP_c$  [*significance*].

This *multidimensional sequential rule* means that the multidimensional property  $prop$  is characteristic of the *sequential pattern*  $SP_c$  with the computed *significance*. The example of table 1 shows a *multidimensional sequence database*. The tuple  $(1, \langle s_1, s_2, \dots, s_n \rangle, a_1, \dots, a_m)$  stands for a *multidimensional sequence* of the database.

**TABLE 1: A Multidimensional Sequence Database**

RID	S	A <sub>1</sub>	A <sub>2</sub>	...	A <sub>i</sub>	...	A <sub>m</sub>
1	$\langle s_1, s_2, \dots, s_i, \dots, s_n \rangle$	a <sub>1</sub>	a <sub>2</sub>		a <sub>i</sub>		a <sub>m</sub>
k	...						

### 2.3 Description of the Use Case and datasets

The target application is related to population time-use analysis and more precisely their daily activities and displacements. This dataset describes daily activities and displacements carried out by each person of a surveyed household at the scale of a whole urban area. It can be seen as a sequence of activities, also called *activity program* [24]. For example, during a day, an individual can leave home, drive children to school, go to work, pick children up from school and come back to home. This sequence can be described as (Home, School, Work, School, Home). In order to simplify the notations, we represent each activity by a specific character, e. g. H for Home, W for Work, and S for School. Other activities are Market (denoted M), Restaurant (R), Leisure (L), etc. This alphabet can be as long as necessary. Then, by removing the comma separators, a sequence could be simplified to a character string, e.g. HSWSH for the previous sequence. Although we have used activity programs as an example in our experiments, the analysis is also relevant for other sequences, such as the transport mode used for displacements, the departure time, and so on.

Activity programs of most individuals may be the same or be similar. Each activity program could be seen as a sequence of single value, making it possible to discover frequent activity sequences that characterise groups of the surveyed individuals. This allows analyzing the mobility of this urban population. Likewise, when considering transport mode, schedules or duration sequences, it would be possible to determine a typology of used transport modes, schedules, and so on. Besides, the survey holds

other information about individuals as their age, gender, profession, and so on. Then, each group having similar activity pattern is likely to have some characteristic attribute values. Hence, it is very relevant to characterize those groups and their corresponding sequential patterns. Those data have been used in the second part yielding characteristic rules for the groups of the surveyed individuals that share approximately those activity patterns. Moreover, other datasets have been used mainly to test the scalability and to compare our algorithms in the most widely used contexts, such as public datasets<sup>1</sup>.

### 3 A Sequential Pattern Mining Algorithm

This section proposes an algorithm of *sequential pattern* mining. We focus on the specific case where the considered sequences composed of single items instead of item-sets. We argue that this is the case if the most popular sequences, such as DNA [9], documents, web-logs, or activity program sequences. Our algorithm is compared to *PrefixSPAN*, and *SPAN*, ones of the most efficient mining algorithms.

#### 3.1 Algorithm Overview

The proposed algorithm is two phases. The first stage is the data encoding into a memory resident data structures. The second one is the frequent generation that in turn is composed of candidate generation, and candidate support checking. This algorithm only makes one scan of the database during which the total number of distinct sequences, the frequency of these sequences and the number of sequences by size are computed. This allows computing the support of each generated sequence.

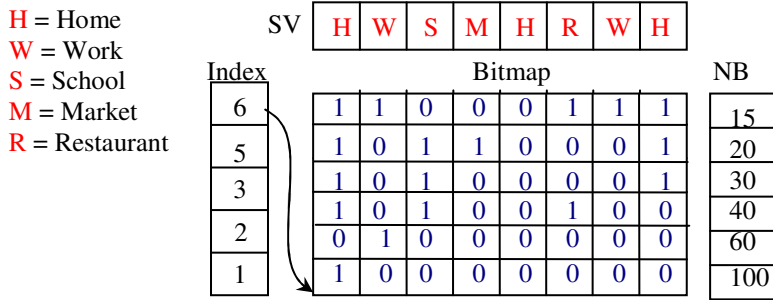
The backbone of our approach is its main memory data structure, called *IBM* as Indexed Bit Map. It is composed of four elements: (i) a Bitmap: a matrix that represents the distinct sequences of the database, (ii) *SV*: a sequence vector that encodes all the ordered combinations of sequences, (iii) *INDEX*: an index on the Bitmap that allows a direct access to sequences according to their size, (iv) *NB*: a table associated to the Bitmap which informs about the frequency of each distinct sequence (Fig. 1). Only distinct sequences are stored in the Bitmap, they are classified by decreasing size. An index by size allows a direct access to sequences according to their size, which optimizes the candidate generation and counting phase of the algorithm. In the example of Fig. 1, *IBM* encodes the whole distinct sequences of the database. Notice that *Index* and the *Bitmap* are numbered down-up. Here, there are 6 distinct sequences of size 1 to 5: H, W, HSR, HSH, HSMH and HSRWH. Each cell of the *INDEX* indicates the first line where the corresponding size of sequence is stored. For example, the cell number 5 (with value 6) corresponds to the line number 6 of the first sequence of size 5 encoded in the *Bitmap*. The table *NB* stores the frequency of

---

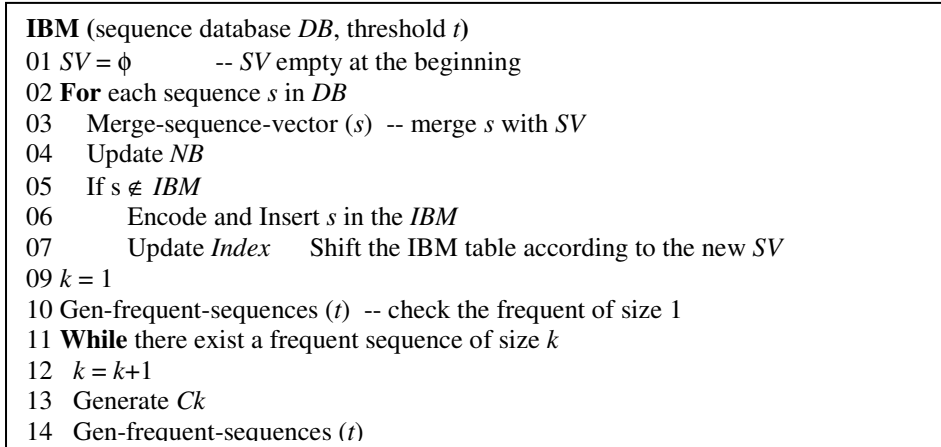
<sup>1</sup> <http://kdd.ics.uci.edu>

each distinct sequence in the database. Thus the sequence HSMH occurs 20 times in the database.

