OLAP-Sequential Mining: Summarizing Trends from Historical Multidimensional Data using Closed Multidimensional Sequential Patterns

Marc Plantevit, Anne Laurent, Maguelonne Teisseire

Noname manuscript No.
(will be inserted by the editor)

OLAP-Sequential Mining: Summarizing Trends from Historical Multdimensional Data using Closed Multidimensional Sequential Patterns

M. Plantevit · A. Laurent · M. Teisseire

Received: date / Accepted: date

Abstract Data warehouses are now well recognized as the way to store historical data that will then be available for future queries and analysis. In this context, some challenges are still open, among which the problem of mining such data. OLAP mining, introduced by J. Han in 1997, aims at coupling data mining techniques and data warehousing. These techniques have to take the specificities of such data into account. One of the specificities that is often not addressed by classical methods for data mining is the fact that data warehouses describe data through several dimensions. Moreover, the data are stored through time, and we thus argue that sequential patterns are one of the best ways to summarize the trends from such databases. Sequential pattern mining aims at discovering correlations among events through time. However, the number of patterns can become very important when taking several analysis dimensions into account, as it is the case in the framework of multidimensional databases. This is why we propose here to define a condensed representation without loss of information: closed multidimensional sequential patterns. This representation introduces properties that allow to deeply prune the search space. In this paper, we also define algorithms that do not require candidate set maintenance. Experiments on synthetic and real data are reported and emphasize the interest of our proposal.

Keywords data mining · datawarehouses · closed multidimensional sequential patterns

1 Introduction

Data warehouses are now well spread over companies. They contain valuable information that can easily be queried and visualized with the OLAP tools, provided the fact that the user is able to design on-line his own queries. However, it is still challenging to provide the user with tools that can automatically extract relevant knowledge from such huge amounts of data. Data warehouses are indeed different from usually mined

Marc Plantevit, Anne Laurent, Maguelonne Teisseire
LIRMM, UniversitéMontpellier 2, CNRS,
161 Rue Ada 34392 Montpellier, France
E-mail: {plantevi,laurent, teisseire}@lirmm.fr

databases as they contain aggregated data, described by means of several dimensions that can possibly be organized through hierarchies. In this paper, we thus try and extend existing methods that are now recognized for mining classical databases to this framework. As data are historized, we argue that sequential patterns are well-suited to this task. Sequential patterns have been studied for more than ten years [1], with a lot of research and industrial applications (*e.g.* user behavior, web log analysis, discovery of patterns from proteins' sequences, security and music). Algorithms have been proposed, based on the APriori-based framework [23],[9] ,[2], or on other approaches [11],[8]. Sequential patterns have recently been extended to multidimensional sequential patterns by [14], [15], and [22]. They aim at discovering patterns that take time into account and that involve several dimensions. For instance in [15], rules like *A customer who bought a surfboard together with a bag in NY later bought a wetsuit in SF* are discovered. Note that such sequences can also contain a wild-card item $*$ instead of a dimension value. For instance, considering the previous example, if there is no frequent pattern in the database describing the fact that wetsuits were later bought in SF, or NY etc, but there are numerous wetsuits bought whatever the city, then the rule *A customer who bought a surfboard together with a bag in NY later bought a wetsuit* will be mined, represented as $\langle(surfboard, NY)(wetsuit, *)\rangle$.

Sequential patterns are usually extracted from the simple schema: (*e.g. products, customer_id* and *date*) but the number of mined patterns can be very huge. This is why condensed representations were proposed for the itemset framework ([10],[13],[24],[5]) and for sequential patterns ([21],[18]). In both cases, the approaches allow a condensed representation and a pruning strategy in the search space.

However, these works are not suitable for multidimensional sequential patterns because they only consider a particular case for candidate generation. In our context, a super-sequence may indeed result from several cases (1) a longer sequence (more items) or (2) a more general sequence based on the relation between dimension values and the wild-card value $*$ (more general items).

The main contributions of this paper are a theoretical framework for mining closed multidimensional sequential patterns and some algorithms called *CMSP* (Closed Multidimensional Sequential Pattern mining) to mine such patterns. When considering multidimensional data, the number of possible patterns is combinatorially explosive and the generate-and-prune methods are no more scalable for long sequences, as highlighted in [12],[16]. This is why we adopt the pattern growth paradigm ([11]) to propose a greedy approach for mining frequent sequences without candidate generation.

The paper is organized as follows. First we recall the related work in Section 2 and we detail why existing works are not suitable for mining data warehouse. Then we present the core of our proposition: the definitions are introduced in Section 3 while the $CMSP$ algorithms are detailed in Section 4. Experiments on synthetic and real data, reported in Section 5, show that our method performs well both on runtime and number of extracted closed sequences. Finally, we provide some concluding remarks and suggestions for future work in Section 6.

2 Related Work

In this Section, we first recall from [15] the seminal definitions of multidimensional sequential patterns. Then we present existing works from the literature on closed patterns.

2.1 Multidimensional Sequential Patterns

To the best of our knowledge, three proposals exist for mining sequential patterns when considering several dimensions.

The approach proposed by [14] mines sequences over one single dimension (*e.g.* product) labeled by multidimensional patterns. However, no combination of dimensions is possible through time (within the sequence).

[22] is very particular since the dimensions are embedded one within the other one (web *pages* are visited within one *session* during one *day*).

In [15], rules not only combine several dimensions but they also combine these dimensions over time. For instance in the rule *A customer who bought a surfboard together with a bag in NY later bought a wetsuit in SF*, NY appears before SF, and $surfboard$ appears before $wetsuit$.

As the last approach is more general than the other two ones, we focus here on the concepts of multidimensional sequential patterns introduced in [15].

More formally, let us consider a database DB where data are described with respect to n dimensions. We consider a 3-bin partitioning of the dimensions:

- the set of those dimensions that will be contained within the rules (*analysis dimensions*) is denoted by D_A;
- the set of those dimensions on which the counting will be based (*reference dimensions*) is denoted by D_R;
- the set of those dimensions that are meant to introduce an order between events (*e.g. time*) is denoted by D_T.

The database can then be partitioned into *blocks* defined by their positions on the reference dimensions.

A *multidimensional item* e is a m-tuple defined over the set of the m D_A dimensions. We consider $e = (d_1, d_2, \ldots, d_m)$ where $d_i \in Dom(D_i) \cup \{*\}, \forall D_i \in D_A$ and where $*$ stands for the *wild-card* value. For instance, $(1, 2)$ is a multidimensional item defined with respect to two dimensions.

A *multidimensional itemset* $i = \{e_1, \ldots, e_k\}$ is a non-empty set of multidimensional items. All items of the itemset have to be "incomparable" to preserve the notion of set. For instance, $\{(1, 1), (1, 2)\}$ is a multidimensional itemset whereas $\{(1, 1)(1, *)\}$ is not an itemset because $(1, 1) \subseteq (1, *)$.

A *multidimensional sequence* $\varsigma = \langle i_1, \ldots, i_l \rangle$ is a non-empty ordered list of multidimensional itemsets. For instance, $\varsigma_1 = \langle \{(1, 1), (1, 2)\}\{(1, *), (*, 4)\}\{(1, 3)\} \rangle$ is a multidimensional sequence.

