



**HAL**  
open science

## **S2MP: Similarity Measure for Sequential Patterns**

Hassan Saneifar, Sandra Bringay, Anne Laurent, Maguelonne Teisseire

► **To cite this version:**

Hassan Saneifar, Sandra Bringay, Anne Laurent, Maguelonne Teisseire. S2MP: Similarity Measure for Sequential Patterns. AusDM: Australasian Data Mining, Nov 2008, Adelaide, Australia. pp.095-104. lirmm-00324488

**HAL Id: lirmm-00324488**

**<https://hal-lirmm.ccsd.cnrs.fr/lirmm-00324488>**

Submitted on 25 Jun 2009

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# S<sup>2</sup>MP: Similarity Measure for Sequential Patterns

Hassan Saneifar      Sandra Bringay      Anne Laurent      Maguelonne Teisseire

Laboratory of Informatics, Robotics, and Microelectronics of Montpellier (LIRMM)  
University of Montpellier 2,  
161 rue Ada, 34392 Montpellier Cedex 5, France  
Email: {saneifar, bringay, laurent, teisseire}@lirmm.fr

## Abstract

In data mining, computing the similarity of objects is an essential task, for example to identify regularities or to build homogeneous clusters of objects. In the case of sequential data seen in various fields of application (e.g. series of customers purchases, Internet navigation) this problem (*i.e. comparing the similarity of sequences*) is very important. There are already some similarity measures as Edit distance and LCS suited to simple sequences, but these measures are not relevant in the case of complex sequences composed of sets of items, as is the case of sequential patterns. In this paper, we propose a new similarity measure taking the characteristics of sequential patterns into account.  $S^2MP$  is an adjustable measure depending on the importance given to each characteristic of sequential patterns according to context, which is not the case of existing measures. We have experimented the accuracy and quality of  $S^2MP$  against Edit distance by using them in a clustering of sequential patterns. The results show that the clusters obtained by  $S^2MP$  are more homogeneous. Moreover these clusters are more precise and more complete according to the clusters obtained using Edit distance. The experiments show also that  $S^2MP$  is efficient in term of calculation time and size of used memory.

**Keywords:** Data Mining, Sequential Patterns, Similarity Measure, Clustering, Clustering of Sequential Patterns, S<sup>2</sup>MP.

## 1 Introduction

In some areas, like biology, logs analysis, anomaly detection, natural language processing and telecommunications, data can be seen in the form of sequences. Sequential patterns introduced by Agrawal & Srikant (1995) represent a frequent diagrams often extracted from sequential databases. Sequential patterns can be considered as an extension of association rules on the dimension of time. Indeed, they highlight inter-transaction associations. For example, the frequent sequential patterns extracted from a market basket data identify common and frequent customers behaviour in terms of purchased products. An example of sequential patterns is  $\langle\langle(Chocolate, Soda)(cakes, chips)(leanness\ product)\rangle\rangle$ , which means: “customers buy chocolate and soda in the same time, then in the next purchase, they buy cakes and chips and then they come back later on to buy a slimming product.”

The extraction of frequent sequential patterns in these areas provides important knowledge about frequent correlations. However, these patterns do not always convey enough information for the end-users. In order to get a clear view of the data, the clustering of sequential patterns is for instance a solution to group similar behaviours uncovered by frequent sequential patterns. This facilitates the interpretation of sequential patterns, allows to model behaviours and also to seek for outliers in the data. For clustering sequential patterns, the similarity of sequential patterns needs to be computed. Note that comparing sequential patterns has many other applications than clustering. For example, the extraction of sequential patterns under similarity constraints is of great interest, as well as sequential pattern visualization.

In this context, the definition of similarity may vary depending on the type of resemblance that we look for. The different similarity measures may reflect the different faces of data and of their context. Two objects can be seen very similar by one measure and very different by another measure (Moen 2000). Moreover, we argue that similarity has not to be always symmetrical, as pointed out by Tversky (1977), with the famous example he provided in his seminal paper: We say “Turks fight like tigers” and not “Tigers fight like Turks”. In some applications, for instance, the extraction of sequential patterns under similarity constraints or querying a set of sequential patterns, there is a reference pattern that we compare others with. Indeed, we do a directional comparison. In these applications, a non-symmetric measure is well applicable. Several approaches have been developed to compare the similarity between two sequences in particular in bioinformatics. However, the existing measures are not adapted to the specific characteristics of sequential patterns.

To compute the similarity of sequential patterns, we define here a similarity measure (S<sup>2</sup>MP: Similarity Measure for Sequential Patterns), which takes the characteristics and the semantics of sequential patterns into account. This measure compares two sequential patterns both at the level of itemsets and their positions in the sequences and also at the level of items in itemsets, thus resulting from the combination of two scores:

1. the score given by the weight of itemsets mapping (resemblance between the items of mapped itemsets),
2. the score given by the resemblance of corresponding itemsets in terms of their positions in the two sequences.

S<sup>2</sup>MP is a measure which is very well suited to the contexts and the characteristics of sequential patterns. Each score may vary according to the context. It makes hence S<sup>2</sup>MP a modular measure. For example, according to the definition of resemblance of the itemsets in a particular field, we can adapt the score given by the weights

of the mapping of the itemsets. We can also decide that the order is more important than the resemblance of the mapped itemsets and change the coefficients of the two scores in the calculation of the final similarity. This makes our measure flexible, which is not the case of the other existing measures.

The paper is organized as below. Section 2 states the problem. In Section 3 we describe the existing works on the similarity measures for the sequential patterns. Our similarity measure ( $S^2MP$ ) is presented in Section 4. The results obtained by some experiments on  $S^2MP$  are detailed in Section 5.

## 2 Problem Statement

The volumes of data stored in databases are dramatically increasing. In many applications like telecommunication, bio-informatics, market basket data etc. the data are stored in sequential form. In general, we can consider two major types of **sequences**:

- sequence of items,
- sequence of itemsets.

The sequence of items, are the most simple kind of sequences. In such a sequence, the elements of sequence are atomic. But, in many real application (*e.g.* market basket data), the sequences have more complex elements. The sequences of itemsets are an example of complex sequence. In Data Mining problems, we can note two kinds of **itemset sequences**:

- Data sequences,
- Frequent Sequential Patterns.

We consider here a transactional database of market basket data containing a set of transactions, where every transaction is a *set of items* (attributes) usually referred to as an *itemset*. A data sequence is defined as follow:

**Definition 2.1.** A **data sequence** consists of all transactions of a customer when they are ordered chronologically.

Frequent sequential patterns introduced by Agrawal & Srikant (1995) are a kind of **schema** extracted from data sequences. Indeed, frequent sequential patterns are the frequent subsequences of data sequences of a transactional database. We define the sequential patterns as below:

**Definition 2.2.** A **sequential pattern** is a non-empty ordered list of itemsets where an itemset is a set (non-ordered) of items.

Although the data sequences and sequential patterns are semantically different (sequential patterns are the schemas extracted from data sequences), they share some common characteristics. The main characteristics of itemset sequences (*e.g.* sequential patterns and data sequences) are:

1. itemsets as a set of items (non-ordered),
2. order of itemsets in sequence.