In the first stage of the algorithm, INDEX, SV, NB and the Bitmap are built on the fly during one pass. At each insertion of a sequence, the Bitmap matrix may become larger, and a set of shifting operations are applied to the bit values stored in this table.



**Fig. 1.** The data structure



**Fig. 2.** IBM algorithm

Fig. 2 shows the general IBM algorithm that takes as parameters: the sequence database *DB* and a threshold *t*. This value (*t*) stands for the minimum subsequences frequency taken into account for the generation of the candidates. Then for each sequence *s* that it reads from the database during the scan, *SV* (line 03) is generated using a merging process (as detailed in Fig. 3). Three situations may hold. The first is when the sequence is already encoded in *the Bitmap* which only requires the update of *NB* table (line 04): the line corresponding to this sequence in *NB* is incremented to maintain its frequency counting. The second case is when the sequence itself is not present in *the Bitmap*, but may be represented in *SV*. In this case, *s* will be encoded and inserted in *the Bitmap* as a new line, involving the update of *Index* and *NB* accordingly (lines 06 and 07). The last case is when the sequence cannot be

represented in *SV*, i.e. it is affected by the Merge-sequence-vector(*s*) function (Fig. 4). In this case, the algorithm adds the process of shifting the *Bitmap* table in order to adapt existing sequences coding to the new *SV* (line 08). Once all the data have been encoded in this structure, new candidates (line 10) are generated (see candidates generation section) and compared to the data stored in the *Bitmap* (line 11) with a fast access thanks to the *Index* (see support counting section).

### 3.2 Generation of the sequence vector

The sequence vector is generated during the unique scan of the database according to the algorithm of Fig. 3, which depicts the Merge-sequence-vector function which builds the *SV* vector according to the sequences present in the database.

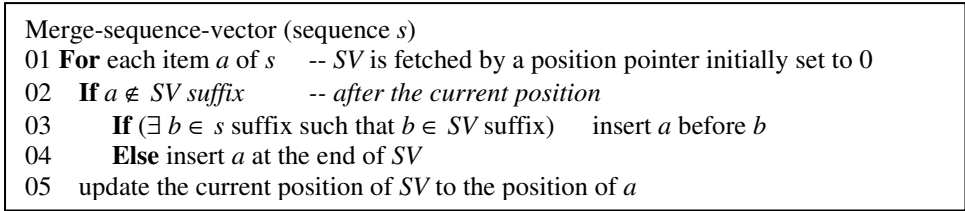


Fig. 3. Generation of the Sequence Vector

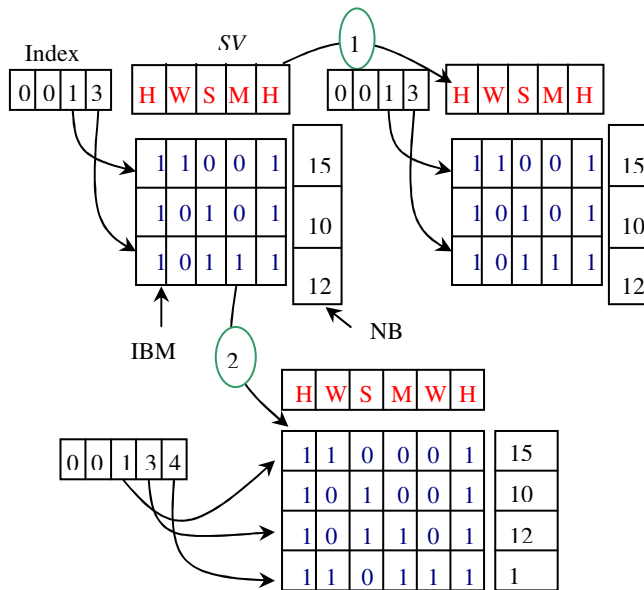


Fig. 4. IBM data structure generation process

It takes as parameter a sequence *s* of the database. It fetches the sequence items one by one checking their presence in *SV* in the right order (line 01). If a given item *a*

is not presents in *SV* after the current position (line 02), the function checks if their exist an item *b* in *SV* such that *b* in located after *a* in *s* and also located after the current position in *SV*. If those two conditions hold, item *a* will be inserted before item *b* in *SV* (line 03). If only the first condition holds, *a* will be positioned at the end of *SV* (line 04). The process will continue for the next item starting by the position of item *a* in *SV* (line 05). Notice that *SV* will stay unchanged when the sequence items belong to *SV* suffix. This holds when the current *SV* is sufficient to encode the new sequence. In the example of Fig. 4, the arrow with value 1 shows an insertion of a sequence HSH. This operation does not change *SV*. Since this sequence already exists in the Bitmap, only the corresponding frequency in the NB table is incremented. The arrow with value 2 shows an insertion of the sequence HWMWH. This operation modifies the *SV* vector; then, shifting operations are applied to the Bitmap in order to preserve the existing encoded sequences, and a new line is added in the Bitmap for the new sequence. Finally, the frequency of this new sequence is set to one.

### 3.3 Candidate generation and support counting

The second phase is based on the *APRIORI* principle, with the difference that it operates in a main memory data structure instead of scanning the database. Using the Index table in IBM, the process starts by the sequences candidates of size 1. Their support is counted (as explained later). They are retained as frequent if the support fulfils the support threshold. Then, bigger size candidates are generated from these frequent items using the fusion process (joining phase) as in the GSP algorithm [20]. The frequencies of the candidates are counted again, and the process is repeated.

