Constraints
DOI 10.1007/s10601-016-9252-z

# Prefix-projection global constraint and top-*k* approach for sequential pattern mining

**Amina Kemmar[1] · Yahia Lebbah[1] · Samir Loudni[2] · Patrice Boizumault[2] · Thierry Charnois[3]**



**Abstract** Sequential pattern mining (SPM) is an important data mining problem with broad applications. SPM is a hard problem due to the huge number of intermediate subsequences to be considered. State of the art approaches for SPM (e.g., PREFIXSPAN Pei et al. 2001) are largely based on the pattern-growth approach, where for each frequent prefix subsequence, only its related suffix subsequences need to be considered, and the database is recursively projected into smaller ones. Many authors have promoted the use of constraints to focus on the most promising patterns according to the interests of the end user. The top-*k* SPM problem is also used to cope with the difficulty of thresholding and to control the number of solutions. State of the art methods developed for SPM and top-*k* SPM, though efficient, are locked into a rather rigid search strategy, and suffer from the lack of declarativity and flexibility. Indeed, adding new constraints usually amounts to changing the data-structures used in the core of the algorithm, and combining these new constraints often require new developments. Recent works (e.g. Kemmar et al. 2014; Négrevergne and

✉ Yahia Lebbah
lebbah.yahia@univ-oran.dz; ylebbah@gmail.com

Amina Kemmar
kemmar.amina@edu.univ-oran1.dz

Samir Loudni
samir.loudni@unicaen.fr

Patrice Boizumault
patrice.boizumault@unicaen.fr

Thierry Charnois
thierry.charnois@lipn.univ-paris13.fr

[1] LITIO, University of Oran 1 Ahmed Ben Bella, Oran, Algeria

[2] GREYC (CNRS UMR 6072), University of Caen, Caen, France

[3] LIPN (CNRS UMR 7030), University PARIS 13, Paris, France

Published online: 26 August 2016

Guns 2015) have investigated the use of Constraint Programming (CP) for SPM. However, despite their nice declarative aspects, all these modelings have scaling problems, due to the huge size of their constraint networks. To address this issue, we propose the Prefix-Projection global constraint, which encapsulates both the subsequence relation as well as the frequency constraint. Its filtering algorithm relies on the principle of projected databases which allows to keep in the variables domain, only values leading to a frequent pattern in the database. Prefix-Projection filtering algorithm enforces domain consistency on the variable succeeding the current frequent prefix in polynomial time. This global constraint also allows for a straightforward implementation of additional constraints such as size, item membership, regular expressions and any combination of them. Experimental results show that our approach clearly outperforms existing CP approaches and competes well with the state-of-the-art methods on large datasets for mining frequent sequential patterns, sequential patterns under various constraints, and top-$k$ sequential patterns. Unlike existing CP methods, our approach achieves a better scalability.

**Keywords** Global constraints · Data mining · Sequential pattern mining · Prefix-Projection · Top-k

## 1 Introduction

Mining sequential patterns (SPM) is an important task in data mining with many useful applications, including the analysis of medical or biological data and textual data, the analysis of web click-streams, the analysis of DNA sequences, and so on. Given a set of sequences, where each sequence consists of an ordered list of items, SPM consists in finding sequential patterns (sequences of items), which are sub-sequences of at least $minsup$ sequences, where $minsup$ is a minimum support threshold. For instance, let us consider the set of items $\{A, B, C, D\}$ and the database depicted by Table 1. Let $minsup = 2$, $\langle AC \rangle$ is a sequential pattern since it appears twice:[1] first, in $\langle \underline{\mathbf{A}}B\underline{\mathbf{C}}BC \rangle$ and second, in $\langle B\underline{\mathbf{A}}B\underline{\mathbf{C}} \rangle$.

SPM is a hard problem due to the huge number of intermediate subsequences to be considered, particularly when a large number of long frequent subsequences exist, such as in DNA sequence data sets, or at a low minimum support [21]. State of the art approaches for SPM, (e.g., PREFIXSPAN [20]) adopt a pattern growth approach which is considered to be the most efficient one for SPM. Indeed, the generation of infrequent candidate patterns is avoided by recursively projecting the database into smaller ones. For each frequent prefix subsequence, only its related suffix subsequences need to be considered without candidate generation. Many authors, (e.g., Pei et al. [22]) have promoted the use of constraints to focus on the most promising patterns according to the interests of the end user. Srikant and Agrawal [29] have generalized the scope of SPM [1] to include time constraints and sliding time windows. Garofalakis et al. [9] have introduced regular expressions as constraints for SPM. A few other constraints have been proposed (e.g. [15, 32, 34, 36]).