As described, in itemset sequence (*e.g.* data sequences, sequential patterns), the elements of sequence (*i.e.* itemsets) are composed of various items. Thus, the ways that we treat the sequences of items are not necessarily adapted to the sequences of itemsets like sequential patterns. In this paper, we are specially interested to compare the similarity between the sequential patterns.

Sequential patterns are very interesting kind of diagram extracted from sequential data. They describe the

inter-transaction correlations. In order to find regularities from such data (**itemset sequences**), it is necessary to describe how far from each other two data objects are. This is the reason why **similarity** between objects is one of the central concepts in data mining and knowledge discovery (Moen 2000). According to the volumes of data, we should consider also the scalability aspect of the similarity measures.

A similarity measure for sequential patterns can be used for clustering of sequential patterns. The principle of such a clustering is to regroup the extracted sequential patterns into several clusters. Each cluster represents a homogeneous kind of correlations. The clustered sequential patterns can, for instance, be used to create the behaviour profiles. For anomaly detection using data mining techniques, for example, we can use the sequential patterns extracted from normal connection logs to identify the general behaviour of network users. Several kinds of extraction are conceivable. But in any case, the patterns should be regrouped to achieve more abstract behaviour representation. The clustering of sequential patterns is a relevant and scalable solution for behaviour modeling. (Sequeira & Zaki 2002).

Besides, by clustering, outliers in data can be identified. Sequential patterns which are not assigned to any cluster, may be considered as anomalous. In market basket data, for example, the clustering of sequential patterns may help in customer segmentation or prediction in terms of their purchasing behaviour.

The notion of similarity between sequential patterns could also be used when extracting sequential patterns. The extracted sequential patterns by apriori-like algorithms (Agrawal & Srikant 1995) are usually very voluminous. There are thus many works trying to integrate some constraints like similarity constraint (Capelle et al. 2002) to reduce the size of the output of the algorithms and to better meet the end-user needs. Given a reference pattern, the idea is to extract only the patterns that are similar to the reference pattern.

The querying a set of sequences is another application of similarity measure for sequential patterns. Given a sequential pattern as a query, for example, we look for the similar patterns. Querying sequences sets has real application in bio-informatics and more generally in sequential patterns visualisation.

As described, a similarity measure has many applications in sequence analysis especially when considering sequential patterns. A great deal of works has been done in the field of item sequences. On the contrary, there are not so many works in itemset sequences area. The existing measures, used for item sequences, are not necessarily adapted to the itemsets sequences. Hence, we define a similarity measure which takes the characteristics of itemset sequences into account. As domain knowledge about the notion of similarity can vary according to different contexts, we define a similarity measure that is adaptable to the context. That is the reason why we define a modular measure by combining two scores. The final similarity degree is the weighted average of values of these scores.

## 3 Related Works

We report here the main approaches dealing with comparing sequential data especially sequential patterns. The two main similarity measures used for itemset sequences are **Edit distance** and **LCS**. We explain the disadvantages of these two measures for sequential patterns. Next, we cite an approach based on the comparison of corresponding itemsets.

The **Edit distance** (Levenshtein 1966) was used by Capelle et al. (2002) for extracting sequential patterns under similarity constraints. The authors define a sequen-

tial pattern as an ordered list of symbols belonging to  $\Sigma$  where  $\Sigma$  is a finite set of alphabet. We show why this measure is irrelevant for sequential patterns by taking an example according to the given definition and representation of sequential patterns by Capelle et al. (2002):

**Example 3.1.** Given two sequential patterns  $M_1 = \{(ab)(c)\}$  and  $M_2 = \{(a)(c)\}$  represented as:  
 $M_1\{(ab)(c)\} \Rightarrow X \rightarrow Y \mid X = (ab), Y = (c)$   
 $M_2\{(a)(c)\} \Rightarrow Z \rightarrow Y \mid Z = (a), Y = (c)$

where  $X, Y, Z$  are symbols from  $\Sigma$

Since Edit distance's operators are applied on the elements of sequence ((i.e. *itemsets*)), the distance is the cost of replacing  $X$  and  $Z$  ( $repl(X, Z, 1)$ ). In fact, in this work, an itemset in a sequential pattern is reduced to an event type. Hence, a sequential pattern is treated as an event type sequence<sup>1</sup>. In such a sequences, an event type is characterized by the values of some attributes. A list of event types ordered according to the occurrence time of events is an event type sequences (Mannila & Ronkainen 1997, Moen 2000). As described, in this work, an itemset is seen as an event type when their items are considered as the values of the event's attributes. Hence, the itemsets  $(ab)$  and  $(a)$  are treated as two symbols (event) completely different. However, we argue that  $(ab)$  and  $(a)$  are not completely different; but on the contrary, these are two similar behaviours.

Although a sequential pattern is sometimes seen as an event sequence (like in this work), this interpretation of itemset as an event is not always relevant. We explain this issue in more details with Examples 3.2 and 3.3.

**Example 3.2.** Let  $R = \{Sen_1, Sen_2, \dots, Sen_m\}$  be a set of sensors in an automatic alarms system. An alarm (*event*) occurs according to the values of sensors within a specific time.  $Alarm_A$ , for instance, is characterized by the values of sensors:

$A = (Sen_1 = 0, Sen_2 = 1, Sen_3 = \mathbf{0})$ . If, for example, the value of  $Sen_3$  becomes 1, the system is in a new situation and it sets off hence another alarm  $Alarm_B = (Sen_1 = 0, Sen_2 = 1, Sen_3 = \mathbf{1})$ . We see that despite a small difference in the attributes (value of sensor  $Sen_3$ ), but according to the data and the context,  $Alarm_A$  and  $Alarm_B$  are two events (*situation of the system*) completely different.

**Example 3.3.** Let us now consider a sequential pattern:

$M = \{(chips, soda, breads)(pizza)(chips, soda, chocolate)(flour)\}$  extracted from market basket data. The two itemsets  $(chips, soda, breads)$  and  $(chips, soda, chocolate)$  differ by a single item, but in this context, we know that these two items correspond to two very close behaviours of a customer. Thus, we can not consider them as two behaviours (*events*) being completely different.

The Edit distance measure is also used in ApproxMAP approach to cluster the sequences of itemsets. ApproxMAP developed by Kum et al. (2003), identifies the consensus sequences in large database in two phases : (1) clustering of itemsets sequences (2) extraction of consensus patterns directly from each cluster. In Phase 1, the authors used Edit distance as a measure of similarity, but with a modification on the cost of "replacement operator" to adjust the measure to itemsets sequences. The *normalized set difference* is adopted as the cost of replacement operator. Although this modification overcomes the disadvantage of Edit distance argued previously, the authors noted that the *normalized set difference* emphasis the common elements. This behaviour is appropriated if the commonalities are more important than the differences.

In addition, as noted by (Moen 2000), the type of edit operations and their costs have a remarkable effect on what kind of sequences are considered to be similar or not. She indicates that it is more natural to give more weight to the insertion (remove) of rare itemsets that to

insertion (remove) of frequent itemsets in the sequences. With different choices, we obtain different results.

Edit distance is not adaptable (*configurable*) to the various definitions of similarity. In sequential patterns extracted from bio-informatics data (*data from the analysis of DNA chips*), for example, to seek similar patterns, the content of itemsets (*i.e items*) is more important than the order of itemsets. It means that if we look for similar patterns, we should consider the similarity of itemsets according to their contents more important than the similarity of itemset's order. However, Edit distance does not take this characteristic of data into account. Hence, Edit distance is not sensible to the different definitions of similarity for sequences.