The fusion process consists to merge two candidates having a common contiguous subsequence of size  $n-2$  in one sequence *s* of size *n*. For example, consider the two candidates  $c = \text{MMH}$  and  $c' = \text{MHM}$  of size 3. **MH** is a common contiguous subsequence of *c* and *c'*, and of size 2. Therefore, the candidate  $s = \text{MMHM}$  is generated from *c* and *c'*.

More formally, given two sequences  $c=c_1c_2\dots c_{n-1}$  and  $c'=c'_1c'_2\dots c'_{n-1}$  of size  $n-1$ , a sequence of size  $n$   $s=s_1s_2\dots s_n$  may be generated from *c* and *c'* as follows:

- (i) if  $c = c_1$  and  $c' = c'_1$ , then  $s = c_1c'_1$ .
- (ii) if  $n > 2$  and  $\forall i \in [2..n-1] c_i = c'_{i-1}$ , then  $s = c_1c_2\dots c_{n-1}c'_{n-1}$

As for the support counting of the generated candidates, it is facilitated by the data structure. For a given candidate *C* of size *S*, the algorithm (see Fig. 5) first looks in the cell number *S* of the Index where the first sequence of size *S* is encoded. Then, this line is accessed. For each line starting from this line to the last line of the Bitmap table, the algorithm determines using the *SV* vector if *C* is contained in each line of IBM. If so, the corresponding frequency of this sequence stored in the NB table, is added to the frequency of the candidate. After the comparison with each line until the last one, the support of *C* is computed.

Fig. 5 shows the Gen-frequent-sequences algorithm that determines all frequent candidates of size *k*. It takes as input a threshold *t*. All frequent candidates of size *k* are put in the set *Lk*. The function scans each generated candidate *Ck* of size *k* (line 02) and checks if it is included in the distinct sequences of size greater or equal than *k*

in *IBM* (line 03). *Index[k]* points to the first sequence of size *k* and *max* stands for the sequence of greatest size encoded in *IBM*. Then if a candidate is a subsequence of a given sequence encoded in *IBM* (line 05), his support is incremented by its frequency (*NB[k]*) in the database. At the end of this process, if the support of a given candidate of size *k* is greater then the threshold *t*, the candidate is placed into the set *Lk*.

Gen-frequent-sequences (Threshold <i>t</i> )	
01	$Lk = \emptyset$ -- Set of frequent sequences of size <i>k</i>
02	<b>For</b> all sequences $s \in Ck$ -- Candidates of size <i>k</i>
03	<b>For</b> all lines <i>l</i> in <i>IBM</i> from line <i>Index[k]</i> to <i>max</i>
04	<b>If</b> $s \subset l$ $s.count = s.count + NB[k]$ -- frequency of <i>s</i>
05	<b>If</b> $s.count \geq t$ $Lk = Lk \cup s$

**Fig. 5.** Generation of Frequent Sequences of Size *k*

Suppose the example of Fig. 1 and a generated candidate  $C=HSH$  of size  $S=3$ . Then the algorithm will access the cell number 3 of the Index which pin points to the line 3 of the *IBM* table, where the first sequence of size 3 starts. This sequence does not contain *C*, but those in line 4 to 6 contain *C*. So the frequency of *C* is computed as  $30+20+15=65$ . The support of *C* is equal to  $65 / (100+60+40+30+20+15)=0.245$ . If the support threshold is equal to 0.4, *C* candidate will not be retained as frequent pattern.

## 4 Implementation and Performance Study

The experiments aimed to validate our approach and to compare it to other methods. This comparison focuses on processing performances, storage costs, and scalability. The tests were performed on a 2.5 Ghz Pentium IV with 1 GB of memory running Microsoft Windows XP Professional. They aimed at showing: (i) our method effectiveness by applying it to a real dataset, and (ii) its scalability while increasing the data size.

Indeed, *IBM* has been performed on real data related to daily activity programs of the population of a north French urban area. In this application, the number of items is about 10. The database contains about 10,800 sequences among which 3,429 distinct sequences. The sequence size varies between 2 and 34 with an average size 6. The application aims at discovering frequent activity patterns in order to derive some population profiles. Interesting and previously unknown patterns have been produced which allow the decision makers better understanding of the daily activity and mobility for the reported population. For instance: (Home, Leisure, Home) is the most frequent with a support of 49%; (Home, Work, Home) is surprisingly less frequent with 37%, among which 9% correspond to (Home, Work, Home, Work, Home); (Home, School, Leisure, Home) appears in 11% of the sequences; while (Home, Shopping, Home, Leisure, Home) appears in 8% of them.

As for the scalability test, we generated and tested three different sizes of datasets with respectively: 100,000; 300,000; 600,000; and 1,000,000 rows. Items and the size of the sequences have been randomly generated for most experiments. The size of sequences was randomly generated from 2 to 60, and the number of distinct items was

about 10 (from 0 to 9). This number has been pushed to 20, 35 and 75 distinct items, notably by using the public dataset from UCI KDD archive. For our experimentations, we have used the packages PrefixSPAN-0.4.tar.gz<sup>2</sup> and Spam.1.3.1.tar.gz<sup>3</sup>. Moreover, we have augmented the activity dataset by randomly generating arbitrary sequences. Finally, we have used the public datasets provided in UCI KDD archive chess.dat<sup>4</sup> that has a larger alphabet than in the activity survey dataset. We have measured two features: the overall response time from one hand, and the storage cost from the other.