Moreover, top-$k$ SPM is also used to cope with the difficulty of thresholding and to control the number of solutions. Unless specific domain knowledge is available, the

[1] $\langle AC \rangle$ appears respectively in the first sequence $\langle ABCBC \rangle$ (i.e., two positions (1, 3) at $\langle \underline{\mathbf{A}}B\underline{\mathbf{C}}BC \rangle$ and (1, 5) at $\langle \underline{\mathbf{A}}BCB\underline{\mathbf{C}} \rangle$) and in the second sequence $\langle BABC \rangle$ (i.e., one position (2, 4) at $\langle B\underline{\mathbf{A}}B\underline{\mathbf{C}} \rangle$); thus it is a frequent pattern. Notice that when some sequential pattern appears many times in some sequence (e.g., $\langle AC \rangle$ in $\langle ABCBC \rangle$), it is considered a single occurrence.

**Table 1** Example of a sequence database $SDB_1$

| sid | Sequence |
|---|---|
| 1 | $\langle ABCBC \rangle$ |
| 2 | $\langle BABC \rangle$ |
| 3 | $\langle AB \rangle$ |
| 4 | $\langle BCD \rangle$ |

choice of a threshold is often arbitrary and relevant patterns can be missed. This drawback is obviously even deeper when several constraints have to be combined and thus several thresholds are required. So, measure functions are introduced to reflect the interestingness of a pattern and to determine the $k$ best patterns. Several methods have been designed to compute the top-$k$ patterns [5, 11, 24, 31, 33]. Even if the specific methods developed for SPM and top-$k$ SPM are efficient, they all lack of declarativity and flexibility. Indeed, adding new constraints usually amounts to changing the data-structures used in the core of the algorithm, and combining these new constraints often require new developments.

Several proposals have investigated relationships between SPM and constraint programming (CP) to revisit data mining tasks in a declarative and generic way [6, 14, 16]. The declarative aspect represents the key advantage of the proposed CP approaches. By this way, one can add/remove any user-constraint without requiring the explicit development of new specialized solving methods. But, all these proposals have scaling problems, due to the huge size of their constraints networks. Recently, Negrevergne et al. have presented in [18] two CP improved encodings for SPM. The first one uses a global constraint to encode the subsequence relation (denoted `global-p.f`), while the second one encodes explicitly this relation using additional variables and constraints (denoted `decomposed-p.f`) allowing to express constraints over embeddings like the `max-gap` constraint. But these proposals still relies on reified constraints and additional variables.

For instance, consider the problem of finding sequential patterns $p$ belonging to at least $minsup$ sequences of the database given in Table 1. An array $S$ of 4 boolean variables (one per sequence) is defined, representing which sequences of the $SDB_1$ include the current pattern. Then, for each sequence $s_i$ of $SDB_1$, a reified constraint, stating whether (or not) the unknown pattern $p$ is a subsequence of $s_i$, is imposed: $(S_i \Leftrightarrow p \preceq s_i)$. To ensure that at least $minsup$ sequences of the database should include the pattern $p$, we add the following minimum frequency constraint: $S_1 + S_2 + S_3 + S_4 \geq minsup$. Solving these 5 constraints allows to find sequential patterns covering $minsup$ sequences of Table 1. Even if this encoding provides an elegant modelling, it has a major drawback since the number of constraints and variables grows linearly with the size of the database. It is clear from the context that such a model cannot cope with large databases, since that managing a huge number of constraints is very costly for CP solvers.

Therefore, we propose in this work a single global constraint, called Prefix-Projection, to encode the subsequence relation and the frequency constraint which represents the core of the SPM problem. The filtering algorithm of this constraint is based on the principle of projected databases [20] which allows to remove from domains, the values leading to infrequent patterns. The frequent items in each projected database represent consistent values allowing to extend the current partial assignment of variables. The global constraint Prefix-Projection can be combined easily with additional constraints to express many restrictions on the extracted patterns.