According to these examples, for a relevant comparison, it is necessary to compare two sequential patterns by considering their itemsets and their positions in sequences **and** importantly by considering the content of itemsets (*i.e. common and non-common items*). We should pay attention that the elements of a sequential patterns (*i.e. itemsets*) are not the atomic elements. We should not hence treat them without considering their content (*items*). Also, it is necessary that a similarity measure be adaptable to the different contexts and to the characteristics of data. This allows us to capture different kind of similarity.

The **LCS** measure (Longest Common Subsequence) is used for the comparison of sequences (Sequeira & Zaki 2002). The *LCS* gives the length of the longest common subsequence of two sequences. It is possible to use *LCS* to compare the similarity of sequential patterns (*itemsets sequences*) without being optimal. We note three reasons why the *LCS* is not a optimal measure for sequential patterns (*itemset sequences*).

Firstly, *LCS* does not take the position of itemsets (*in order of sequence*) into account in the two sequences.

**Example 3.4.** Let us consider:

$M_1 = \{(bc)(df)(e)\}$   
 $M_2 = \{(abc)(mn)(de)(egh)(fg)\}$   
 $M_3 = \{(e)(bc)(df)\}$ .

$LCS(M_1, M_2) = 2$  and  $LCS(M_1, M_3) = 2$  with the longest common subsequence =  $\{(bc)(d)\}$ . This subsequence ( $\{(bc)(d)\}$ ) corresponds to the consecutive itemsets in  $M_1 = \{(bc)(df)(e)\}$  and  $M_3 = \{(e)(bc)(df)\}$ . But, it is not appeared consecutively in  $M_2 = \{(abc)(mn)(de)(egh)(fg)\}$ . Obviously, *LCS* does not take into account this fact. However, semantically the emergence of the subsequence  $\{(bc)(d)\}$  in  $M_1$  and  $M_3$  is not similar to its appearance in  $M_2$ .

Secondly, *LCS* does not consider the length of the part which is not common.

**Example 3.5.** The non-common part in  $M_2$  (*i.e.  $\{(abc)(mn)(de)(egh)(fg)\}$* ) is longer than  $M_3$ 's (*i.e.  $\{(e)(bc)(df)\}$* ). This is because the value of *LCS* is not normalized by the number of items in the sequence.

Thirdly, the number of different items in itemsets (in which the subsequence appears) does not affect the value of *LCS*.

**Example 3.6.** In  $M_2$ , the itemset  $(bc)$  of subsequence is included in the itemset  $(abc)$  while in  $M_1$  and  $M_3$ , it is included in the itemset  $(bc)$ . *LCS* does not consider that in the itemset  $(abc)$  of  $M_3$ , there is another item different from "bc" (*i.e. "a"*). This problem is not resolved with the normalization because it depends on the number of items in sequence and not by the number of items in itemsets in which the subsequence appears.

A comparison of the similarity between two multidimensional sequential patterns is done by Plantevit et al.

<sup>1</sup>Edit distance is also used for event sequences (Moen 2000)

(2007) for outliers detection in a data cube. The distance between two multidimensional sequences is defined as:

Let  $S_1 = \{b_1, b_2, \dots, b_k\}$  and  $S_2 = \{b'_1, b'_2, \dots, b'_k\}$  be two multidimensional sequences,  $dist$  is a distance measure and  $Op$  an aggregation operator. The distance between  $S_1$  and  $S_2$  is defined as follows:

$d(s_1, s_2) = Op(dist(b_j, b'_j))$  for  $j = 1 \dots k$ . Comparison is done between corresponding blocks<sup>2</sup>. It means that we compare the block  $i$  in  $S_1$  with the block  $i$  in  $S_2$ . This kind of comparison is not useful when, for example, there is a shift in one of the sequence.

There are some approach developed to compare the XML document (Lee et al. 2004). A XML structure can be seen as a sequence of complex objects. But in this kind of sequences the object are not ordered. It means that there is not any order relation between objects. But in this paper, we introduce a similarity measure for sequential patterns. The order relation between the objects of a sequential pattern (*i.e.* *items*) is a fundamental concept. Thus, we must compare the sequential pattern according to their order.

The existing measures in the literature have some disadvantages in the case of sequential patterns or they are just applicable for sequences of items. A similarity measure for sequential patterns must take into account the fact that sequential patterns are sequences of ordered *items* and not *items*. The positions of items in sequences (*distance in order*) must also be taken into account when calculating the similarity. Moreover, the number of common items and non-common items must be considered both at the sequence level as well as at the level of corresponding items.

In this paper, we define a similarity measure ( $S^2MP$ : Similarity Measure for Sequential Patterns) which takes these characteristics into account.

#### 4 $S^2MP$ : Description

Our similarity measure ( $S^2MP$ : Similarity Measure for Sequential Patterns) results from the aggregation of two scores:

1. The *mapping score* which measures the resemblance of two sequences based on the links that can be established between items,
2. The *order score* which measures the resemblance of the two sequences based on the order and positions of items in sequences.

As we consider the itemset proximity as very important, we first build the best mapping of itemsets based on the similarity of their contents (common and non-common items) in step 1. The mapping score is calculated at the end of this step by considering all the mappings and their degree of mapping (*weight of mapping*).

In step 2, the goal is to give a score according to the resemblance of two sequences in terms of their order and position of mapped itemsets in two sequences. We take the result of step 1 without any rearrangement of the mappings. Then, we are firstly looking for the mappings, which comply with the order of items in two sequences (*cf.* Figure 2). It means that we discard the mappings, which associate the current itemset of a sequence to an itemset before the last mapped itemset of other sequence. This kinds of mapping are called cross-mappings (*cf.* Figure 2). Moreover, we also measure the resemblance of the mapped itemsets according to their positions in two sequences.

Finally, the order score is calculated based on the proportion of the mappings complying with order to the all

<sup>2</sup>Here, a data block can be seen as an itemset

mappings in addition to the score measuring the similarity of itemsets according to their positions.

**Step 1: Mapping Score Calculation.** Let us consider  $Seq_1$  and  $Seq_2$  as two sequential patterns to be compared to. For each itemset  $i$  in  $1^{st}$  sequence  $Seq_1(i)$ , we are looking for the most similar itemset  $j$  in the  $2^{nd}$  sequence  $Seq_2(j)$ . Then we match these two itemsets and we give each pair of mapping a weight which is the degree of similarity between the two itemsets. We define below how the weight of mapping is computed:

**Definition 4.1.** *Weight*( $i, j$ ) between the  $i^{th}$  itemset of  $Seq_1$  and the  $j^{th}$  itemset of  $Seq_2$

$$\text{Weight}(i, j) = \frac{|Seq_1(i) \cap Seq_2(j)|}{(|Seq_1(i)| + |Seq_2(j)|) / 2}$$

For equal weights, we choose the itemset with the lowest *timeStamp* ( $ts$ )<sup>3</sup>. It means the itemset located before others regarding the order of sequence to comply with the order of itemsets. Thus, the mapping is formalized as:

$$\text{Mapping}(Seq_1(i), Seq_2(j)) \mid \text{Weight}(i, j) = \max_{x \in [0, |Seq_2|]} (\text{Weight}(i, x)) \\ \&\& \text{Weight}(i, j) \neq 0$$

$$\text{If } \text{Weight}(i, j) = \text{Weight}(i, k) \Rightarrow ts(j) < ts(k)$$

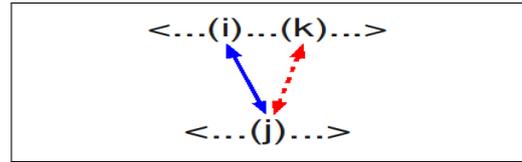


Figure 1: Conflict of mapping.