Sequences that contain k items and g itemsets are called g-k-sequences:

Definition 1 (g-k-Sequence) *A g-k-sequence S is a sequence that is composed of g itemsets and k items as following:*
$S = \langle \{e_1^1, e_2^1, \ldots, e_{k_1}^1\}, \{e_1^2, e_2^2, \ldots, e_{k_2}^2\}, \ldots, \{e_1^g, e_2^g, \ldots, e_{k_g}^g\} \rangle$ *where* $\sum_1^g (k_i) = k$.

A multidimensional sequence can be included into another one:

Definition 2 (Sequence inclusion) *A multidimensional sequence $\varsigma = \langle a_1, \ldots, a_l \rangle$ is said to be a subsequence of $\varsigma' = \langle b_1, \ldots, b_{l'} \rangle$ if there exist integers $1 \leq j_1 \leq j_2 \leq \ldots \leq j_l \leq l'$ such that $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \ldots, a_l \subseteq b_{j_l}$.*

For instance, the sequence $\langle \{(1, 2), (*, 3)\}\{(2, 2)\} \rangle$ is a subsequence of the sequence $\langle \{(1, *), (*, 3)\}\{(*, 2)(*, 3)\} \rangle$.

We consider that each block defined over D_R contains one multidimensional data sequence, which is thus identified by that block. A block *supports* a sequence ς if all items of ς can be matched (each value on each analysis dimension has to be matched except with wild-card value *) with an item of the data sequence with respect to time order.

Thus, the *support* of a multidimensional sequence is the number of those blocks defined over D_R which contain this sequence.
When considering the *classical* case of sequential patterns, the sets of analysis, reference, and order dimensions consist of only one dimension (usually the *product*, *customer_id* and *time* dimensions). Note that even in this classical case, the number of frequent sequential patterns discovered from a database can be huge. And this problem becomes worse in the case of multidimensional patterns since multidimensional framework produces more patterns than classical framework.

For this reason, it is necessary to study condensed representations. We aim at representing the sequential patterns *and* their support. Thus, we do not consider the solution of representing *only* maximal sequential patterns, since this would result in the loss of the information about the support of subsequences.

In this paper, we consider *closed patterns*. They indeed allow to represent the patterns in a compressed manner without any loss of information.

2.2 Closed Patterns

A major challenge in mining frequent patterns is the fact that such mining often generates a huge number of patterns satisfying the minimum support threshold. This is because if a pattern is frequent, each of its subpatterns is also frequent. A pattern will contain an exponential number of smaller subpatterns. To overcome this problem, closed frequent pattern mining were proposed by [10]. Frequent closed patterns algorithms, which heavily draw on Formal Concept Analysis (FCA) mathematical settings [6,20], present a novel alternative with a clear promise to dramatically reduce, without information loss, the size of the frequent pattern set. From the frequent closed patterns, the support of frequent non-closed patterns can be inferred.

Based on the previous approaches [5],[10],[13],[24], a closed pattern is defined as follows:

Definition 3 (Closed Pattern) *A (sequential) pattern α is* closed *if there does not exist any pattern β such that $\alpha \subseteq \beta$ and $support(\alpha) = support(\beta)$.*

For instance, let us suppose that the sequences $\langle a \rangle_3$ and $\langle a, b, c \rangle_2$ are closed, where $\langle a \rangle_3$ stands for the sequence $\langle a \rangle$ having support 3. Then it can be deduced that the sequences $\langle a, c \rangle$ and $\langle b, c \rangle$, which are included in the previous ones, have support 2. If their support was 3 then the sequence $\langle a \rangle$ would not be closed. Thus, their frequency is directly correlated to the frequency of $\langle a, b, c \rangle$. All the sequences are shown in Figure 1 where the closed sequences are circled.

It should be noticed that many works have been done on the extraction of closed itemsets [5,13,24,19] but only two approaches have been proposed for sequential closed patterns: BIDE and CloSpan.

CloSpan [21] first extracts a set of closed sequence candidates, which contains the set of frequent sequences. In this first set, some sequences are not closed. Thus,

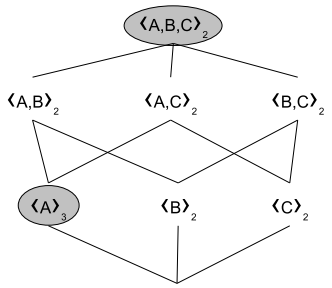


Fig. 1 Searching all the frequent sequences from the closed ones

CloSpan prunes the non-closed sequences in a second step. CloSpan uses the pattern-growth approach, which is different from the APriori-like candidate-generation-and-test approaches. CloSpan decomposes the database in order to discard non-needed computations [12]. In order to reduce the space search, the database is projected by the current mined sequence also called prefix sequence. The projected database according to the sequence α is denoted $DB|\alpha$. For instance, given the data sequence $S = \langle(abcd)\text{-}ea(bc)(ac)\rangle$ and $\alpha = \langle(ab)a\rangle$, $S|\alpha = \langle(bc)(ac)\rangle$. As soon as a sub-pattern or a super-pattern of the current sequence shares the same projected database as the current one, we do not need to explore this database in order to grow this sequence. Indeed, their subtrees can be merged into one without having to mine a subtree in the search space already mined.

BIDE [18] enhances the previous approach (CloSpan). The authors propose an approach without any candidate maintenance-and-test paradigm. This approach prunes the search space more deeply. It checks a pattern closure in a more efficient way while consuming less memory than CloSpan. Indeed, BIDE does not maintain the set of historic closed patterns.

It should be noted that these two approaches not only reduce the number of sequences to be considered, but also result in better performances (both in time and memory).

We can also cite the work of [17] which considers closed sequential patterns in multidimensional framework. However, this approach uses a condensed representation of [14]. So this approach mines sequences over one single dimension (*e.g.* product) labeled by multidimensional patterns. Thus, no combination of dimensions is possible through time (within the sequence).

OLAP Mining has been studied since 1997 [7], aiming at designing methods to automatically extract relevant knowledge from multidimensional databases. In this framework, the challenges are numerous, as this kind of data is different from classical databases that are usually mined. First, multidimensional databases contain aggregated data. Moreover, this data is described using dimensions, which can be organized through hierarchies. Finally, the data is historized so as to report the evolution through time. Discovering trends from such data can thus be seen as the task of extracting the relevant frequent sequences that occur. This is the reason why we choose here to study sequential patterns, as efficient methods have been designed to extract such trends, while remaining scalable (typically by using algorithms that do not have to manage sets of candidates) and concise (typically by using closed sequential patterns). However to the best of our knowledge, multidimensional sequential patterns have not been con-

sidered in the existing work. So, we propose a theoretical framework for mining closed multidimensional sequential patterns.

2.3 Particularity of the Multidimensional Framework

In this section, we explain why it is quite difficult to apply existing work on sequential pattern to the multidimensional framework. These reasons are essentially due to the inclusion of wild-carded -sequences.

Unfortunately, the classical framework (one analysis dimension) is no longer suitable when considering wild-carded multidimensional sequential patterns. For instance, if we assume that $\langle\{(a_1, b_1), (*, b_2)\}\rangle_2$ is closed, then by calculating the other sequences, it could be calculated that the sequences $\langle\{(a_1, b_1), (a_1, b_2)\}\rangle$ and $\langle\{(a_1, b_1), (a_2, b_2)\}\rangle$ are in the same case as previously (support should be 2). However, it may happen that the sequences $\langle\{(a_1, b_1), (a_1, b_2)\}\rangle$ and $\langle\{(a_1, b_1), (a_2, b_2)\}\rangle$ have a support of 1. This may occur because a super-sequence may result from several cases: a longer sequence (more items) or a more general sequence based on the relation between dimension values and the wild-card (more general items). More precisely, it is due to the fact that all values of domain dimensions are contained in wild-card.