**Contributions and roadmap**

1. We introduce the Prefix-Projection global constraint for SPM. This global constraint encapsulates both the subsequence relation as well as the frequency constraint into a single constraint, thus avoiding extra variables and extra constraints. Its filtering relies on the principle of projected databases. Prefix-Projection filtering algorithm enforces domain consistency on the variable succeeding the current frequent prefix in polynomial time. Moreover, we show how this global constraint also allows for a straightforward implementation of additional constraints such as size, item membership, regular expressions and any combination of them.
2. We show how Prefix-Projection global constraint can be exploited for mining top-$k$ sequential patterns, and propose an effective strategy for initializing the top-$k$ patterns with the most promising ones so that high support patterns can be derived earlier. Then, we investigate the effect of adding constraints for mining top-$k$ sequential patterns, and show how they can be easily combined with Prefix-Projection.
3. We present an extensive empirical study which includes a wide range of real datasets and comparisons of our techniques to state-of-the-art ones. Experimental results show that our approach clearly outperforms CP approaches and competes well with the state-of-the-art methods on large datasets for mining frequent sequential patterns, sequential patterns under various constraints, and top-$k$ sequential patterns. Our approach achieves a better scalability while it is a major issue for CP approaches.

Compared to the conference paper [12], contribution #1 is presented in more depth with additional materials in the core of the paper and new experiments evaluating the scalability have been performed (see Section 6.7). Contribution #2 (top-$k$ SPM) and all related experiments are new (see Section 5 and Section 6.8).

The paper is organized as follows. Section 2 recalls preliminaries. Section 3 provides a critical review of ad hoc methods and CP approaches for SPM. Section 4 presents the global constraint Prefix-Projection. Section 5 details our approach for mining top-$k$ sequential patterns. Section 6 reports experiments we performed. Finally, we conclude and draw some perspectives.

## 2 Preliminaries

This section presents background knowledge about sequential pattern mining and constraint satisfaction problems.

### 2.1 Sequential patterns

Let $\mathcal{I}$ be a finite set of *items*. The language of sequences corresponds to $\mathcal{L}_{\mathcal{I}} = \mathcal{I}^n$ where $n \in \mathbb{N}^+$.

**Definition 1** (**sequence, sequence database**) A sequence $s$ over $\mathcal{L}_{\mathcal{I}}$ is an ordered list $\langle s_1 s_2 \dots s_n \rangle$, where $s_i$, $1 \leq i \leq n$, is an item. $n$ is called the length of the sequence $s$. A sequence database $SDB$ is a set of tuples $(sid, s)$, where $sid$ is a sequence identifier and $s$ a sequence.

*Example 1* (Running example) Let us consider the sequence database $SDB_1$ given in Table 1. $SDB_1$ contains four sequences where the set of items is $\mathcal{I} = \{A, B, C, D\}$. The

sequence $s = \langle ABCBC \rangle$ has 5 items, where items $B$ and $C$ appear two times. The length of $s$ is 5, we say that $s$ is a 5-length sequence.

In sequence mining, a key concept is the pattern subsequence relation. A sequence $\alpha$ is subsequence of another sequence $s$ if there exists a mapping of every item in $\alpha$ to the same symbol in the sequence $s$ such that the order is respected.

**Definition 2** (subsequence, $\preceq$ relation) A sequence $\alpha = \langle \alpha_1 \dots \alpha_m \rangle$ is a subsequence of $s = \langle s_1 \dots s_n \rangle$, denoted by $(\alpha \preceq s)$, if $m \leq n$ and there exist integers $1 \leq j_1 \leq \dots \leq j_m \leq n$, such that $\alpha_i = s_{j_i}$ for all $1 \leq i \leq m$. We also say that $\alpha$ is contained in $s$ or $s$ is a super-sequence of $\alpha$. A tuple $(sid, s)$ contains a sequence $\alpha$, if $\alpha \preceq s$.

*Example 2* For example, consider the sequence $\langle ABCBC \rangle$ of $SDB_1$ given in Table 1, then $\langle AC \rangle$ is a subsequence of $\langle ABCBC \rangle$, denoted by $\langle AC \rangle \preceq \langle ABCBC \rangle$. We also say that $\langle ABCBC \rangle$ is a super-sequence of $\langle AC \rangle$.