**Conflict of mapping.** When computing the mappings, an itemset selected from the  $2^{nd}$  sequence to correspond to an itemset of the  $1^{st}$  sequence may have already been mapped with another itemset. This situation, called “conflict of mapping”, is illustrated in Figure 1. The itemset  $Seq_2(j)$  is proposed to map with  $Seq_1(k)$ . But  $Seq_2(j)$  is already mapped with  $Seq_1(i)$ . To solve this problem, we must find another mapping candidate for one of the itemsets in conflict. For this, we use a function (*SolveConflict*) to propose a new itemset as a new candidate for mapping.

In the *SolveConflict*, for each itemset (*in conflict*) of sequence 1 (*i.e.*  $Seq_1(i)$  and  $Seq_1(k)$ ), we are seeking two other mapping candidates in the  $Seq_2$  (other than  $Seq_2(j)$ , the current candidate itemset):

- The  $1^{st}$  candidate is the itemset located before  $Seq_2(j)$  which also owns the maximum weight among itemsets placed before  $Seq_2(j)$ . We name them as: *nextMaxBefore<sub>i</sub>* as a candidate for  $Seq_1(i)$  and *nextMaxBefore<sub>k</sub>* for  $Seq_1(k)$ .
- The  $2^{nd}$  candidate is selected the same way but it is sought after  $Seq_2(j)$ : *nextMaxAfter<sub>i</sub>* for  $Seq_1(i)$  and *nextMaxAfter<sub>k</sub>* for  $Seq_1(k)$ .

Next, we create all possible mapping pairs. We get at the most four possible cases of mapping:

- $\langle Seq_1(i), Seq_2(j) \rangle, \langle Seq_1(k), nextMaxBefore_k \rangle,$
- $\langle Seq_1(i), Seq_2(j) \rangle, \langle Seq_1(k), nextMaxAfter_k \rangle,$
- $\langle Seq_1(k), Seq_2(j) \rangle, \langle Seq_1(i), nextMaxBefore_i \rangle,$
- $\langle Seq_1(k), Seq_2(j) \rangle, \langle Seq_1(i), nextMaxAfter_i \rangle.$

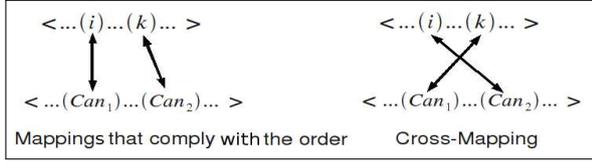


Figure 2: Cross mapping and the order of itemsets.

We consider here two opposite kinds of mapping pairs, namely mapping pairs that comply with the order and mapping pairs, which violate the order (see Figure 2) referred to as *cross-mappings*.

More precisely, let us consider a pair of mappings: in the first mapping the itemset at position  $i$  on the first sequence is mapped with the itemset at position  $i'$  in the second sequence; in the second mapping the itemset at position  $j$  (where  $i < j$ ) on the first sequence is mapped with the itemset at position  $j'$  in the second sequence. Then this mapping pair is said to be *order compliant* if  $i' < j'$ . It is said to be *cross-mapping* if  $i' > j'$ .

We calculate the relevance of the four possible cases of mapping with *localSim*. The calculation of *localSim* depends on the type of mappings:

– when mappings comply with order, we consider:

$$\text{localSim}(i, \text{Can}_1)(k, \text{Can}_2) = \frac{\text{Weight}(i, \text{Can}_1) + \text{Weight}(k, \text{Can}_2)}{2}$$

– when we have cross mapping, as the order is half respected in cross-mapping, we divide the *localSim* by two. We thus consider:

$$\text{localSim}(k, \text{Can}_1)(i, \text{Can}_2) = \frac{1}{2} \times \frac{\text{Weight}(k, \text{Can}_1) + \text{Weight}(i, \text{Can}_2)}{2}$$

The mapping pair having the highest *localSim* is then selected as the output of the *SolveConflict* function. Note that at the end, the initial candidate (*i.e.*  $\text{Seq}_2(j)$ ) is proposed as a candidate either for  $\text{Seq}_1(i)$ , or for  $\text{Seq}_1(k)$  depending on the value of *localSim*.

**Conflict Loop.** Mapping function handles the cases when there is a conflict loop. The conflict loop is a situation where the candidate itemset, which is proposed to resolve a conflict of mapping (output of the *SolveConflict*) is itself mapped to another itemset. In the case of a conflict loop, we continue to call the function *SolveConflict* until its output (new proposed candidate) is not already mapped. In each loop, we exclude the proposed candidate which is already mapped. In the situation where there is not any candidate for one of the itemsets in conflict (*i.e.*  $\text{Seq}_1(i)$  and  $\text{Seq}_1(k)$ ), we associate the initial candidate ( $\text{Seq}_2(j)$ ) with the itemset owning the highest weight of mapping. The other itemset remains without any corresponding itemset in the  $2^{\text{nd}}$  sequence (without mapping).

**Output of Step 1.** At the end of step 1, we have the final mappings. We come up with the mappings associating each itemset from sequence 1 with the more relevant (in terms of common and non-common items) itemset from sequence 2. These mappings are stored in a list named *mapOrder*. The first element (resp.  $2^{\text{nd}}, \dots, n^{\text{th}}$ ) in the list is the timeStamp of the itemset from sequence 2 corresponding to the first itemset (resp.  $2^{\text{nd}}, \dots, n^{\text{th}}$ ) from

<sup>3</sup>The *timeStamp* in our context, corresponds to the position of itemsets in the order of sequence. For example in the sequence (a)(b)(c),  $ts(ab) = 2$ .

sequence 1. We thus have:

$$\text{mapOrder} = \{t_1, t_2, \dots, t_i, \dots, t_n\}$$

$t_i$  = the timeStamp of the itemset of the  $\text{Seq}_2$  mapped with  $i^{\text{th}}$  itemset in  $\text{Seq}_1$ .

At last, we calculate the *mapping score* by average of weight of mappings (*AveWeightScore*).

We explain here why the content of itemsets, which have cross-mappings, should be considered into the mapping score. In fact, if we consider only the weight of itemsets, which have ordered mappings, we lose some information about the similarity of sequences. For instance, when comparing (a)(b) with (b)(a) and (a)(b) with (b)(d), if we take only the weight of itemsets with ordered mapping (*i.e.* the weight of (b)→(b)), then (a)(b) and (b)(a) will be treated as same as (a)(b) and (b)(d). But, by considering the weight of all found mappings, we consider also the weight of (a)→(a) (*equal to 1*) and the weight of (a)→(d) (*equal to 0*).