3 CMSP - Closed Multidimensional Sequential Patterns

In order to define Closed Multidimensional Sequential Patterns, we introduce a specialization relation between patterns to catch the specific context of the multidimensional data.

Definition 4 (Specialization/Generalization) *A multidimensional sequential pattern $\alpha = \langle a_1, \ldots, a_l\rangle$ is more general than $\beta = \langle b_1, \ldots, b_{l'}\rangle$ ($l \leq l'$) (and β is more specific than α) if there exist integers $1 \leq j_1 \leq j_2 \leq \ldots \leq j_l \leq l'$ such that $b_{j_1} \subseteq a_1, b_{j_2} \subseteq a_2, \ldots, b_{j_l} \subseteq a_l$.*

If β is more specific than α, we write $\alpha \subset_S \beta$ where $\subset_S$ denotes a *specialization relation*.

Example 1 *The sequence $\beta = \langle\{(a_1, b_1), (a_2, b_2)\}\{(*, b_1)\}\rangle$ is more specific than the sequence $\alpha = \langle\{(a_1, *), (a_2, b_2)\}\rangle$. We denote $\alpha \subset_S \beta$. We can note that this definition is different from the inclusion definition (definition 2). Here, the main idea is that a sequence is more specific than another one if it is at least longer than the other one and its items should be more specific. For instance, $\langle(1, *)\rangle \subset_S \langle(1, 1), (1, 1)\rangle$ and $\langle(1, *)\rangle \not\subseteq \langle(1, 1), (1, 1)\rangle$.*

We can now define closed multidimensional sequence and closed multidimensional sequential patterns as follows:

Definition 5 (Closed Multidimensional Sequence) *A multidimensional sequence α is closed if there does not exist β such that $\alpha \subset_S \beta$ and $support(\alpha) = support(\beta)$*

Definition 6 (Closed Multidimensional Sequential Pattern) *Let minsup be a minimal support threshold, a sequence s is a closed multidimensional sequential pattern if s is closed and $support(s) \geq minsup$.*

Example 2 (Closed Multidimensional Sequential Patterns and Inference) *With a minimal support threshold equal to* 3 *and the database shown in Table 1, the set of closed patterns is given by the first part of Table 2.*

With a minimal support threshold equal to 2*, the set of closed sequential patterns is given by Table 2. Unclosed sequences can be inferred from the closed ones. For instance, the support of the sequences* $\langle(*, b_2)\rangle$ *and* $\langle(a_2, *)\rangle$ *can then be computed as being equal to* 3*.*

Thus, two levels of knowledge can be inferred from closed patterns:

1. for subsequences containing fewer items;
2. for subsequences containing more general items (more wildcards).

These two levels can be mixed in order to infer even more general knowledge.

1	$\langle\{(a_1, b_1), (a_1, b_2)\}\{(a_2, b_2)\}\{(a_1, b_3)\}\{(a_1, b_2)(a_2, b_2)\}\rangle$
2	$\langle\{(a_1, b_2), (a_2, b_1)\}\{(a_3, b_2)\}\{(a_2, b_1)\}\{(a_2, b_1)\}\rangle$
3	$\langle\{(a_4, b_4)\}\{(a_2, b_1)\}\{(a_1, b_1)(a_2, b_2))\}\rangle$

Table 1 Running Example

$\langle(a_1, *)\rangle_3$
$\langle(a_2, *), (a_2, *)\rangle_3$
$\langle(a_2, *), (*, b_2)\rangle_3$
$\langle(*, b_1), (a_2, *)\rangle_3$
$\langle(*, b_2), (*, b_2)\rangle_3$
$\langle\{(a_1, b_1), (*, b_2)\}\rangle_2$
$\langle\{(a_1, b_2), (*, b_1)\}, (a_2, *), (a_2, *)\rangle_2$
$\langle\{(a_1, b_2), (*, b_1)\}(*, b_2), (a_2, *)\rangle_2$
$\langle(a_2, b_1), (a_2, *)\rangle_2$
$\langle(a_2, b_1), (*, b_1)\rangle_2$
$\langle(a_2, b_1), (*, b_2)\rangle_2$
$\langle(a_2, *), (2, 2)\rangle_2$
$\langle(a_2, *), (1, *)\rangle_2$
$\langle(*, b_1), (a_2, b_2)\rangle_2$
$\langle(*, b_1), (a_1, *)\rangle_2$

Table 2 Closed multidimensional sequences with support 2 and 3

4 *CMSP*: Mining Closed Multidimensional Sequential Patterns

As mentioned in the introduction, generate-and-prune methods cannot be scalable for long sequences. This non-scalability problem is worse in a multidimensional framework since the number of possible patterns is combinatorially explosive.Thus, we adopt the pattern-growth paradigm in order to define complete and scalable algorithms for mining multidimensional closed sequential patterns. The definitions we consider here are taken from the approaches that are recognized as being efficient for mining for sequential patterns. The first approach is described in Section 4.2. It considers the methods defined

in CloSpan [21]. The second approach we consider here is based on Bide [18] described in Section 4.3. In this approach, no candidate set is maintained.

We define below the definitions that are common to these two approaches, and we then detail on each of them the implementation we propose. Experimental results are reported in Section 5.

4.1 Order Within The Itemset of a Sequence

Ordering items within the itemsets is one of the main basis to improve the implementation and to discard already examined cases. The existing methods presented in [21] and [18] for ordering sequences are not directly applicable to the multidimensional framework. Indeed wild-carded items are not explicitly present in databases. Such items are retrieved by inference since there is no associated tuple in the database.

1	$\langle\{(a_1, b_1), (a_1, b_2)\}\rangle$
2	$\langle\{(a_1, b_2), (a_2, b_1)\}\rangle$

Table 3 Where is the sequence $\langle\{(a_1, b_2), (*, b_1)\}\rangle$?

Table 3 shows an example of a database that cannot be treated by these two methods, since wild-carded items are not explicitly present in the database. Thus, no total lexicographic order can be defined between the elements of the itemsets. So these methods cannot mine the sequence $\langle\{(a_1, b_2), (*, b_1)\}\rangle$. As an example, CloSpan finds the frequent item (a_1, b_2) with a support of 2 and then, it constructs the projected database prefixed by the sequence $\langle\{(a_1, b_2)\}\rangle$. This projected database contains the sequences $\langle\{\}\rangle$ and $\langle\{(a_2, b_1)\}\rangle$. Thus the item $(*, b_1)$ does not appear to be frequent in this projected database whereas it is frequent in the initial database.

This trivial example highlights the need to introduce a lexical order taking wild-carded items into account.

It should be noted that it is not possible to extend the whole database with all the possible wild-carded items before the mining process. For example, if we consider a database containing m analysis dimensions and n_i items in an itemset i, this transformation would produce $(2^m - 1) \times n_i$ items instead of n_i leading to a database of size $(2^m - 1) \sum_{t_i \in DB} n_{t_i}$.