Moreover, we have studied two variants of the IBM algorithm at the implementation level. The first one, called IBM2, is based on the observation that the binary matrix manipulation necessitates shifting operations. In order to avoid these superfluous and costly computations, we have proposed the variant algorithm IBM2 where the Bitmap is replaced by a matrix of Boolean types. This type takes 8 bits in languages like Java or C++. This solution requires more space in memory, but it performs better since accessing each cell becomes direct. The gain in performances of IBM2 has been confirmed by experimental results below. The second variant, called *IBM\_OPT* (respectively *IBM2\_OPT* when combined with *IBM2*), uses the fact that the structure of IBM is independent from the support threshold value. The idea is to serialize it as a *Java object* and stored in a file, which makes it available for later use after a simple load in the main memory. Thus the cost of pre-processing can be totally avoided. This is particularly suitable to data mining process where the user interactively tries different support parameters before getting a satisfactory result. This optimization improves the performance in processing time, especially for large and very large databases as shown in the next section.

Fig. 6 shows all proposed algorithms, namely *IBM*, *IBM2*, *IBM\_Opt* and *IBM2\_Opt*, outperform *PrefixSPAN* in the most cases for the dataset having 100,000 sequences. *SPAM* has approximately the same response time and outperforms *IBM* when the support decreases, but *IBM* remains faster. The size of *SV* is about 173 items, the generation time for the structure is equal to 2 seconds and the number of distinct items in the dataset is around 17,000 sequences. This experiment also confirms that *IBM* outperforms *IBM2*, and shows the performance gain increases as the support decreases. Indeed, a lower support increases the number of generated candidates. The more the candidates are generated, the more the number of comparisons in the structure increases and the more the number of shifting operations increases. Finally, this experiment shows that *IBM\_OPT* (respectively *IBM2\_OPT*) outperform *IBM* (respectively *IBM2*). The performance gain (materialized by the gap between the corresponding curves) remains constant. Indeed, this difference corresponds to the saved time of the data structure construction and it is independent of the support threshold. For the dataset composed of 300,000 sequences, we observe the same behavior than in the previous experiment (Fig. 7). The size of *SV* is about 219 with 50,000 distinct sequences in the dataset and the structure is generated in 7.5 seconds. For a dataset composed of 600,000 sequences (Fig. 8), *SPAM* algorithm produced a memory overflow and failed. The size of *SV* is composed of 265 items with 90,000 distinct sequences in the dataset and the time to generate the structure is

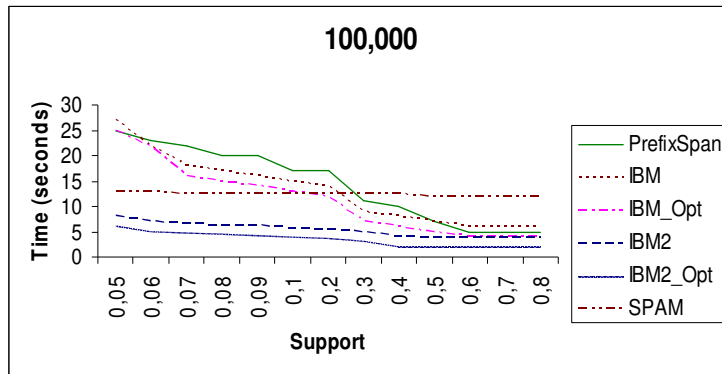
---

<sup>2</sup> <http://chasen.org/~taku/software/PrefixSPAN/>

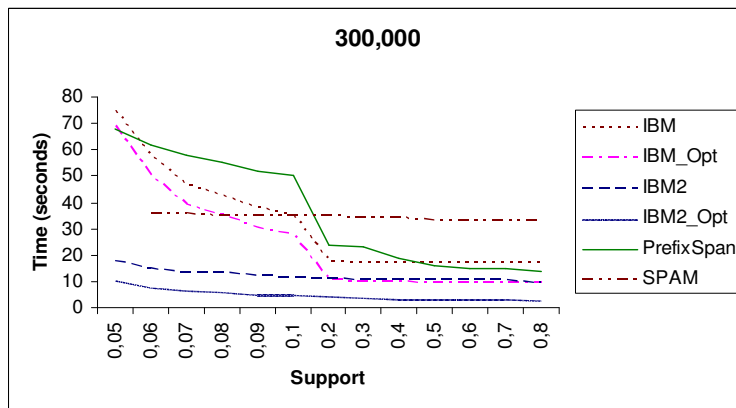
<sup>3</sup> <http://himalaya-tools.sourceforge.net/Spam/#download>

<sup>4</sup> <http://kdd.ics.uci.edu>

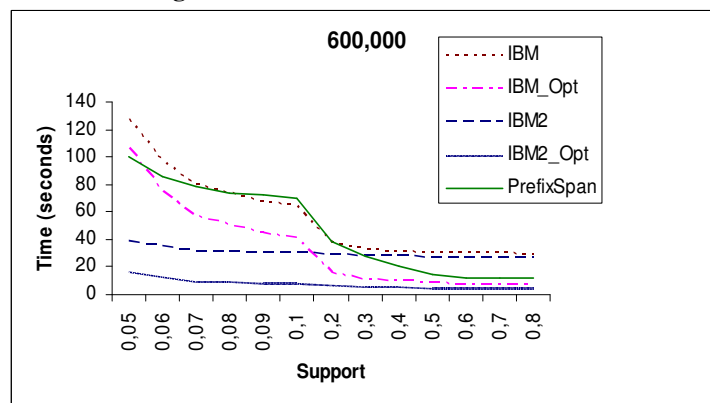
about 22.6 seconds. We observe that *PrefixSPAN* shows better performances than *IBM*. *IBM\_OPT* and *IBM2\_OPT* still outperform *PrefixSPAN*. *IBM2* remains better than *IBM* and than *PrefixSPAN*, especially for a support below 0.3.