We now go to Step 2, which aims at evaluating to what extent the mappings found in Step 1 respect the order of the sequences.

**Step 2: Order Score Calculation** This step is two-folded:

- *totalOrder*,
- *positionOrder*.

Firstly, we aim at discarding cross-mappings (*cf.* Figure 2) using *totalOrder*. *mapOrder* is a list of integers representing the timeStamps of itemsets. In this list, as while as the integers (timeStamps) are increasing, the corresponding mappings are totally ordered (there are not cross-mappings). We look hence for the maximum increasing subsequences of *mapOrder* to find all the possible series of mappings which comply with order. In this way, we avoid the cross-mappings.

More precisely, *totalOrder* measures the percentage of non-cross mappings (*i.e.* mappings respecting the order of itemsets in the two sequences). The calculation of *totalOrder* is formalized as follow:

$$\text{totalOrder} = \frac{\text{nbOrderedItemSets}}{\text{aveNbItemSets}}$$

*nbOrderedItemSets* = The number of itemsets in the increasing subsequence  
*aveNbItemSets* = The average of the number of all itemsets in the two sequences

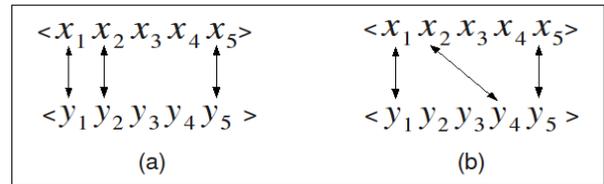


Figure 3: Distance between mappings according to positions of itemsets.

Secondly, we aim at considering the width between the mapped itemsets using *positionOrder*. For instance, Figure 3 shows that the itemsets can be mapped differently. Here, in part “a” of Figure 3, the itemsets are mapped in a closer way compared to part “b”.

The timeStamps also allows us to check similarities of mapped itemsets according to their positions in the sequences.

*positionOrder* shows if the distance between two successive mappings based on the positions of itemsets in

the sequence 1 is equal to that according to the positions of itemsets in the sequence 2. Figure 3-a shows an example where the distance between the successive mappings according to the positions of itemsets in the sequence 1 is equal to that according to the positions of itemsets in the sequence 2. But in Figure 3-b, the distance between two first mappings depending on the positions of  $x_1$  and  $x_2$  is not equal to the distance depending on the positions of  $y_1$  and  $y_2$ . The *positionOrder* formula is:

**positionOrder** =

$$\sum_{i=1}^{|sub|} \frac{|sub(i) - sub(i-1)| - |mapOrder^{-1}(sub(i)) - mapOrder^{-1}(sub(i-1))|}{aveNbItemSets}$$

$sub(i)$  = The value of  $i^{th}$  position in the subsequence

$mapOrder^{-1}(x)$  = The position of itemset "x" in *mapOrder*

At last, for each increasing subsequence of *mapOrder*, we calculate the multiplication of *totalOrder* and *positionOrder* and we keep the highest score as (*orderScore*):

$$orderScore = \max\{totalOrder(sub) \times (1 - positionOrder(sub))\}$$

$sub \in \{maximum\ increasing\ subsequences\ of\ mapOrder\}$

At the end of Step 2, the *orderScore* is calculated. The final similarity degree can now be computed by an aggregation between order score (*orderScore*) and the average of weight of mappings (*AveWeightScore*) calculated at the end of Step 1. This final degree is calculated in Step 3 as described below.

**Step 3: Final Similarity Degree Calculation.** We calculate the similarity degree *SimDegree* as an aggregation between the *orderScore* and the average weight of mapping of itemsets *AveWeightScore*. The *orderScore* compares the similarity of two sequences based on the positions of itemsets (likeness of order of itemsets) in sequences. By *AveWeightScore*, we compare the similarity in terms of common items and non-common items in itemsets. The aggregation can be a weighted average while we can define the coefficient for each score. By defining the coefficient for each score, we choose to consider the order as more (or less or as) important than the content according to the application context. Our measure is thus a very flexible measure.

**SimDegree** =

$$\frac{(orderScore \times Co_1) + (AveWeightScore \times Co_2)}{Co_1 + Co_2}$$

The calculation of the similarity is explained in the following example. We tried to treat most cases and also the conflicts of mapping in the illustration.

**Example 4.1.** Let us take  $M_1 = \{(bc)(df)(e)\}$  and  $M_2 = \{(abc)(mn)(de)(egh)(fg)\}$  that we used to show disadvantages of *LCS* as the two sequential patterns.

*Step 1: Mapping Score Calculation.* For each itemset  $M_1(i)$  in the first sequence, we are looking for the most similar itemset  $M_2(j)$  in the 2<sup>nd</sup> sequence. This is done by the weight of mapping calculations.

$$\begin{aligned} Weight(M_1(1), M_2(1)) &\implies Weight((bc), (abc)) = \frac{2}{3+2} = 0.8 \\ Weight(M_1(1), M_2(2)) &\implies Weight((bc), (mn)) = \frac{0}{2+2} = 0 \\ Weight(M_1(1), M_2(3)) &\implies Weight((bc), ((de))) = \frac{0}{2+2} = 0 \\ Weight(M_1(1), M_2(4)) &\implies Weight((bc), ((egh))) = \frac{0}{2+3} = 0 \\ Weight(M_1(1), M_2(5)) &\implies Weight((bc), ((fg))) = \frac{0}{2+2} = 0 \end{aligned}$$

We choose the case with the highest weight: *MappedItemSets.put*( $M_1(1)$ ,  $M_2(1)$ ).

We continue by the following itemset  $M_1(2)$ :

$$\begin{aligned} Weight(M_1(2), M_2(1)) &\implies Weight((df), (abc)) = \frac{0}{3+2} = 0 \\ Weight(M_1(2), M_2(2)) &\implies Weight((df), (mn)) = \frac{0}{2+2} = 0 \\ Weight(M_1(2), M_2(3)) &\implies Weight((df), ((de))) = \frac{1}{2} = 0.5 \\ Weight(M_1(2), M_2(4)) &\implies Weight((df), ((egh))) = \frac{0}{2+3} = 0 \\ Weight(M_1(2), M_2(5)) &\implies Weight((df), ((fg))) = \frac{1}{2+2} = 0.5 \end{aligned}$$

The *Weight*((df), (de)) and the *Weight*((df), (fg)) are equal, so we select the itemset with lower timeStamp (i.e. (de)) to map with the itemset (df). We have thus: *MappedItemSets.put*( $M_1(2)$ ,  $M_2(3)$ ).

We do the same for the 3<sup>rd</sup> itemset  $M_1(3)$ :

$$\begin{aligned} Weight(M_1(3), M_2(1)) &\implies Weight((e), (abc)) = \frac{0}{3+2} = 0 \\ Weight(M_1(3), M_2(2)) &\implies Weight((e), (mn)) = \frac{0}{2+2} = 0 \\ Weight(M_1(3), M_2(3)) &\implies Weight((e), ((de))) = \frac{1}{1+2} = 0.6 \\ Weight(M_1(3), M_2(4)) &\implies Weight((e), ((egh))) = \frac{1}{1+3} = 0.5 \\ Weight(M_1(3), M_2(5)) &\implies Weight((e), ((fg))) = \frac{1}{2+2} = 0 \end{aligned}$$

According to the calculation, we select the itemset  $M_2(3)$  (i.e. (de)). But this itemset (de) has already been associated with the itemset (df) of  $M_1$ . Therefore, we use the function of conflict resolving.