It is then necessary to take all wild-carded items into account *during* the process of closed multidimensional sequential patterns and not *before* as a pre-treatment.

We will then introduce an lexical order and functions to locally materialize the wild-carded items.

It is necessary to have a lexicographic order when mining frequent patterns since it is the basis foundation of the non-duplication of items during the computation. We now define the concept of *extended itemset*.

Definition 7 (Extended Itemset) *A frequent itemset is said to be* extended *if it is equal to its downward closure according to the specialization relation ($\subset_S$).*

The extended itemsets enable to mine wild-carded items which can be deduced from data sequences.

In order to optimize the computation of closed multidimensional sequential patterns, we introduce a *lexico-graphico-specific* (LGS) order. This order results from an

alpha-numerical order according to the precision of the items (number of * in the item). Thus the priority is given to the most specific items during the mining process. For instance, itemset $i_1 = \{(a_1, b_1), (a_2, b_1), (a_1, *), (a_2, *), (*, b_1)\}$ is sorted w.r.t *LGS* order.

We define a function which transforms an itemset (transaction) into its extended itemset.

Definition 8 (Function LGS-Closure) *LGS-Closure is an application from an itemset i to the closure of i w.r.t. the LGS order* $<_{lgs}$.

Example 3 (LGS-Closure) *LGS-Closure*$(\{(a_1, b_1), (a_2, b_1)\}) = \{(a_1, b_1), (a_2, b_1),$-$(a_1, *), (a_2, *), (*, b_1)\}$

This closure is illustrated in the Figure 5. Note that we do not return the most general item (,*) of the lattice . This item does not need to be mined since it is a tautology.*

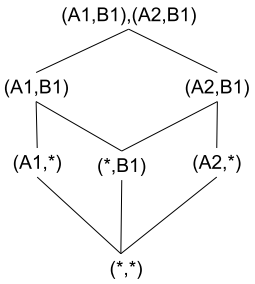


Fig. 2 LGS-Closure

The extraction of frequent items is then performed on each extended itemset. In the pattern-growth approach, the sequences are greedily extracted by adding a frequent item to a frequent *prefix sequence*. The prefix sequence can be extended by adding a frequent item in a new itemset or by adding a frequent item in the last itemset of the prefix sequence. It is thus necessary to define an efficient way to extend a prefix sequence according to its last itemset. Furthermore, we have to preserve the notion of set of an itemset (two comparable items cannot appear together within an itemset). For that purpose, we define a restriction as follows.

Definition 9 (Function LGS-Closure$_X$) *The function LGS-Closure*$_X(i)$ *in an application from an itemset i to the closure of i taking filtering of the itemset* $X = \{x_1 \leq_{lgs} \ldots \leq_{lgs} x_{k'}\}$ *into account such that:*

$$\textit{LGS-Closure}_X(i) = \{e \in \textit{LGS-Closure}(i) \textit{ s.t. } e \geq_{lgs} x_{k'} \textit{ and } \nexists x_j \in X \mid x_j \subseteq e\}$$

Example 4 (LGS-Closure$_X$) *LGS-Closure*$_{\{(a_1,b_1)\}}(\{(a_1, b_1), (a_1, b_2), (a_1, b_3)\}) = \{(*, b_2), (*, b_3)\}$

4.2 $CMSP\_Cand$

In this section, we define an algorithm adapted from CloSpan.

Closed sequential patterns are extracted using algorithms 1 ($CMSP\_Cand$) and 2 (SequenceGrowing) following a depth-first strategy.

Instead of scanning the whole database level by level in the same way as APriori-like methods, the database is projected according to the current examined prefix sequence. This projection is quite different from [11]. Since we should take into account wild-carded items, the database projection should take into account the transaction (itemset of the block data sequence) where item was found, and not only the item itself like in [11]. To take into account this transaction, we use the *LGS-Closure* function by filtering the already found items.

For instance, if we consider the following data sequence $S = \langle (1,1), \{(1,2),(1,3)\}, (2,2)\rangle$ and the current prefix sequence $\alpha = \langle (1,2)\rangle$. According to [11], the projected database $S|\alpha = \{\_(1,3)\}, (2,2)\rangle$ where $\_(1,3)$ indicates that the item $(1,3)$ and the last itemset of α shares the same itemset in the data sequence. With our algorithm, the projected database is quite different since we take into account wild-card values. The projected database is built as follows: $S|\alpha = \_LGS\text{-}Closure_{\{(1,2)\}}(\{(1,2)(1,3)\}), \{(2,2)\}$, thus we have: $S|\alpha = \{\_(1,3)\_(1,*)\_(*,3)\}, \{(2,2)\}$.

The use of projected database prevents from scanning already seen data. Indeed, if we consider a frequent sequence α and the current examined prefix sequence β such that $\beta \subseteq_S \alpha$ or $\alpha \subseteq_S \beta$ and such that the projected database is the same for β and α, then it is not necessary to expand this last sequence. We just need to copy the subtree (already mined) of the sequence α to the sequence β.

We can note that:

- if $\alpha \subseteq_S \beta$ then α cannot be closed;
- if $\beta \subseteq_S \alpha$ then the sequences prefixed by β are already known, thus allowing us to discard the frequent suffixes of β.

In the latter case, it should be noted that β cannot be closed. However, it is necessary to keep this sequence as it can be included in some other ones, thus avoiding database scans.

Algorithm 3 considers locally frequent items from projected fragments of the database. It is based on the *LGS-Closure* function (definition 9). The projected database is scanned only once in order to extract all frequent items. Two types of items can be mined:

1. The items which cannot be included in the last itemset of the prefix sequence ς. These items should be included in a new itemset of ς. In order to mine such items and to take into account wild-carded item, we need to extend all transactions of the projected database (step by step) thanks to the function *LGS-Closure*.
2. The items which can be included in the last itemset of the prefix sequence ς. In that case, we use the function *LGS-Closure* parametrized with the last itemset of ς.

The last task of Algorithm 1 is to eliminate non-closed multidimensional sequences from the set of closed sequence candidates. The problem is to check out for each multidimensional sequence ς, whether there exists a multidimensional sequence ς' such that $\varsigma \subset_S \varsigma'$ and $support(\varsigma) = support(\varsigma')$. A naive algorithm, which compares each multidimensional sequence with other ones in the set, cannot work because of its quadratic

complexity in the number of closed sequence candidates. We adopt the fast subsumption checking algorithm introduced by Zaki [24]. The value of support is very dense. Thus *support* cannot be a relevant hash key which enables a sparse distribution of keys. [24] proposes using the sum of sequences' identifiers (denoted $\tau(D_S)$)as its hash key instead of using support. However, in sequence framework, the equivalence of $\tau(D_S)$ does not imply the equivalence of support. Thus, for the multidimensional sequences that share the same $\tau(D_S)$, we need to check their support in order to eliminate the invalid candidates. This hash key, also used in CloSpan, is easy to compute. Furthermore, it enhances the space search reduction. Thus, the complexity of this operation is $\Theta(\sum n_{\tau_i}^2)$ where n_{τ_i} is the number of closed sequence candidates that share the same τ_i. The n_{τ_i} are significantly less than the total number of closed sequence candidates $(\sum n_{\tau_i})$.