**Fig. 6.** Performances with 100,000 rows



**Fig. 7.** Performances with 300,000 rows



**Fig. 8.** Performances with 600,000 rows

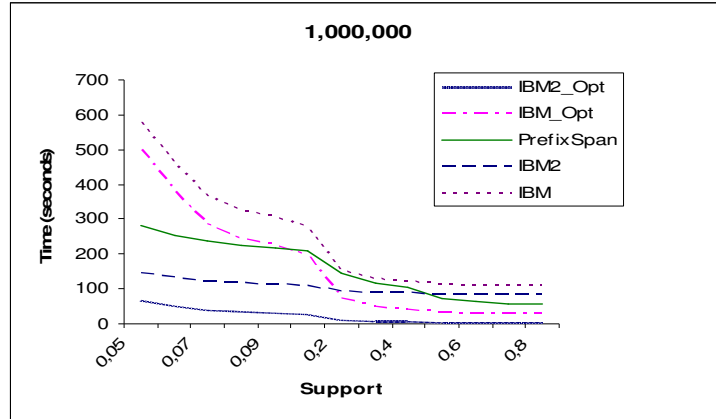


Fig. 9. Performances with 1,000,000 rows

For a dataset composed of 1,000,000 sequences, the size of  $SV$  is about 370 with 160,000 distinct sequences in the dataset and the time to generate the structure is about 80 seconds. *PrefixSPAN* consolidates its performance compared to *IBM*. *IBM\_OPT* always outperforms *PrefixSPAN*. In general, *IBM* shows better performances than *PrefixSPAN*, unless the support is high. Conversely, *IBM\_OPT* outperforms *PrefixSPAN* when the support is greater than 0.1. In order to measure the impact of the alphabet size on the performances, we have generated datasets using a larger alphabet (i.e. with more than 20 distinct items). Fig. 10 and 11 show the results for 20 and 35 distinct items for dataset composed of respectively 130,000 and 100,000 sequences. For 20 distinct items,  $SV$  is composed of 192 items with 26,000 distinct sequences in the dataset. For the dataset composed of 35 distinct items,  $SV$  is composed of 214 items with 32,000 distinct sequences. We observe that *PrefixSPAN* outperforms *IBM\_OPT* and *IBM*, but *IBM* and *IBM\_OPT* win *PrefixSPAN*. Until 35 distinct items, *IBM\_OPT* wins *PrefixSPAN*, with a support lower than 0.2. But, *PrefixSPAN* outperforms *IBM* due to the generation of the structure.

For more than 35 distinct items, *PrefixSPAN* becomes the faster, but this is true only for this large database size. Indeed, an interesting result has been obtained on a smaller database which has a larger alphabet (of 75 items).

This test (see Fig. 12) used the public dataset *chess.dat*, provided notably in the UCI KDD archive. The dataset is composed of 75 distinct items with 3,196 sequences of size between 36 to 37 items. The generation time of the structure is insignificant. Although the number of distinct items is greater than 35 (75 here), *IBM* and *IBM\_OPT* outperform *PrefixSPAN* (see Fig. 12). The main reason is the size of sequences, which is larger (from 36 to 37) compared to the synthetic datasets. Thus height of the tree representing the projected database in *PrefixSPAN* is greater than for the synthetic data. Therefore, traversing this projected data in *PrefixSPAN* is slower than scanning our *IBM* data structure. The memory space required by *IBM*, *IBM\_OPT* and *PrefixSPAN* are respectively equal to 2.3 MB, 0.29 MB and 9 MB. Nevertheless, the  $SV$  is only composed of 91 items. Concerning the storage cost, we have measured the memory resource consumption for each algorithm for each tested dataset.

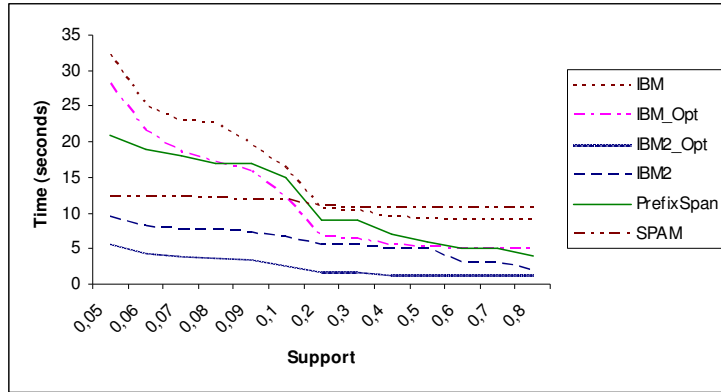


Fig.. 10. Performances for a dataset with 130,000 rows and 20 distinct items

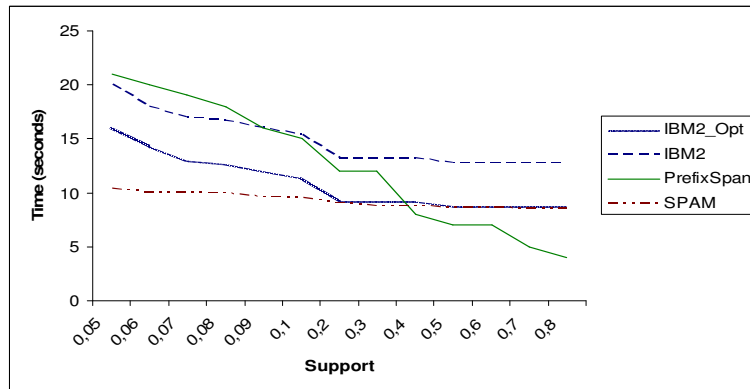


Fig.. 11. Performances for a dataset with 100,000 rows and 35 distinct items

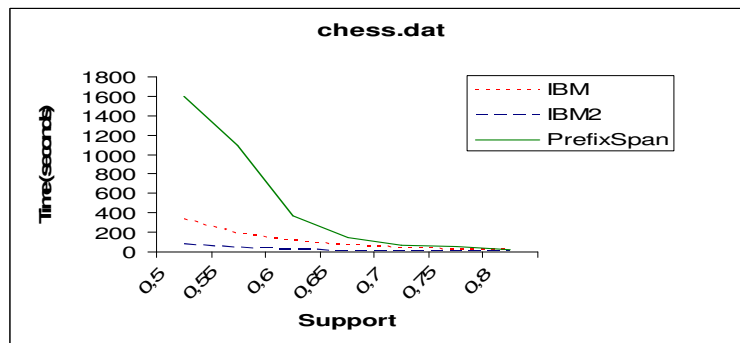


Fig.. 12. Performances with chess.dat file.