We look for new candidates in  $M_2$  for itemsets in conflict ((df) and (e)) before and after the current candidate itemset (de). We get the following candidates:

- **for the itemset (df):**  
 $nextMaxBefore_{(df)} = \emptyset$   
 $nextMaxAfter_{(df)} = M_2(5) = (fg)$
- **for the itemset (e):**  
 $nextMaxBefore_{(e)} = \emptyset$   
 $nextMaxAfter_{(e)} = M_2(4) = (egh)$
- **Possible pairs of mappings:**  
 $\langle\langle(df), (de)\rangle\rangle, \langle\langle(e), (egh)\rangle\rangle$   
 $\langle\langle(e), (de)\rangle\rangle, \langle\langle(df), (fg)\rangle\rangle$ .

Using the weight of mapping and considering case of cross-mapping ( $\langle\langle(e), (de)\rangle\rangle, \langle\langle(df), (fg)\rangle\rangle$ ), we get:

$$\begin{aligned} localSim(\langle\langle(df), (de)\rangle\rangle, \langle\langle(e), (egh)\rangle\rangle) &= \frac{0.5+0.5}{2} = 0.5 \\ localSim(\langle\langle(e), (de)\rangle\rangle, \langle\langle(df), (fg)\rangle\rangle) &= \frac{1}{2} \times \frac{0.6+0.5}{2} = 0.27 \end{aligned}$$

We select the pair having the highest *localSim*:  $\langle\langle(df), (de)\rangle\rangle, \langle\langle(e), (egh)\rangle\rangle$ . The itemset (df) is thus mapped with (de) and the itemset (e) with (egh): *MappedItemSets.put*( $M_1(2)$ ,  $M_2(3)$ )  
*MappedItemSets.put*( $M_1(3)$ ,  $M_2(4)$ ).

Final mappings are:

$$\begin{aligned} \langle M_1(1) = (abc), M_2(1) = (ab) \rangle \\ \langle M_1(2) = (df), M_2(3) = (de) \rangle \\ \langle M_1(3) = (e), M_2(4) = (egh) \rangle \end{aligned}$$

We create now the *mapOrder* list. We put at the  $i^{th}$  place in *mapOrder* the timeStamp of itemset mapped to

$i^{th}$  itemset  $Seq_1(i)$  of the first sequence. Hence, the 1<sup>st</sup> place in  $mapOrder$  is taken with “1” according to the timeStamp of the itemset  $M_2(1)$  mapped with the 1<sup>st</sup> itemset  $M_1(1)$  of the first sequence. For the 2<sup>nd</sup> place, the timeStamp of the itemset  $Seq_2(3)$  mapped with the 2<sup>nd</sup> itemset  $M_1(2)$  of the first sequence (*i.e.* “3”) and in the same manner we put “4” in the 3<sup>th</sup> place in the  $mapOrder$ . Therefore the  $mapOrder$  is:

$$mapOrder = \{1, 3, 4\}$$

At last, we calculate the mapping score by averaging the weight of mappings ( $AveWeightScore$ ).

$$AveWeightScore = \frac{Weight((bc),(abc)) + Weight((df),(de)) + Weight((e),(egh))}{3} = \frac{0.8 + 0.5 + 0.5}{3} = 0.6$$

$$AveWeightScore = 0.6$$

**Step 2: Order Score Calculation.** In this step the aim is to compare the order of itemsets in the two sequences. We seek all maximum increasing subsequences of  $mapOrder$  (output of step 1). In this Example, there is only one maximum increasing subsequence.

The only maximum increasing subsequence of  $mapOrder$ :

- $subseq = \{1, 3, 4\}$

According to the formula of  $totalOrder$ , of  $positionOrder$  and of  $orderScore$ :

- $totalOrder((1,3,4)) = \frac{3}{(3+5)/2} = 0.75$
- $positionOrder((1,3,4)) = \frac{|(3-1)-(2-1)|}{(3+5)/2} + \frac{|(4-3)-(3-2)|}{(3+5)/2} = 0.25$
- $orderScore = 0.75 \times (1 - 0.25) = 0.56$

**Step 3: Similarity Degree Calculation.** With the multiplication of  $orderScore$  and the  $AveWeightScore$ , we get the degree of similarity between the two sequential patterns:

$$SimDegree = 0.56 \times 0.6 = 33\%$$

## 5 Experiments

In this paper, we introduce a measure of similarity for sequential patterns. In this section, we report the experiments led to show the accuracy, the relevance and the scalability of our approach. A measure of similarity must capture the similarity of compared items. Such a measure is usually used within another algorithm like as clustering or extraction of sequential patterns under similarity constraint. It must be efficient and scalable. We consider two main directions :

- the accuracy of the similarity degree obtained by  $S^2MP$ ,
- the efficiency of the  $S^2MP$  algorithm at execution time and size of used memory.

**Accuracy of  $S^2MP$ .** We experiment  $S^2MP$  to assess its quality (*accuracy*) and compare the results obtained by  $S^2MP$  and Edit distance. We apply two clusterings of sequential patterns: one with  $S^2MP$  and one with Edit Distance. In both cases we use the same dataset. To compare clusters obtained by each measure, we calculate

the entropy of each cluster.

The dataset consists of 100 sequential patterns. We manually create 10 categories of sequential patterns. In each category, we put the similar patterns. These categories will be used as references. The sequential patterns have different sizes. Among these 10 reference categories, 4 categories contains different patterns and 6 categories contain patterns similar to the patterns of at least one other category. This allows us to assess the accuracy of each measure when it comes to distinguish clusters with a small inter-cluster distance.

We adopted the K-means clustering for sequential patterns. We cluster the patterns at first by using  $S^2MP$  and then by using Edit distance. For each clustering, we calculate the entropy of obtained clusters. We compare also the clusters with reference categories for calculating the precision and recall of each clustering. These experiments show that the cluster obtained with  $S^2MP$  are more homogeneous (*according to entropy of clusters*) than those obtained with Edit distance. Moreover, the clusters obtained by  $S^2MP$  are more accurate (*according to precision of clusters*) and more complete (*according to recall of clusters*).

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
$S^2MP$	0.98	0	0.99	0.86	0.95	0	0.97	0.95	0.65	0
Edit dist	0.97	0	0.99	1.20	0.89	0	0.98	0.98	0.70	0.99

Table 1: Entropy of clusters obtained by  $S^2MP$  and Edit distance.

The table 1 shows the entropy of clusters obtained with each measure. More entropy of a cluster is small, more cluster is homogeneous and it contains more informations. The average entropy for clustering with  $S^2MP$  is 0.63 and using Edit distance is 0.77. The precision and recall of clusters obtained by each measure is illustrated in table 2. The precision and recall are calculated based on the reference categories.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Precision ( $S^2MP$ )	0.57	1	0.53	0.71	0.65	1	0.5	0.6	0.68	1
Recall ( $S^2MP$ )	0.4	1	0.7	1	1	0.5	0.5	0.4	1	0.4
Precision (Edit)	0.6	1	0.53	0.58	0.68	1	0.4	0.62	0.83	0.57
Recall (Edit)	0.3	1	0.6	1	0.9	0.8	0.2	0.5	1	0.4

Table 2: Precision and recall of clusters using  $S^2MP$  and using Edit distance.