Algorithm 1: $CMSP\_Cand$

```
Data: Database DB, minimal support minsup
Result: The set of closed C
begin
    /* Initialization */
    Set L ← {}
    Set C ← {}
    Sequence α ← ⟨⟩
    /*Frequent sequence mining (depthfirst)*/

    SequenceGrowing(α, DB, L, minsup)
    /*Pruning of non-closed in L*/
    foreach s1 ∈ L do
        foreach s2 ∈ L do
            if s1 ⊆_S s2 et support(s1) = support(s2) then
                delete(s1, L)
    C ← L
    return C
end
```

Algorithm 2: SequenceGrowing: Mining algorithm

```
Data: Sequence α,projected database DB|α, closed sequence candidate set L , minimal support minsup
Result: The set of sequences prefixed by α
begin
    /*α may be closed*/
    insert(α, L);
    /*Check if the sequence was already checked out */
    if ∃β | (α ⊆_S β or β ⊆ α) and they both share the same projected database then
        Copy the descendants of β in α;
        return
    Set F_l ← getFrequentItems(DB|α, minsup);
    foreach α' ← α.b do
        Build DB|α';
        SequenceGrowing(α', DB|α', L, minsup);
end
```

Algorithm 3: getFrequentItems: Localy frequent item mining

```
Data: Projected database DB|α, minsup
Result: localy frequent item set F_l
begin
    /*We assume that for each data sequence S_i of DB|α we have: S_i =
    LGS-Closure_{lastItemset(α)}(same).otherTrans
    We should examine all the data sequences of DB*/
    foreach S_i ∈ DB|α do
        foreach item _e in same do
            handle _e ;
        foreach itemset is in other do
            /*Search all items which could be inserted inyo a further itemset of α*/
            SearchOtherTransFrequentItem e in LGS-Closure(is);
            /*Search all items which could be inserted into the last itemset of α*/
            if is supports lastItemset(α) then
                SearchSameTransFrequentItem _e in LGS-Closure_{lastItemset(α)}(is);
    return (F_l = {e|support(e) ≥ suppmin});
end
```

Example

Let us consider the multidimensional sequence database from Tab. 4 . We want to discover all closed multidimensional sequential patterns with *minsup* equal to 2.

B_1	$\langle\{(a_1, b_2, c_3)\}\{(a_1, b_1, c_1), (a_1, b_3, c_2)\}\rangle$
B_2	$\langle\{(a_1, b_2, c_2)(a_1, b_2, c_3)\}\{(a_1, b_1, c_1), (a_1, b_3, c_4)\}\{(a_1, b_1, c_1)\}\rangle$

Table 4 Multidimensional Sequence Database DB

The main algorithm **CMSP_Cand** calls routine **SequenceGrowing** with the empty sequence $\langle\rangle$, DB and $minsup = 2$ as parameters.

The first step aims at discovering all the frequent items on DB thanks to routine **getFrequentItems**:

$\{(a_1, b_1, c_1)_2, (a_1, b_2, c_3)_2, (a_1, b_1, *)_2, (a_1, b_2, *)_2, (a_1, b_3, *)_2, (a_1, *, c_1)_2, (a_1, *, c_2)_2,$
$(a_1, *, c_3)_2, (*, b_1, c_1)_2,$
$(*, b_2, c_3)_2, (a_1, *, *)_2, (*, b_1, *)_2, (*, b_2, *)_2, (*, b_3, *)_2, (*, *, c_1)_2, (*, *, c_2)_2, (*, *, c_3)_2\}$

The sequences are mined with a depth-first strategy according to the order LGS.

The prefix sequence $\langle(a_1, b_1, c_1)\rangle$ is examined. All sequences with prefix $\langle(a_1, b_1, c_1)\rangle$ are searched on $DB|\langle(a_1, b_1, c_1)\rangle$.

Sequences $\langle\{(a_1, b_1, c_1), (a_1, b_3, *)\}\rangle$ and $\langle\{(a_1, b_1, c_1), (*, b_3, *)\}\rangle$ are discovered. There is no frequent item on the projected database according to $\langle\{(a_1, b_1, c_1), (a_1, b_3, *)\}\rangle$ et $\langle\{(a_1, b_1, c_1), (*, b_3, *)\}\rangle$.

The discovery of frequent sequences continue with the examination of the prefix sequence $\langle(a_1, b_2, c_3)\rangle$.

When prefix sequence $\langle\{(a_1, b_2, c_3)\}\{(a_1, b_1, c_1)\}\rangle$ is considered. Algorithm SequenceGrowing detects that $\langle\{(a_1, b_2, c_3)\}\{(a_1, b_1, c_1)\}\rangle$ and $\langle(a_1, b_1, c_1)\rangle$ share the same projected database. Thus, the exploration of the prefix sequence $\langle\{(a_1, b_2, c_3)\}\{(a_1, b_1, c_1)\}\rangle$

can be stopped. Indeed, sequences $\langle\{(a_1, b_2, c_3)\}, \{(a_1, b_1, c_1), (a_1, b_3, *)\}\rangle$ and $\langle\{(a_1, b_2, c_3)\}$-$\{(a_1, b_1, c_1), (*, b_3, *)\}\rangle$ are discovered without scanning the projected database $DB_{\langle(a_1,b_1,c_1)\rangle}$.

The discovery of frequent sequences goes on with the examination of the prefix sequence $\langle(a_1, b_1, *)\rangle$.

The process is iterated untill the extraction of all the sequences with prefix equal to $\langle(*, *, c_3)\rangle$.

Finally, closed sequences are retrieved from the closed candidates set. Only sequences $\langle\{(a_1, b_2, c_3)\}\{(a_1, b_1, c_1), (a_1, b_3, *)\}\rangle$ and $\langle\{(a_1, *, b_2)\}\rangle$ are closed.

4.3 *CMSP_Free*: Mining CLosed Multidimensional Sequential Patterns Without Candidate Set Maintenance

The previous algorithm, *CMSP_Cand* Algorithm 1, needs to maintain a set of potentially closed sequences (set L). In post-processing, it has to compute closed sequences among candidate sequences of this set. This maintenance is expensive (quadratic in the size of the set) even if optimization techniques allow to reduce this cost. In this section, we propose an algorithm without candidate-set-maintenance. This approach is based on Bide [18]. We first detail some preliminar definitions (Sequence extensions) then we present the associated algorithm.

4.3.1 Sequence Extensions

According to the definition of closed multidimensional sequential pattern, if a *g-k*-sequence $S = s_1, \ldots, s_g$ is not closed then there exists a sequence S' with the same support of S such that $S \subset_S S'$. Definition 10 enumerates the five possible ways to have a more specific sequence from a *g-k*-sequence.

Definition 10 *A more specific sequence can be built in five different ways from a g-k-sequence* $\langle s_1, s_2, \ldots, s_g\rangle$*:*

- inter itemset *forward extension* $S' = \langle s_1, s_2, \ldots, s_g, \{e'\}\rangle$*;*
- intra itemset *forward extension* $S' = \langle s_1, s_2, \ldots s_g \cup \{e'\}\rangle$*;*
- inter itemset *backward extension* $S' = \langle s_1, s_2, \ldots, s_i, \{e'\}, s_{i+1}, \ldots, s_g\rangle$*;*
- intra itemset *backward extension* $S' = \langle s_1, s_2, \ldots s_i \cup \{e'\}, s_{i+1}, \ldots, s_g\rangle$*;*
- specialization *of an item* $\exists i \in \{1, \ldots, g\}, \exists e, \exists e'$ *s.t.* $e \subset_S e'$: $S' = \langle s_1, s_2, \ldots, s_{i-1}$,-$s_i[e'/e], s_{i+1}, \ldots, s_g\rangle$ *where* $s_i[e'/e]$ *is the substitution of* e *by* e' *in* s_i*.*

We will notice that the last point can easily be detected thanks to the order of sequence we consider as soon as the four ones (extensions) are detected.