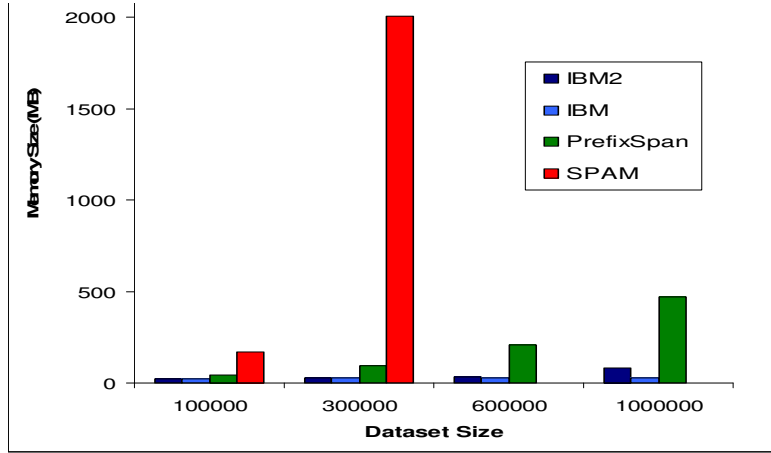


Fig. 13. Memory consumption

Fig. 13 shows the total memory consumption (in MB) used by *IBM* and *IBM2*, *SPAM*, and *PrefixSPAN*. For instance, with a database composed of 600,000 rows, *SV* contains about 265 values for 90,000 distinct rows. The size of the Bitmap in *IBM2* is then equal to:  $265 \times 90,000 = 23.85$  MB. As *IBM* is 8 times more compact, the size of the binary Bitmap is less than 3 MB. With 1,000,000 rows (Fig. 9), *SV* contains 370 elements for 160,000 distinct rows. Then, the size of the Bitmap reaches 59.2 MB in *IBM2*, whereas the size of the Bitmap fits in 7.5 MB in *IBM* algorithm. These results show that *IBM* is more appropriate than *IBM2* for very large databases, due to data compression. However, *IBM2* runs faster than *IBM*. This is due to the costs of shifting operations necessary to access target values, whereas *IBM2* directly accesses the target sequences. As we can see in Fig. 13, the difference between memory costs in *IBM* and in *IBM2* is insignificant compared to memory costs in *SPAM* and in *PrefixSPAN*. For example with 1,000,000 rows, the total memory size for *IBM* is equal to 28 MB whereas for *PrefixSPAN*, it is about 468 MB.

The size of the Bitmap also depends on the size of *SV*, which also increases with the number of distinct sequences. Notice that *SV* size does not depend on the size of the database itself. In fact, it only increases when the encountered sequence can not be encoded using the current *SV*. Moreover, since the probability to find common ordered items between *SV* and the current sequence becomes high as the building process advances, *SV* size becomes stable regardless of the size of the database.

In order to prove the efficiency of *IBM* in extreme cases, we have performed a test with a sequence  $c_T$  of the following form: HWHWHW...HW of size 200, composed of repeated series of H and W items. This type of sequence is likely to increase the size of the data structure. This experiment aims first to demonstrate that this situation does not affect the processing costs, and second to evaluate the loss of storage performance.

Tests have been done with datasets composed of 600,000 and 1,000,000 sequences. We observe no variation of processing costs. This is because, according to the sequence vector generation algorithm (see the general algorithm), the repeated items H and W that are not located in other sequences are put at the end of *SV*. Therefore,

unless repeated series are actually frequent in the database, the probability to have long repeated series of HW in the middle of SV is very low. Then, using the data structure for candidate generation and frequent patterns will not be affected, because the items H and W put at the end of the structure would never be accessed.

## 5 Characterization Based Multidimensional Sequence Mining

In order to extract relevant knowledge from a multidimensional sequence database, we propose a characterization based approach. This process combines: *sequential patterns mining*, sequence similarity, and characterization. It produces *sequential multidimensional rules* as defined in Background section.

### 5.1 Method Overview

Our approach is sequence centered. Indeed, it aims to characterize the most frequent sequences according to the other dimensions. It is divided into three steps:

1. first, *sequential patterns* are mined ;
2. then these subsequences are used to represent classes composed of subset of database sequences. In order to determine to which class a sequence belongs, an algorithm of similarity between sequences is used. The sequences of database which are the most similar to a given *sequential pattern* are gathered in the same class. At the end of this process, a set of classes composed of database sequences and represented by their *sequential pattern* are defined<sup>5</sup>. In order to reduce the intersections between classes, only *maximal sequential patterns* are considered i.e. *sequential patterns* that are not included in another one ;
3. once the classes have been built, a characterization algorithm is used to discover the dimension values characterizing them.

### 5.2 General Algorithm

Based on the three steps previously described, the general algorithm for mining multidimensional sequential patterns is depicted in Fig. 14. The input parameters of this algorithm are:

- A *multidimensional sequences* database DB.
  - A support FT for mining *sequential patterns*.
  - A *dissimilarity* threshold DT for mining the most similar sequence to a given *sequential pattern*.
  - A *characterization* threshold CT and a set of attributes and associated values  $E(A_i, V_j)$ .
-

The algorithm first mines *maximal sequential patterns* and places the result in D (line 03). Each *pattern* becomes a *model* of a class of sequences. Individuals whose sequences are the most similar to a given *maximal sequential pattern* are placed in the corresponding class  $S_c$  (line 05). The characterization algorithm (line 07) determines which attribute values characterize the subset  $S_c$  among  $E(A_i, V_j)$ .

```

Mining_MSP(DB, FT, DT, CT, E(Ai, Vj))
01 D = ∅
02 R = ∅
03 D = Maximal_Sequential_Patterns (DB, FT)
04 For each S ∈ D
05     Sc = Cluster (DB, S, DT)
06 For Each Sc
07     R = Characterization (Sc, DB, CS, E(Ai, Vj))