We also experiment  $S^2MP$  and Edit distance on their ability to identify similar sequential patterns in different contexts. We consider the bioinformatics domain and more precisely the characteristics of sequential patterns extracted from DNA chips. In this area, according to experts, the contents of itemsets (*i.e.* *items*) are more important than the order of itemsets. For example, the two sequential patterns  $M_1 = \langle (G_1, G_2)(G_3)(G_4) \rangle$ ,  $M_2 = \langle (G_3)(G_1, G_2)(G_4) \rangle$  are so similar because the content of itemsets are similar however their order are not. To experiment  $S^2MP$  in this situation, we manually create 10 categories each one contains 10 similar sequential patterns according to the content of their itemsets, which are ordered so differently in different sequences (*i.e.* *each category contains 10 sequential patterns, which are similar based in the content of itemsets but order of itemsets differs*). We also consider the categories, which contain the patterns rather similar.

We do a clustering on the data set with  $S^2MP$  by giving to the score of mapping a weight two times more than the weight of order score (*i.e.* *we configure  $S^2MP$  in the way that the content of itemsets is more important that the order of itemsets*). Then, we do other clustering on this

dataset with Edit distance. The results with  $S^2MP$  show that we can capture similar patterns according to the particular definition of similarity in this context. This shows that  $S^2MP$  is well parametrizable and is adaptable to different definition of similarity for sequential patterns. Results obtained with Edit distance in this context, are not satisfactory.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
$S^2MP$	0.99	0.52	0.99	0.99	0.99	0	0	0.99	0	0.98
Edit dist	1.2	1.2	1.3	1.3	1.3	1.2	1.4	0.95	0.4	1.7

Table 3: Entropy of clusters obtained by  $S^2MP$  and Edit distance when the contents of itemsets are more important than the order of itemsets – (e.g. patterns extracted from the DNA chips data)

Table 3 shows the relevance of  $S^2MP$  in this context and its adaptability to different definition of similarity for sequential patterns. On this dataset, the average entropy of clustering using  $S^2MP$  is 0.64 and using Edit distance is 1.19. We demonstrate the precision and recall of clusters for each clustering in the table 4. The precision and recall of clusters are calculated according to the reference categories.

**Efficiency of  $S^2MP$ .** Despite the complex appearance of our measure’s algorithm, we show that our method is very efficient in terms of runtime and size of memory used, by studying how it performs depending on three factors, as detailed below. We test execution time and size of memory used by our similarity measure in three directions: (1) depending on the number of itemsets in sequential patterns(2) depending on the number of items in sequential patterns and finally (3) depending on the number of sequential patterns that we want to calculate their similarities.

We create a matrix of similarity with  $(n \times n)$  dimensions where  $n$  represents the number of sequential patterns. Our measure of similarity is not symmetrical, we calculate thus all the matrix instead of a diagonal calculation. The time for calculating the similarity matrix is the time necessary to make  $n \times n$  comparisons of similarity. Our results show that our measure of similarity is calculated very quickly even when there are many conflict loops at the mapping phase.

The experiments are performed on a machine with a 2GHz Intel CPU with 2GB of RAM under the ubuntu Linux operating system. Our algorithm is implemented using Java 5.

**Data set.** We carry out experiments on two different types of itemset sequences:

1. frequent sequential patterns,
2. data sequences.

The frequent sequential patterns are extracted from synthetic data generated by the generator IBM quest<sup>4</sup>. In the second stage of our experimentation, we decided to make a test on data sequences because in such sequences, we are more likely to have conflicts of mapping. This allows us to study the impact of conflict on runtimes. In the tests depending on the number of itemsets and items, data sets are made up of 1000 sequential patterns (or data sequences). At each stage, we calculate the similarity matrix, (*i.e.* we realize 1,000,000 comparisons of similarity).

<sup>4</sup>www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data-mining/datasets/syndata.html

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Precision ( $S^2MP$ )	0.55	0.71	0.54	0.54	0.55	1	1	0.53	1	0.57
Recall ( $S^2MP$ )	0.5	1	0.6	0.6	0.5	0.6	1	0.7	1	0.4
Precision (Edit)	0.52	0.61	0.46	0.60	0.50	0.5	0.5	0.37	0.9	0.16
Recall (Edit)	0.9	0.6	0.6	0.3	0.5	0.6	0.7	0.3	0.9	0.2

Table 4: Precision and recall of clusters using  $S^2MP$  and using Edit distance when the contents of itemsets are more important than the order of itemsets – (e.g. patterns extracted from the DNA chips data)

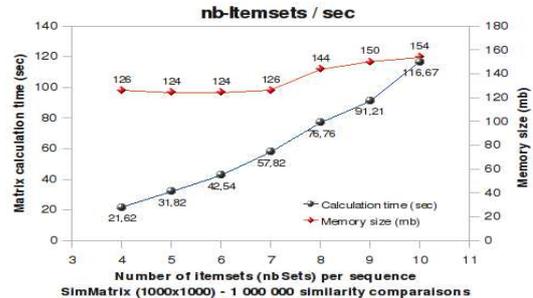


Figure 4: Calculation time and memory size depending on the number of itemsets.

**Results.** Figures 4 and 5 represent the evolution of the running time of 1,000,000 comparisons and the size of memory used according to the number of itemsets and items per sequence. According to the curves, the size of memory used does not change significantly when the number of itemsets (or items) per sequence increases. The time for calculating the similarity matrix (*1,000,000 comparisons*) when there are 10 itemsets per sequence is satisfactory (116 sec). This means that the time for calculating similarity between two sequential patterns, each one with 10 itemsets is equal to  $116\mu$  sec. Our experiments show that the number of items per sequence does not affect the runtime as much as the number of itemsets per sequence.

We show the time of calculating similarity matrix on the Figure 6 and the size of memory used on Figure 7 when the number of sequences increases. In each case, there is  $n \times n$  similarity comparisons where  $n$  is the number of sequences in the data set. We run this test on three types of sequences: the sequences with 5 itemsets, with 7 itemsets and sequences with 9 itemsets. In cases where there are 5000 sequences (*i.e.* 25,000,000 similarity comparisons), and each sequence contains 9 itemsets, the execution time is only 1974 sec.

Figure 8 shows the results of experimentation on data sequences. For each case, we noted the number of resolved conflicts when calculating the similarity matrix. The X-axis represents the different data sets. For each, the number of itemsets and the average number of items per sequence are marked. There are 1000 sequences in each data set (thus 1,000,000 similarity comparisons). The curves represent the running time of the calculating similarity matrix for each case and the size of used memory. For example, where there are 20 itemsets and on average 109 items per sequence and 103437 solved conflicts, the calculating time of 1,000,000 similarity comparisons is equal to 961 sec. We note that the size of memory used is almost constant.

With these experiments, we have shown that our measure is efficient in term of runtime and size of memory for data sequences and frequent sequential patterns.