Theorem 1 (Bi-Directional Extension) *A sequence S is closed if and only if there does not exist neither any backward or forward extension nor any specialization that preserves the support of S.*
The proof trivially comes from the definition of closed multidimensional sequential patterns.

In order to check if a sequence is closed, we have to check if there exists some backward or forward extension or specialization of an item, thanks to theorem 1.

It is easy to find the forward extensions of a sequence:

Lemma 1 (Forward Extension) *Let S be a sequence, the complete set of forward extensions of S is equivalent to the set of locally frequent items on the projected database according to S such that their support is equal to support(S).*

Proof Locally frequent items are discovered by scanning the projected database according to prefix sequence S_p. Since each event occurs during or after the prefix sequence S_p, if an item e occurs in all data sequence from the projected database, then e is a forward extension.

All events (items) that occur after the first instance of the sequence S_p is included in the projected database $DB|_{S_p}$, which means that the complete set of forward extensions can be extracted by scanning the projected database DB_{S_p}.

It is harder to detect the backward extensions of a sequence than forward extensions. There are two types of backward extension:

- $S' = s_1, s_2, \ldots, s_i, \{e'\}, s_{i+1}, \ldots, s_g$
- $S' = s_1, s_2, \ldots s_i \cup \{e'\}, s_{i+1}, \ldots, s_g$

An item can be inserted in a new itemset between two itemsets s_i and s_{i+1} (inter itemset backward extension). An item can also be inserted in an already defined itemset.

As a sequence can appear several times within a block data sequence, we can identify g intervals to localize the potential backward extension of a g-k sequence.

Figure Fig. 3 describes the g intervals of a multidimensional prefix sequence in a multidimensional data sequence.

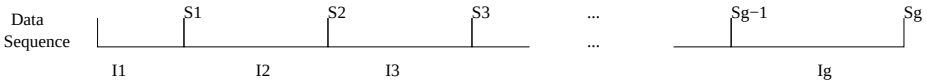


Fig. 3 The g intervals that may contained backward extensions of prefix g-k-sequence $S_p = \langle s_1, s_2, \ldots, s_g \rangle$ in a block data sequence

We have to maximize these intervals in order to detect all backward extension items.

Definition 11 (i^{th} Maximal Interval) *Let $S_p = \langle s_1, s_2, \ldots, s_g \rangle$ be a g-k prefix sequence and S be a block data sequence, the i^{th} maximal interval of S according to S_p is defined as follow:*

- *if $i = 1$: the subsequence from the beginning of S to $la(s_1)$ excluded where $la(s_1)$ is the last appearance of the itemset s_1 in S such that $la(s_1) < la(s_2) < \ldots < la(g)$.*
- *if $1 < i \leq g$: the subsequence between the first appearance of the sequence $\langle s_1, s_2, \ldots, s_{i-1} \rangle$ (denoted by $fa(\langle s_1, s_2, \ldots, s_{i-1} \rangle)$) and $la(s_i)$ excluded such that $la(s_i) < la(s_{i+1}) < \ldots < la(g)$.*

As an example, the first maximal interval of the sequence $s = \langle \{(1,1)\}\{(1,2)\} \rangle$ in the data sequence $S = \langle \{(1,1)\}\{(1,2)\}\{(1,1)\}\{(1,2)\}\{(1,1)\}\{(1,2)\} \rangle$ is the subsequence $\langle \{(1,1)\}\{(1,2)\}\{(1,1)\}\{(1,2)\} \rangle$. The second maximal interval of s in S is $\langle \{(1,2)\}\{(1,1)\}\{(1,2)\}\{(1,1)\} \rangle$.

Lemma 2 (Backward Extension Checking) *Let $S_p = \langle s_1, s_2, \ldots, s_g \rangle$ be a g-k prefix sequence, if there exists an item e that appears in each i^{th} maximal intervals of S_p in DB, then e is a backward extension.*

Otherwise, if there is no item e that appears in each i^{th} maximal interval of S_p in DB, then S_p cannot have a backward extension.

Proof Let $S_p = \langle s_1, s_2, \ldots, s_g \rangle$ be a g-k-prefix sequence, if there exists an item e that appears in each i^{th} maximal interval of S_p in DB (each i^{th} maximal interval of S_p in each data sequence of DB), then we can build the sequence $S'_p = \langle s_1, s_2, \ldots, s_{i-1} \cup \{e\}, s_i, \ldots, s_g \rangle$ or $S' = \langle s_1, s_2, \ldots, s_{i-1}, \{e\}, s_i, \ldots, s_g \rangle$ such that $S_p \subset_S S'_p$ and $support(S'_p) = support(S_p)$. Thus, e is a backward extension of S_p in DB.

Suppose that there exists a sequence $S'_p = \langle s_1, s_2, \ldots, s_{i-1} \cup \{e\}, s_i, \ldots, s_g \rangle$ or $S'_p = \langle s_1, s_2, \ldots, s_{i-1}, \{e\}, s_i, \ldots, s_g \rangle$ such that $S_p \subset_S S'_p$ and $support(S'_p) = support(S_p)$. In each data sequence of DB that contain S_p, item e must appear after the first appearance of $\langle s_1, \ldots, s_{i-1} \rangle$ and before the last appearance of the subsequence $\langle s_i, \ldots, s_g \rangle$. That means that e must appear in each i^{th} maximal interval of S_p in DB. Thus, if we cannot find an item that appears in each i^{th} maximal interval of S_p then there is no backward extension of the sequence S_p in DB.

A prefix sequence cannot be closed if there exists a specialization of an item of the sequence; LGS order allows us to extract closed sequences by considering sequences that contain the most specific items (no or few $*$). Thus, if there exists a specialization of an item of a sequence, then the specialized sequence" that contains at least one more specific items is already extracted and added in the set of closed multidimensional sequential patterns. As a consequence, if a sequence is potentially closed (no backward and forward extension), it is sufficient to check if there is no more specific sequence in the set of already mined closed sequence. It should be noted that this set is significantly smaller than the set of frequent sequences. In the worst case, it is necessary to consider all already mined closed sequences that have the same support of the current examined prefix sequence.

4.3.2 Pruning The Search Space

While seeking new sequences with frequent sequence enumeration algorithm , we can use the bidirectional closure property (theorem 1) to check if the current prefix sequence is closed in order to generate a set of non-redundant knowledge. Although this property allows to return a more compact set, it does not allow to retrieve sequences more efficiently. For instance, there is no closed sequence prefixed by a sequence s, therefore it is useless to continue to search such sequences in this case. We define a pruning method to reduce space search by not considering unpromising sequence.

As noticed previously, a sequence may appear several times in a data sequence. In Definition 11, we introduced the notion of maximal interval to detect all the backward extensions. Now, we want to minimize these intervals in order to detect the unpromising sequences. We thus define the notion of i^{th} minimal interval.