```

**Fig. 14.** General Algorithm

```

Characterization (class se, database DB, real S, set E of (attribute A, value V))
00 se.characterization = ∅ ;
01 compute freqDB(prop) for all properties prop = (attributes, values) ;
02 For each attribute A
03     For each value V of A
04         compute freqse(prop) for the property prop = (attribute, value) ;
05         If FDBse(prop) ≥ S     Add (se.characterization, prop) ;

```

**Fig. 15.** Characterization Algorithm

The first step performs *maximal sequential pattern mining*. As seen before, *IBM* algorithm shows the best performances. Therefore, we use this algorithm at this step of the process. Frequent patterns are likely to have much more similar sequences in the database than non frequent ones. Therefore, our approach of class generation starts from those frequent patterns to determine classes' centers. We call them *models*.

However, not all *sequential patterns* can be considered as *models*, because some are very similar to each others, and then do not maximize the dissimilarity between different classes. Especially, as subsequences of frequent patterns are all frequent, their similarity is high. Therefore, *models* are restricted to maximal frequent patterns. The second step assigns the sequences of the database to the classes that are represented by those *models*. This is performed based on similarity computation between the database sequences and those *models*. However, most algorithms rather compute the *dissimilarity distance*. Depending on the selected similarity algorithm, the more the *dissimilarity distance* is small, the more the candidates are similar i.e. have common ordered items. We adopt LCSS algorithm because the presence of each item in the sequence is here more relevant than its occurrence probability (see section 2). A *dissimilarity* threshold allows the user to adjust the *overlapping* between classes as well as the class coverage. The more this threshold will be high, the more the

*overlapping* probability will be high. Inversely, if this threshold is too small, too many database sequences may not be clustered.

The final step is to discover the main characteristics of individuals (profiles) characterizing each *class* i.e. the dimension values specific to a given *class*. In order to extract such knowledge, we propose an algorithm based on the definition of the characterization proposed by Han and Kamber [9]. The algorithm is detailed in Fig. 15 above.

### 5.3 Experimentation

This experimentation has been done using the real dataset provided by *household activity survey*. The dataset contains 10840 individuals, with the dimensions: gender, age (10 classes of age) and work types (about 9). The total number of distinct sequence activities is about 3429. The tests have been realized using the following threshold values: *support*=0.1 ; *dissimilarity*=1 ; *characterization*= 8.

The algorithm has found 17 classes. For instance, the sequential pattern HMRH has been mined with a support equal to 0.14. Its similar sequences are: HWMRH, HMHR, HMRH, HMMRH, HRH, HLMRH, HMH, and HMRRH. This set of sequences composes the class of HMRH. The following rules have been mined for this class:

- (gender = F)  $\wedge$  (30 < age < 40)  $\wedge$  (job category = TE8)  $\rightarrow$  HMRH [9.064].
- (gender = F)  $\wedge$  (50 < age < 60)  $\wedge$  (job category = TE7)  $\rightarrow$  HMRH [10.413].
- (gender = F)  $\wedge$  (50 < age < 60)  $\wedge$  (job category = TE5)  $\rightarrow$  HMRH [12.466].
- (gender = H)  $\wedge$  (60 < age < 70)  $\wedge$  (job category = TE6)  $\rightarrow$  HMRH [10.407].

The three first rules mean that individuals having an activity sequence similar to HMRH standing for (Home, Market, Restaurant, Home) are frequently (significance between 9.064 and 12.466) women between 30 and 40 years old working as liberal profession (here TE8), or women working as clerk (TE5) or without profession (TE7) between 50 and 60 years old.

## 6 Conclusion and Future Work

This chapter has proposed novel approaches for mining sequential patterns and multidimensional characterization. The first part has presented a new algorithm *IBM* and its variants *IBM2*, *IBM\_OPT* and *IBM2\_OPT*. The aim of this algorithm is to mine frequent sequences in item sequences. *IBM* only makes one scan of the database and provides efficient data structure that optimizes memory space as well as access costs. It has been applied to discover all frequent activity sequences in a time use survey database within an urban area. The experiments have shown that *IBM* and its variants provide better performances than existing algorithms in most cases: a better response time is reached, while a very low memory is required. Experimental results have also shown that *IBM2* and *IBM2\_OPT* outperforms *IBM* and *IBM\_OPT*, which in turns outperforms *SPAM* and *PrefixSPAN* for large and very large databases and for a limited number of distinct items.

Notice that the proposed data structure for *IBM* and *IBM2* algorithms, and especially the *SV* vector, could be used for other purposes as similarity search between sequences and sequence clustering. Another perspective is to apply it to different application contexts, as the analysis of query plans and genomic sequences. We have already experimented many various datasets: activity sequences, chess play sequences, and the web pages sequences *msnbc.dat* from the UCI KDD archive. In the context of the activity-mobility survey, we will explore the mining of spatial sequences, such as trajectories [7]. This is still a challenging research issue.

This chapter also describes a sequence centered approach for mining *multidimensional sequential rules*. It characterizes the main sequences and performs in three steps. First, it mines *sequential patterns* and sets them as class *models*. In the second step, a similarity algorithm is used to gather each *model* with similar sequences of the database to form the classes. Then in the third step, a characterization algorithm is used to extract attribute values characterizing each class. Compared to existing works, our approach: (i) allows extracting rules of the form  $(a_1, a_2, a_i, \dots, a_n) \rightarrow \langle s_1, s_2, s_i, \dots, s_n \rangle [R]$  where attribute values  $(a_1, a_2, a_i, \dots, a_n)$  are characteristics of objects whom sequences are similar to  $\langle s_1, s_2, s_i, \dots, s_n \rangle$  with a significance  $R$ ; ii) considers similar sequences rather than their exact values, producing more relevant knowledge for a better decision-making. In perspective, this approach will be extended to multidimensional spatial and temporal sequences in order to characterize the trajectories of moving objects. At this end, similarity search of trajectories may use the approach of [23] which is also based on LCSS.