The cover of a sequence $p$ in $SDB$ is the set of all tuples in $SDB$ in which $p$ is contained. The support of a sequence $p$ in $SDB$ is the number of tuples in $SDB$ which contain $p$.

**Definition 3** (coverage, support) Let $SDB$ be a sequence database and $p$ a sequence. $cover_{SDB}(p) = \{(sid, s) \in SDB \mid p \preceq s\}$ and $sup_{SDB}(p) = \#cover_{SDB}(p)$.

*Example 3* Assume that $p = \langle BC \rangle$, $cover_{SDB_1}(p) = \{(1, \langle A\mathbf{BC}BC \rangle), (2, \langle \mathbf{B}AB\mathbf{C} \rangle), (4, \langle \mathbf{BC}D \rangle)\}$, then $sup_{SDB_1}(p) = 3$.

To decide if a pattern $p$ is frequent or not in the sequence database, we have to define some threshold value called the *minimum support threshold* and denoted by $minsup$.

**Definition 4** (sequential pattern) Given a minimum support threshold $minsup$, every sequence $p$ such that $sup_{SDB}(p) \geq minsup$ is called a sequential pattern [1]. $p$ is said to be frequent in $SDB$.

*Example 4* Consider the sequence $p = \langle AC \rangle$. Since $p$ appears only in the two sequences $s_1$ ans $s_2$ in $SDB_1$, we have $cover_{SDB1}(p) = \{(1, s_1), (2, s_2)\}$. Given $minsup = 2$, $p = \langle AC \rangle$ is a sequential pattern because $sup_{SDB_1}(p) \geq 2$.

**Definition 5** (sequential pattern mining (SPM)) Given a sequence database $SDB$ and a minimum support threshold $minsup$. The problem of sequential pattern mining (SPM) is to find all patterns $p$ such that $sup_{SDB}(p) \geq minsup$.

*Example 5* Solving the SPM problem on the sequence database $SDB_1$ with $minsup = 3$ consists to find the set of patterns which appear at least in 3 sequences in $SDB_1$. We obtain the following sequential patterns: $\langle A \rangle$, $\langle B \rangle$, $\langle C \rangle$, $\langle AB \rangle$ and $\langle BC \rangle$.

Most SPM algorithms rely on the *anti-monotonicity property of frequency* to reduce the search space.

**Definition 6** (Anti-monotone constraint) A constraint $c$ is said to be anti-monotone if and only if, for every pattern satisfying $c$, all its subsequences satisfy $c$ as well.

Hence, by contraposition, if a pattern $p$ does not satisfy an anti-monotone constraint, then all of its super-sequences will not satisfy it. Since the frequency constraint is anti-monotone, once a pattern is detected as infrequent, then no super-sequence of it can be frequent. This property, which is at the heart of SPM algorithms, allows growing frequent sequences to their super-sequences, and trimming the exponential search space by ignoring super-sequences of non-frequent sequences. This is the well known pattern growth principle [20]. To illustrate the idea behind this principle, consider the sub-sequence $\langle ABC \rangle$ from $SDB_1$ (Table 1) with $minsup = 2$. Clearly, any sequence that contains $\langle ABC \rangle$ must also contain its sub-sequences, $\langle AB \rangle$, $\langle AC \rangle$, $\langle BC \rangle$, $\langle A \rangle$, $\langle B \rangle$, and $\langle C \rangle$. As a result, if $\langle ABC \rangle$ is frequent, then all sub-sequences of $\langle ABC \rangle$ must also be frequent. Conversely, if a sub-sequence such as $\langle CB \rangle$ is infrequent, then all of its super-sequences must be infrequent too.

## 2.2 SPM under constraints

Mining the complete set of frequent patterns is a difficult task since we are faced with the major issue of a huge number of patterns which are not all interesting to the user. Constraint-based approaches help to avoid these non interesting patterns, by allowing the user to express as many constraints as he wants, to produce only patterns of interest. In this section, we define the problem of mining sequential patterns in a sequence database satisfying user-defined constraints. Then, we review the most usual constraints for the sequential mining problem [22].

SPM under constraints problem consists of finding a set of patterns $\{p \in \mathcal{L}_{\mathcal{I}} \mid C(p, SDB)\ is\ true\}$, where $C(p, SDB)$ is a selection predicate that states the constraints under which the pattern $p$ is a solution w.r.t. the database SDB. In the following, we present different types of constraints that we explicit in the context of sequence mining. Some of these constraints will be handled by our approach (see Section 4.1).