Definition 12 (i^{th} Minimal Interval) *Let S be a data sequence that supports a g-k-sequence prefix sequence $S_p = \langle s_1, s_2, \ldots, s_g \rangle$, the i^{th} minimal interval of S_p in S is defined as follow:*

- *if $i = 1$: it is the subsequence before the $fa(s_1)$.*

– *if* $1 < i \leq g$*: it is the subsequence from* $fa(\langle s_1, \ldots, s_{i-1}\rangle)$ *to* $fa(s_i)$ *excluded such that* $fa(s_i) < fa(s_{i+1}) \leq \ldots \leq fa(s_g)$.

Theorem 2 (Pruning) *Let* $S_p = \langle s_1, s_2, \ldots, s_g\rangle$ *be a g-k prefix sequence, if there exists an integer i such that there is an item e that appears in each* i^{th} *minimal interval of* S_p *in DB, then there does not exist closed sequence with prefix* S_p.

Proof If an item e appears in each i^{th} minimal interval of S_p, then we can use the new prefix sequence S'_p that contains e. Indeed, $S_p \subset_S S'_p$ and $support(S'_p) = support(S_p)$. Thus, all locally frequent items on $DB|_{S_p}$ are also frequent on $DB|_{S'_p}$. Therefore, S_p is an unpromising sequence. There is no closed sequence with prefix S_p. The examination of S_p can thus be interrupted.

Let us consider the multidimensional sequence database from Tab. 4. We can stop the exploration of the prefix sequence $\langle(a_1, b_1, c_1)\rangle$ because item (a_1, b_2, c_3) appears in each first interval of $\langle(a_1, b_1, c_1)\rangle$ in DB. Thus, there is no hope to discover frequent closed sequence from prefix $\langle(a_1, b_1, c_1)\rangle$.

Thanks to theorems and definitions, we can now define the algorithm for mining closed multidimensional sequential patterns without candidate set maintenance.

4.3.3 Algorithm CMSP_Free

Algorithms 4 and 5 describe the extraction of closed sequential patterns without candidate set maintenance. These algorithms follow the same structure for mining frequent multidimensional sequences in depth first. Indeed, in the worst case (each frequent sequence is closed), the search spaces are the same. However, we introduce a pruning condition to efficiently reduce the search space. Algorithm 5 describes the key part of the extraction. First, if the number of backward and forward extensions of the current prefix sequence S_p is 0, then S_p is potentially closed and we have to check if there is no more specific sequence in set FCS that contains the already discovered closed multidimensional sequential patterns. If there is no such sequence with same support, then S_p is added to FCS. Set FCS is partitioned into subsets according to the support of the closed sequences. Thus, the search of more specific closed sequence than S_p is carried out on a subset of FCS. In the worst case, the complexity of this verification is $O(l_\sigma)$ where l_σ is the number of already mined closed sequences which support is equal to σ. Finally, each locally frequent item e on projected database is taking into account. The algorithm checks if it is possible to prune the search space according to the prefix sequence $S_p.e$ (e is added in the last itemset of S_p or in a new one). If it is not possible, the algorithm computes the number of backward extensions of $S_p.e$ and continue the discovery of closed sequences with this new prefix sequence $S_p.e$.

5 Experiments

In this section, we report experiments performed on synthetic data and real data.

5.1 Synthetic Data

We generated a database thanks to *IBM Quest Market-Basket Synthetic Data Generator* (100,000 tuples). Items (1 dimension) were then transformed into multidimensional

Algorithm 4: $CMSP\_Free$

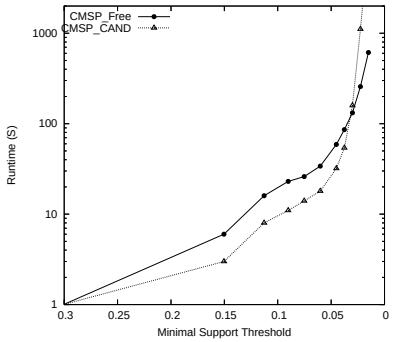
Data: Database DB, minimal support threshold $minsup$
Result: The set of closed multidimensional sequential patterns FCS
begin
  $FCS = \emptyset$;
  $F1 = getFrequentItems(DB, minsup)$;
  foreach *1-sequence* $f1 \in F1$ **do**
    if *Pruning of f1 is not possible)* **then**
      /*Counting the backward extensions.*/
      $BEI = \#$*backward extensions of sequence* $f1$ *in* DB ;
      Call subroutine $CMSP\_F(DB|_{f1}, f1, minsup, BEI, FCS)$;
  return FCS;
end

Algorithm 5: routine $CMSP\_F$

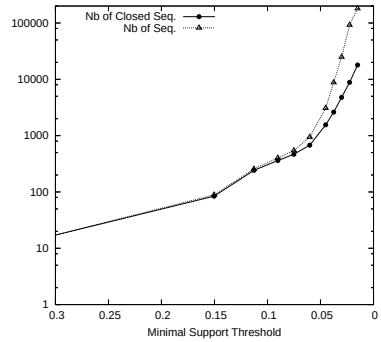
Data: Projected Database $DB|_{S_p}$, prefix sequence S_p, minimal support threshold $minsup$, number of backward extensions BEI
Result: The current set of closed multidimensional sequential patterns FCS
begin
  /*Search for frequent items and forward extensions*/
  $LFI = getFrequentItems(DB|_{S_p}, minsup)$;
  $FEI = |\{z \in LFI \ / \ support(z) = support(S_P)\}|$;
  if $(BEI + FEI) = 0$ **then**
    /*Checking for specialization*/
    if $(\nexists \alpha \in FCS \mid S_p \subset_S \alpha \wedge support(\alpha) = support(S_p))$ **then**
      $FCS = FCS \cup \{S_P\}$;
  foreach $i \in LFI$ **do**
    /*Adding frequent item to the prefix sequence (intra or inter itemset) and compute the projected database*/
    $S'_p = \langle S_p.i \rangle$;
    $DB|_{S'_p} = DB|_{S'_p}$;
  foreach $i \in LFI$ **do**
    /*Checking for pruning */
    if *Pruning of* S'_p *is not possible* **then**
      $BEI = \#$*backward extensions of sequence* S'_p *in* DB ;
      call subroutine $CMSP\_F(DB|S_pi, S'_p, minsup, BEI, FCS)$;
end

items (5 dimensions). Since approach without candidate maintenance has to compute backward and forward extension to determine if a sequence is closed, we suppose that an approach with candidate set maintenance is more efficient in sparse data (number of frequent sequences similar to number of closed ones). Experiments reported here confirm it. As soon as the minimal support threshold is low, the approach with candidate set maintenance is adapted. Indeed, the runtime of such approach is very sensitive to the number of frequent sequences since frequent sequences are potentially closed and retrieving all closed sequences is quadratic in the size of the set of candidate sequences. An approach without candidate set maintenance is more robust since it does not consider any set of candidates. Furthermore, an approach without candidate set maintenance provides efficient search space pruning properties. Figure 4(c) reports the behavior of $CMSP$ according to the size of de database (number of data sequences).

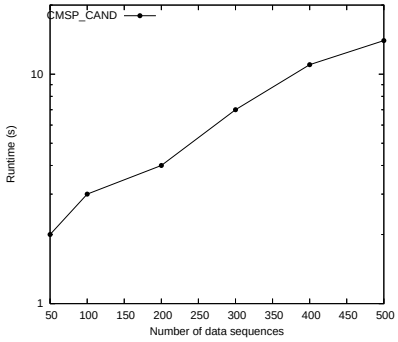
The runtime increases with the size of the database. Thus we can consider that $CMSP$ is scalable according to this parameter.