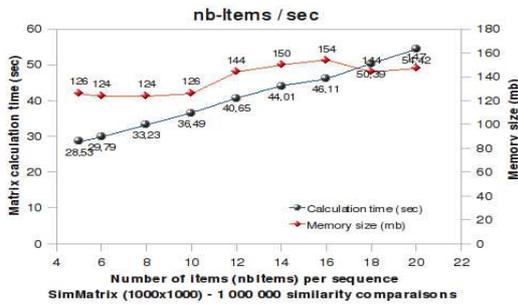


Figure 5: Calculation time and memory size depending on the number of items.

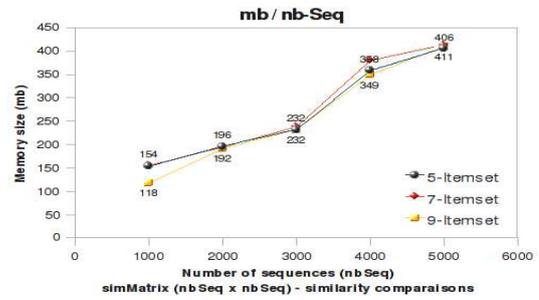


Figure 7: Size of memory used for calculating the similarity matrix based on the number of sequences.

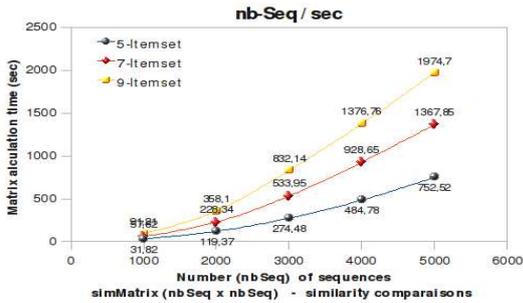


Figure 6: Time for calculating the similarity matrix based on the number of sequences.

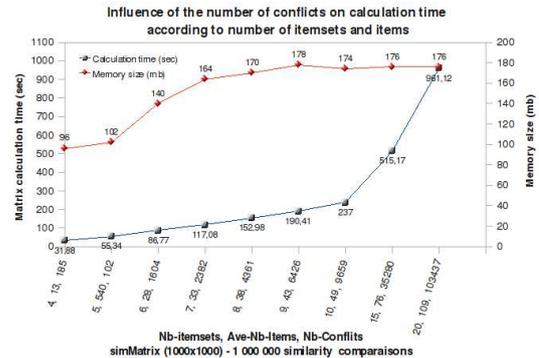


Figure 8: Influence of conflicts on the runtime of calculating the similarity

## 6 Conclusion

In this paper, we have defined a similarity measure ( $S^2MP$ ) adapted to sequential patterns taking into account all the characteristics of sequential patterns and in particular their semantics. The degree of similarity is the result of the aggregation of two scores. These scores measure the similarity of sequential patterns in terms of itemsets and their positions in the sequences (*orderScore*) but also in terms of items contained in the corresponding itemsets (*aveWeightScore*). The combination of two independent scores allows a modular measurement. It is therefore adaptable and parametrizable depending on the context, different definition of similarity and the meaning of itemset in the application domain.  $S^2MP$  overcomes the disadvantages of *LCS* and *Edit distance* in the case of sequential patterns.

Experiments show that  $S^2MP$  is more accurate than Edit distance. The clusters obtained by  $S^2MP$  are more precise and more complete than the clusters obtained by Edit distance. The experiments show also that  $S^2MP$  can be calculated very quickly even when we compare many sequences with several itemsets.

Several areas and methods as the clustering of sequential patterns, outliers detection, extraction of sequential patterns under similarity constraint, compression of sequential patterns, visualisation and querying the sequential patterns, etc are possible applications of  $S^2MP$ .

## References

Agrawal, R., Faloutsos, C. & Swami, A. N. (1993), Efficient Similarity Search In Sequence Databases, in D. Lomet, ed., 'Proceedings of the 4th International Conference of Foundations of Data Organization and Algorithms (FODO)', Springer Verlag, Chicago, Illinois, pp. 69–84.

Agrawal, R. & Srikant, R. (1995), Mining sequential patterns, in P. S. Yu & A. S. P. Chen, eds, 'Eleventh Inter-

national Conference on Data Engineering', IEEE Computer Society Press, Taipei, Taiwan, pp. 3–14.

Bozkaya, T., Yazdani, N. & Ozsoyoglu, Z. M. (1997), Matching and indexing sequences of different lengths, in 'CIKM', pp. 128–135.

Capelle, M., Masson, C. & Boulicaut, J.-F. (2002), Mining frequent sequential patterns under a similarity constraint, in 'IDEAL', pp. 1–6.

Garofalakis, M. N., Rastogi, R. & Shim, K. (1999), SPIRIT: Sequential pattern mining with regular expression constraints, in 'The VLDB Journal', pp. 223–234.

Guralnik, V. & Karypis, G. (2001), A scalable algorithm for clustering sequential data, in 'ICDM '01: Proceedings of the 2001 IEEE International Conference on Data Mining', IEEE Computer Society, Washington, DC, USA, pp. 179–186.

Hartigans, J. (1975), *Clustering Algorithms*, John Wiley and Sons, Inc.

Jianhua Zhu, Z. W. (2005), Fast: A novel protein structure alignment algorithm, Vol. 58, Bioinformatics Program, Boston University, Boston, Massachusetts; Biomedical Engineering Department, Boston University, Boston, Massachusetts.

Kum, H.-C. (2004), Approximate Mining of Consensus Sequential Patterns, PhD thesis, University of North Carolina.

Kum, H.-C., Pei, J., Wang, W. & Duncan, D. (2003), Approxmap: Approximate mining of consensus sequential patterns, in 'SDM'.

Lee, K.-H., Choy, Y.-C. & Cho, S.-B. (2004), 'An efficient algorithm to compute differences between structured documents', *Knowledge and Data Engineering, IEEE Transactions on* **16**(8), 965–979.

- Levenshtein, V. I. (1966), Binary codes capable of correcting deletions, insertions, and reversals, Technical Report 8.
- Mannila, H. & Ronkainen, P. (1997), Similarity of event sequences, *in* 'TIME '97: Proceedings of the 4th International Workshop on Temporal Representation and Reasoning (TIME '97)', IEEE Computer Society, Washington, DC, USA, p. 136.
- Moen, P. (2000), *Attributs, Event Sequence, and Event Type Similarity Notions for Data Mining*, PhD thesis, University of Helsinki, Finland.
- Morzy, T., Wojciechowski, M. & Zakrzewicz, M. (1999), Pattern-oriented hierarchical clustering, *in* 'Advances in Databases and Information Systems', pp. 179–190.
- Pei, J., Han, J., Mortazavi-asl, B., Pinto, H., Chen, Q., Dayal, U. & chun Hsu, M. (2001), Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth, pp. 215–224.
- Plantevit, M., Goutier, S., Guisnel, F., Laurent, A. & Teisseire, M. (2007), Mining unexpected multidimensional rules, *in* 'DOLAP '07: Proceedings of the ACM tenth international workshop on Data warehousing and OLAP', pp. 89–96.
- Sequeira, K. & Zaki, M. J. (2002), Admit: anomaly-based data mining for intrusions, *in* 'KDD', pp. 386–395.
- Tversky, A. (1977), *Psychological Review* (84), 327–352.