1. **Minimum size constraint**. This constraint, denoted $minSize(p, \ell_{min})$, states that the number of items of $p$ must be greater than or equal to $\ell_{min}$.

   *Example 6* For instance, if we impose the constraint $minSize(p, 3) \wedge sup_{SDB_1}(p) \geq 2$, only two sequential patterns are mined from Table 1: $p_1 = \langle ABC \rangle$ and $p_2 = \langle BBC \rangle$.

2. **Maximum size constraint.** This constraint, denoted $maxSize(p, \ell_{max})$, restricts the maximum size (in number of items) of $p$ (i.e. $p$ must contain at most $\ell_{max}$ items).

   *Example 7* For instance, if we impose the constraint $maxSize(p, 2) \wedge sup_{SDB_1}(p) \geq 3$, only the following set of patterns are extracted from Table 1: $\langle A \rangle$, $\langle B \rangle$, $\langle C \rangle$, $\langle AB \rangle$ et $\langle BC \rangle$.

3. **Item constraint.** This constraint, denoted $item(p, t)$, states that an item $t$ must belong (or not) to a pattern $p$.

   *Example 8* Let the following three constraints, $sup_{SDB_1}(p) \geq 3 \wedge maxSize(p, 2) \wedge item(p, C)$, defined over sequences of Table 1. Only patterns $p_3 = \langle C \rangle$ and $p_4 = \langle BC \rangle$ satisfy these three constraints.

4. **Regular expression constraint.** A regular expression constraint $reg(p, exp)$ is a constraint specified as a regular expression $exp$ over the set of items of $SDB$. A pattern

$p$ satisfies $reg(p, exp)$ iff the pattern is accepted by its equivalent deterministic finite automata [9].

*Example 9* Let us consider the two constraints: $sup_{SDB_1}(p) \geq 2 \wedge reg(p, exp)$, where $exp = B\{BC|D\}$. The sequential pattern $p_5 = \langle BBC \rangle$ is extracted from $SDB_1$ (Table 1) since it satisfies the regular expression constraint and $cover_{SDB1}(p_5) = \{(1, s_1), (2, s_2)\}$.

5. **Aggregate constraint**. This constraint allows to express a constraint on an aggregation of items in a pattern. The aggregate function can be *sum*, *avg*, *max*, *min*, etc.

*Example 10* Suppose that each sequence of $SDB_1$ corresponds to a customer and every item in the sequence to a product bought by the customer. A price is associated to each product. In this context, a marketing analyst may be interested to sequential patterns where the average price of items is greater than a certain value.

All above constraints put restrictions on patterns. Other constraints impose a restriction on subsequence relation, by restricting the distance allowed between items in the sequence. Examples of such constraints are duration and gap constraints.

6. **Duration constraint**. The duration constraint is defined only if a time-stamp is attributed to each item in the sequence database. A pattern $p$ satisfies the duration constraint $Dur[M, N]$ if and only if the distance between the first and the last item of $p$ in the original sequences is greater than $M$ and lower than $N$.

*Example 11* If we consider the position of the item in the sequence as its corresponding timestamp, the sequence $\langle BCB \rangle$ is a subsequence of $\langle ABCBC \rangle$ under the duration constraint $Dur[0, 2]$.

7. **Gap constraint**. A sequential pattern with gap constraint $gap[M, N]$ is a pattern such that at least $M$ and at most $N$ items are allowed between every two adjacent items, in the original sequences. Formally, a sequence $p = \langle p_1 \dots p_m \rangle$ is a subsequence of $s = \langle s_1 \dots s_n \rangle$, under the gap constraint $gap[M, N]$, if $m \leq n$ and, for all $1 \leq i \leq m$, there exist integers $1 \leq j_1 \leq \dots \leq j_m \leq n$, such that $p_i = s_{j_i}$, and $\forall k \in \{1, ..., m - 1\}$, $M \leq j_{k+1} - j_k - 1 \leq N$.

*Example 12* Let us consider the sequences of $SDB_1$ (Table 1) and the gap constraint $gap[1, 1]$ (which means that one item must separate each two neighbor items). Patterns $\langle A \rangle$, $\langle B \rangle$, $\langle C \rangle$, $\langle AC \rangle$, $\langle BB \rangle$ and $\langle CC \rangle$ are subsequences of sequence $s_1$ under $gap[1, 1]$, whereas $\langle AB \rangle$ and $\langle BC \rangle$ are not.