## References

1. Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules. In Proc. of the 20th Int. Conf. Very Large Data Bases (VLDB), Santiago, Chile, September (1994)
2. Agrawal, R., Srikant, R.: Mining sequential patterns. In Proc. of the 11th Int'l Conference on Data Engineering, Taipei, Taiwan, March (1995)
3. Altschul, S.F. Gish W., Miller W., Myers E.W. and Lipman D.J.: Basic Local Alignment Search Tool. *J.Mol. Biol.* 215: 403-410, 1990.
4. Beyer, K., Ramakrishnan, R.: Bottom-up computation of sparse and iceberg cubes. In Proc. 1999 *ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'99)*, pp. 359-370, Philadelphia, PA, June 1999.
5. Cheng, H., Yan X., and Han J.! IncSpan: Incremental mining of sequential patterns in large database. In *Proc. 2004 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'04)*, pp. 527-532, Seattle, WA, Aug. 2004.
6. Freschi, V., Bogliolo, A.: Longest Common Subsequence between Run-Length-Encoded Strings: a New Algorithm with Improved Parallelism, *Elsevier Information Processing Letters*, Vol. 90, No 4, pp. 167-173, 2004.
7. Gidófalvi G. and Pedersen T. B.: Mining Long Sharable Patterns in Trajectories of Moving Objects. In Proc. of STDBM, pp. 49-58, 2006.
8. Han, J., Jamil, H. M., Lu, Y., Chen, L., Liao, Y., Pei, J.: DNA Miner: A system prototype for mining DNA sequences. In the proc. of the ACM SIGMOD International Conference on the management of data, Day 21-24, (2001), Santa Barbara, CA, USA
9. Han, J., Kamber, M.: *Data Mining Concepts and Techniques*. Morgan Kaufmann Publishers, ISBN 1-55860-489-8, 2001.

10. Jay, A., Johannes, G., Tomi, Y., Jason F.: Sequential Pattern Mining using A Bitmap Representation. SIGMOD, pp 429-435, July (2002), Edmonton, Alberta, Canada
11. Massegli, F., Cathala, F., and Poncelet, P.: The PSP approach for mining sequential patterns. Proceedings of the *2nd European Conference on Principles of Data Mining and Knowledge Discovery in Databases (PKDD'98)*, Lecture Notes in Artificial Intelligence, Springer Verlag, pp. 176-184, Nantes, France, September 1998.
12. Massegli, F., Poncelet, P., and M. Teisseire: Incremental mining of sequential patterns in large databases. In *Data & Knowledge Engineering (DKE)*, Vol. 46, pp. 97-121, July 2003.
13. Pei, J., Han, J., Mortazavi-Asl, B., and Pinto, H.: *PrefixSPAN*: Mining sequential patterns efficiently by prefix-projected pattern growth. In Proc. of the International Conference on Data Engineering (ICDE), pp 215–224, (2001)
14. Pei, J., Han, J., and Wang W.: Constraint-Based Sequential Pattern Mining: The Pattern-Growth Methods". *Journal of Intelligent Information Systems*, Volume 28, Number 2, pages 133-160, April, 2007, Springer-Verlag.
15. Pearson, W., and Lipman, D.: Improved Tools for Biological Sequence Analysis, In *Proceedings of National Academic Science*, 85, 1988, p. 2444-2448.
16. Pinto, H., Han, J., Pei, J., Wang, K., Chen, Q., and Dayal. U.: Multi-dimensionnal sequential pattern mining. In *ACM CIKM*, 2001, pages 81-88.
17. Savary, L., Zeitouni, K.: Indexed Bit Map (IBM) for Mining Frequent Sequences. *9<sup>th</sup> European Conference on Principles and Practice of Knowledge Discovery in Databases, PKDD 2005.*, Lecture Notes in Computer Science n° 3721 / 2005, Springer-Verlag, Porto, October, 2005, pp. 659 – 666.
18. Savary L., Zeitouni K.: Mining Multidimensional Sequential Rules - A Characterization Approach, First International Workshop on Mining Complex Data in conjunction with ICDM'05, (IEEE MCD'05), November 27, 2005, Houston, Texas, USA, pp. 99-102.
19. Spiliopoulou, M., Faulstich Lukas, C., Winkler, K.: A data miner analyzing the navigational behaviour of web users. In Proc. Of the Workshop on Machine Learning in User Modelling of the ACAI'99 Int. Conf., Creta, Greece, July (1999)
20. Srikant, R., Agrawal, R.: Mining Sequential Patterns : Generalizations and Performance Improvements. Proc. 5th EDBT, Mars 25-29, (1996). Avignon, France. pp 3-17
21. Srikant R.: Fast Algorithms for Mining Association Rules. In Proc. of the 20<sup>th</sup> Int. Conf. Very Large Data Bases (VLDB), Santiago, Chile, Septembre 1994.
22. Srivastava, J., Cooley, R., Deshpande, M., Tan, P-N. : Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data, *ACM SIGKDD Explorations*, Vol.1, No. 2, 2000, p. 12-23.
23. Vlachos, M., Kollios, G., Gunopulos, D.: Discovering Similar Multidimensional Trajectories, In Proc. of 18th International Conference on Data Engineering (ICDE), pp. 673-684, San Jose, CA, 2002.
24. Wang, D., Cheng, T.: A spatio-temporal data model for activity-based transport demand modeling.. *International Journal of Geographical Information Science*, (2001), 15( 6), pp 561-585
25. Zaki, M. J.: Efficient Enumeration of Frequent Sequences. Int. Conference on Information and Knowledge Management, November( 1998), Washington DC
26. Zaki M. J., Lesh N., and Ogihara M.. PLANMINE: Sequence mining for plan failures. In *Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining (KDD'98)*, pages 369{373, New York, NY, Aug.1998.