(a) Runtime over minimal support threshold



(b) # sequences and closed ones over minimal support threshold

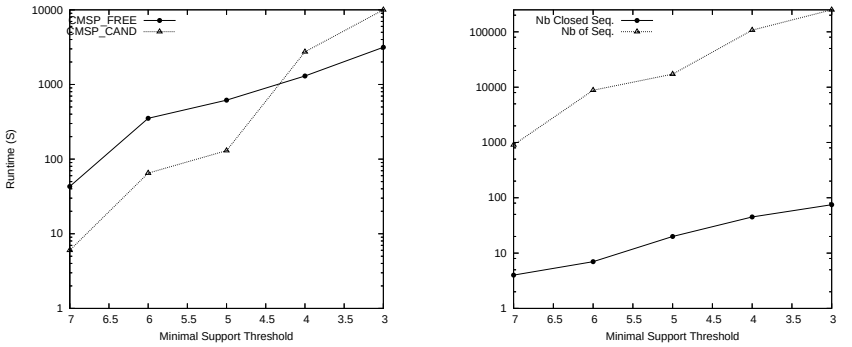


(c) Runtime over $|DB|$

Fig. 4 Experiments carried out on synthetic data

5.2 Real Data Cube

We report experiments performed on real data. They aim at showing the representative power of closed multidimensional patterns. They were performed on data cube issued from EDF (Electricité De France, the main French energy supplier and electricity producer) marketing context. This data cube describes the marketing activity on a very large EDF customer database (about 30 million of residential customers). We consider five analysis dimensions. As soon as the number of frequent sequences becomes too important, an approach with candidate set maintenance is not adapted whereas $CMSP$ allows the knowledge discovery with low support. We also notice the representative power of closed multidimensional sequential patterns. Indeed, a small number of closed sequences provides the representation of all frequent sequences without any loss of information on the support. As an example, $100,000$ sequences can be retrieved thanks to only about 100 closed sequences.



(a) Runtime over minimal support threshold

(b) # sequences and closed ones over minimal support threshold

Fig. 5 Experiments carried out on real data

6 Conclusion

In this paper, we propose a novel framework (definitions and algorithms) for mining closed multidimensional sequential patterns. Mining closed patterns leads to a condensed representation of the patterns without any loss of information. This advantage allows the computation of several measures (*e.g.* confidence for sequential rules) without any extra-scan of the database as all support values are known. Some works had been done on closed patterns and on closed sequential patterns. But we show in this paper that they cannot be directly applied to the multidimensional framework because of the *wild-carded* items we consider, leading to a non-easy lexical order. This paper introduces a new challenge with the inference of wild-carded items which are not directly materialized in the data sequences. Two approaches have been investigated, extending the *pattern growth* framework, with or without candidate maintenance and they are compared. Experiments on synthetic data and real data show the interest of our proposition.

In future work, we plan to consider time constraints and hierarchies which could be easily considered according to the definitions in this framework. It would be interesting to compare $CMSP$ algorithms against an adaptation of work on closed itemset generation using FCA. However, it is necessary to provide an closure operator on multidimensional sequence. The use of more condensed representations could allow a more efficient multidimensional sequential pattern mining. These representations (non-derivable [4], k-free [3]) exist in itemset framework but they do not exist yet for sequential or multidimensional patterns. This work presents thus a great challenge for future work.

Acknowledgements This work was partially funded by the EDF R&D Corporation in the framework of a collaboration aiming at studying OLAP Mining for discovering atypical temporal evolutions in data cubes. Thus the authors would like to thank Françoise Guisnel, Sabine Goutier and Marie-Luce Picard, for providing real data to assess our approach.

References

1. R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 1995 Int. Conf. Data Engineering (ICDE'95)*, pages 3–14, 1995.
2. Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Sequential pattern mining using a bitmap representation. In *KDD*, pages 429–435, 2002.
3. Jean-François Boulicaut, Artur Bykowski, and Christophe Rigotti. Free-sets: A condensed representation of boolean data for the approximation of frequency queries. *Data Min. Knowl. Discov.*, 7(1):5–22, 2003.
4. Toon Calders and Bart Goethals. Mining all non-derivable frequent itemsets. In *PKDD*, pages 74–85, 2002.
5. Mohammad El-Hajj and Osmar R. Zaïane. Finding all frequent patterns starting from the closure. In *ADMA*, pages 67–74, 2005.
6. B. Ganter and R. Wille. Applied lattice theory: Formal concept analysis, 1997.
7. Jiawei Han. OLAP mining: Integration of OLAP with data mining. In *IFIP Conference on Data Semantics*, pages 1–11, 1997.
8. Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *SIGMOD Conference*, pages 1–12, 2000.
9. F. Masseglia, F. Cathala, and P. Poncelet. The PSP Approach for Mining Sequential Patterns. In *Proc. of PKDD*, volume 1510 of *LNCS*, pages 176–184, 1998.
10. Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT*, pages 398–416, 1999.
11. J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(10), 2004.
12. Jian Pei. *Pattern-growth Methods for Frequent Pattern Mining*. PhD thesis, School of Computing Science, Simon Fraser University, Canada, 2002.
13. Jian Pei, Jiawei Han, and Runying Mao. Closet: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 21–30, 2000.
14. Helen Pinto, Jiawei Han, Jian Pei, Ke Wang, Qiming Chen, and Umeshwar Dayal. Multi-dimensional sequential pattern mining. In *CIKM*, pages 81–88, 2001.
15. Marc Plantevit, Yeow Wei Choong, Anne Laurent, Dominique Laurent, and Maguelonne Teisseire. M^2sp: Mining sequential patterns among several dimensions. In *PKDD*, pages 205–216, 2005.
16. C. Rassi and P. Poncelet. Deducing bounds on the support of sequential patterns. Technical report, june 2006.
17. Panida Songram, Veera Boonjing, and Sarun Intakosum. Closed multidimensional sequential pattern mining. In *ITNG*, pages 512–517, 2006.
18. Jianyong Wang, Jiawei Han, and Chun Li. Frequent closed sequence mining without candidate maintenance. *IEEE Trans. Knowl. Data Eng.*, 19(8):1042–1056, 2007.
19. Jianyong Wang, Jiawei Han, and Jian Pei. Closet+: searching for the best strategies for mining frequent closed itemsets. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 236–245, New York, NY, USA, 2003. ACM.
20. Sadok Ben Yahia, Tarek Hamrouni, and Engelbert Mephu Nguifo. Frequent closed itemset based algorithms: a thorough structural and analytical survey. *SIGKDD Explorations*, 8(1):93–104, 2006.
21. Xifeng Yan, Jiawei Han, and Ramin Afshar. Clospan: Mining closed sequential patterns in large databases. In *SDM*, 2003.
22. C.-C. Yu and Y.-L. Chen. Mining sequential patterns from multidimensional sequence data. *IEEE Transactions on Knowledge and Data Engineering*, 17(1):136–140, 2005.
23. Mohammed Javeed Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.
24. Mohammed Javeed Zaki and Ching-Jiu Hsiao. Charm: An efficient algorithm for closed itemset mining. In *SDM*, 2002.