In this paper, we will consider size, item and regular expression constraints.

### 2.3 Projected databases

We now present the necessary definitions related to the concept of *projected databases* [20].

**Definition 7** (prefix, projection, suffix) Let $\beta = \langle \beta_1 \dots \beta_n \rangle$ and $\alpha = \langle \alpha_1 \dots \alpha_m \rangle$ be two sequences, where $m \leq n$.

- Sequence $\alpha$ is called the prefix of $\beta$ iff $\forall i \in [1..m], \alpha_i = \beta_i$.
- Sequence $\beta = \langle \beta_1 \dots \beta_n \rangle$ is called the projection of some sequence $s$ w.r.t. $\alpha$, iff (1) $\beta \preceq s$, (2) $\alpha$ is a prefix of $\beta$ and (3) there exists no proper super-sequence $\beta'$ of $\beta$ such that $\beta' \preceq s$ and $\beta'$ also has $\alpha$ as prefix. In other words, $\beta$ is called the projection of $s$ w.r.t. $\alpha$ iff $\beta$ is the largest subsequence of $s$ which starts by $\alpha$.
- Sequence $\gamma = \langle \beta_{m+1} \dots \beta_n \rangle$ is called the suffix of $s$ w.r.t. $\alpha$. With the standard concatenation operator "*concat*", we have $\beta = concat(\alpha, \gamma)$.

*Example 13* Let $s_1 = \langle ABCBC \rangle$ be the first sequence of $SDB_1$. For instance, sequence $\alpha = \langle AC \rangle$ is a prefix of sequence $\beta = \langle ACBC \rangle$ and $\gamma = \langle BC \rangle$ is its suffix. Sequence $\beta = \langle ACBC \rangle$ is the projection of sequence $s_1$ w.r.t. $\alpha$.

**Definition 8** (**projected database**) Let $SDB$ be a sequence database, the $\alpha$-projected database, denoted by $SDB|_\alpha$, is the collection of suffixes of sequences in $SDB$ w.r.t. prefix $\alpha$.

*Example 14* Let us consider the sequence database of Table 1, and the prefixes $\langle A \rangle$, $\langle AB \rangle$ and $\langle ABC \rangle$. We have:

- $SDB_1|_{<A>} = \{(1, \langle BCBC \rangle), (2, \langle BC \rangle), (3, \langle B \rangle)\}$.
- $SDB_1|_{<AB>} = \{(1, \langle CBC \rangle), (2, \langle C \rangle), (3, \langle \rangle)\}$.
- $SDB_1|_{<ABC>} = \{(1, \langle BC \rangle), (2, \langle \rangle)\}$.

Pei et al. [20] proposed PREFIXSPAN, an efficient algorithm for mining sequential patterns based on the concept of *projected databases*. It proceeds by dividing the initial database into smaller ones projected on the frequent subsequences obtained so far; only their corresponding suffixes are kept. Then, sequential patterns are mined in each projected database by exploring only locally frequent items.

Algorithm 1 depicts the pseudo-code of PREFIXSPAN. Initially, procedure PREFIXSPANPROJ (line 2) is called with an empty prefix, the initial database and the minimum support $minsup$. Then, the $SDB$ is scanned once (line 3) to find all frequent items which will be considered later as prefix to form valid sequential patterns. Finally, each frequent item is appended to the current prefix $\alpha$ (line 5) and then PREFIXSPANPROJ is relaunched recursively in order to extend the current pattern (line 7). By this way, when PREFIXSPAN algorithm terminates, it provides the whole set of frequent patterns.

**Algorithm 1** PREFIXSPAN($SDB, minsup$)

**Data**: $SDB$: the initial database; $minsup$: the minimum support threshold
**Result**: $SP$: The complete set of sequential patterns
**begin**
1 $SP \leftarrow \emptyset$ ;
2 PREFIXSPANPROJ($\langle \rangle, SDB, minsup, SP$) ;
`Procedure` PREFIXSPANPROJ ($\alpha, SDB|_\alpha, minsup, SP$) ;
**begin**
3 $FreqItems \leftarrow$ Frequent items in $SDB|_\alpha$ w.r.t. $minsup$ allowing to extend $\alpha$ to form a new sequential pattern ;
4 **foreach** $a \in FreqItems$ **do**
5 $\alpha' \leftarrow concat(\alpha, a)$ ;
6 $SP \leftarrow SP \cup \{\alpha'\}$ ;
7 PREFIXSPANPROJ($\alpha', SDB|_{\alpha'}, minsup, SP$) ;

*Example 15* Let us consider the sequence database of Table 1 with $minsup = 2$. Figure 1 illustrates the execution of PREFIXSPAN on $SDB_1$. It starts by scanning $SDB_1$ to find all

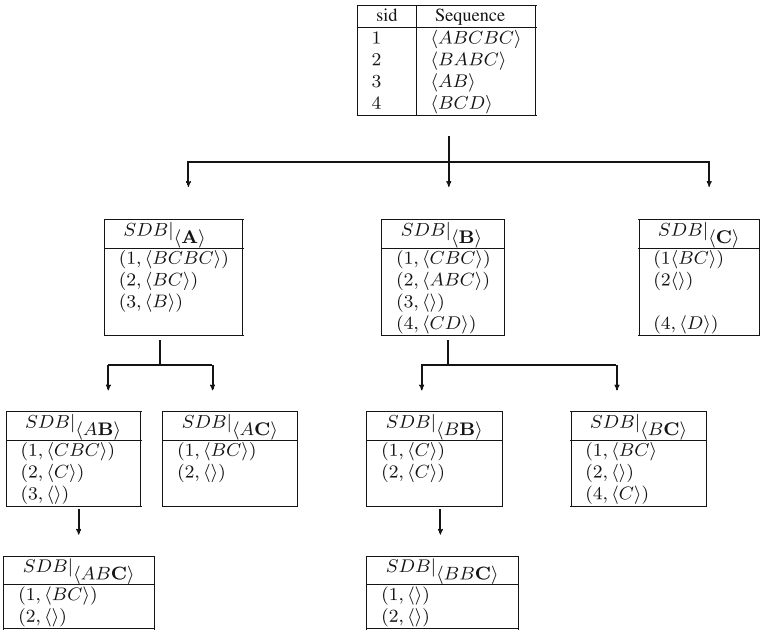



**Fig. 1** Mining sequential patterns from $SDB_1$ using PREFIXSPAN ($minsup = 2$)

the frequent items, each of them is used as a prefix to get projected databases. For $SDB_1$, we get 3 disjoint subsets w.r.t. the prefixes $\langle A\rangle$, $\langle B\rangle$, and $\langle C\rangle$. For instance, $SDB_1|_{\langle A\rangle}$ consists of 3 suffix sequences: $\{(1, \langle BCBC\rangle), (2, \langle BC\rangle), (3, \langle B\rangle)\}$ (see $SDB_1|_{\langle A\rangle}$ Fig. 1). Considering the projected database $SDB_1|_{<A>}$, its locally frequent items are $B$ and $C$. Thus, $SDB_1|_{<A>}$ can be recursively partitioned into 2 subsets w.r.t. the two prefixes $\langle AB\rangle$ and $\langle AC\rangle$. The $\langle AB\rangle$- and $\langle AC\rangle$- projected databases can be constructed and recursively mined similarly. The processing of an $\alpha$-projected database terminates when no frequent subsequence can be generated which is the case of the prefixes respectively $\langle ABC\rangle$, $\langle AC\rangle$, $\langle BBC\rangle$, $\langle BC\rangle$ and $\langle C\rangle$ (Fig. 1). As no frequent item exists in their corresponding projected databases, the mining process terminates.

In order to determine if a sequence $\gamma$ is frequent, we have to compute its support in the sequence database. To ensure an incremental computation, the support can be computed from the projected database w.r.t some prefix of $\gamma$. Proposition 1 establishes the support count of a sequence $\gamma = concat(\alpha, \beta)$ which is equal to the support of $\beta$ in $SDB|_\alpha$ [20].

**Proposition 1** (**Support count of a sequence**) *The support count of a sequence $\gamma$ in $SDB$ with prefix $\alpha$ and suffix $\beta$ s.t. $\gamma = concat(\alpha, \beta)$ is given by $sup_{SDB}(\gamma) = sup_{SDB|_\alpha}(\beta)$.*

This proposition ensures that only patterns grown from $\alpha$ need to be considered for the support count of a sequence $\gamma$. Furthermore, only those suffixes with prefix $\alpha$ should be counted.

*Example 16* Let us consider the sequence $\gamma = \langle ABC\rangle$. Suppose that $\alpha = \langle AB\rangle$, according to Proposition 1, we have $sup_{SDB_1}(\gamma) = sup_{SDB_1}(concat(\langle AB\rangle, \langle C\rangle)) =$

$sup_{SDB_1|_{\langle AB\rangle}}(\langle C\rangle)$. From example 15, we have $SDB_1|_{\langle AB\rangle} = \{(1, \langle CBC\rangle), (2, \langle C\rangle), (3, \langle\rangle)\}$. As item $C$ occurs only in the first and the second sequences in $SDB|_{\langle AB\rangle}$, then $sup_{SDB_1}(\langle ABC\rangle) = sup_{SDB_1|_{\langle AB\rangle}}(\langle C\rangle)=2$.

### 2.4 CSP and global constraints

A *Constraint Satisfaction Problem* (CSP) consists of a set $X$ of $n$ variables, a domain $\mathcal{D}$ mapping each variable $X_i \in X$ to a finite set of values $D(X_i)$, and a set of constraints $\mathcal{C}$. An assignment $\sigma$ is a mapping from variables in $X$ to values in their domains: $\forall X_i \in X, \sigma(X_i) \in D(X_i)$. A constraint $c \in \mathcal{C}$ is a subset of the cartesian product of the domains of the variables that are in $c$. The goal is to find an assignment such that all constraints are satisfied.

**CSPs solving** Constraint solvers interleave constraints filtering and search. The search algorithm enumerates the values of the variables until it finds all the solutions. Filtering algorithms are called usually at each node of the search tree, to remove as many inconsistent values as possible; it tries to prevent the solver to explore an exponential number of combinations. Each time some variable domain is modified, filtering algorithms are systematically applied to remove other inconsistent values. The solving process terminates when a solution is found or proving that no solution exists. In order to be effective, each constraint filtering algorithm should remove as many variable domain values as possible and possibly achieve *domain consistency* (also referred to as generalized-arc consistency).

**Definition 9** (Domain consistency) A constraint $c$ on $X$ is *domain consistent*, if and only if, for every $X_i \in X$ and for every $d_i \in D(X_i)$, there is an assignment $\sigma$ satisfying $c$ such that $\sigma(X_i) = d_i$. Such an assignment is called a support.

**Global constraints** provide shorthands to often-used combinatorial substructures. More precisely, a *global constraint* [27] is a constraint that captures a relation between a non-fixed number of variables. Let us take the All-Different constraint on variables $X = \{X_1, \ldots, X_n\}$ where the values taken by variables must be pairwise different. Suppose that we have a CSP with 3 variables $X_1$, $X_2$ and $X_3$ and an All-Different constraint on $\{X_1, X_2, X_3\}$ with $D(X_1) = D(X_2) = \{a, b\}$ and $D(X_3) = \{a, b, c\}$. This All-Different constraint can be encoded simply with three binary constraints $X_1 \neq X_2$, $X_2 \neq X_3$ and $X_1 \neq X_3$. It is clear from the context that each of these three constraints is domain consistent. But we can also encode the All-Different constraint with only one global constraint $\texttt{AllDifferent}(X_1, X_2, X_3)$ which ensures the pairwise differences. With this single constraint, we can see that the value $X_3 = a$ (resp. $X_3 = b$) cannot be extended with an assignment satisfying $\texttt{AllDifferent}(X_1, X_2, X_3)$. Note that the unique constraint $\texttt{AllDifferent}(X_1, X_2, X_3)$ was able to reduce the domains, while its decomposition $\{X_1 \neq X_2, X_2 \neq X_3, X_1 \neq X_3\}$ does not reduce anything. This example shows clearly the filtering effectiveness of working on a single constraint encoding the considered combinatorial structure instead its decomposition.

Global constraints have been firstly used in sequential pattern mining through the *exists-Embedding* constraints (one per sequence) which verify the inclusion relation directly on the sequence [18]. Later, [12] proposed the Prefix-Projection global constraint which has successfully encoded both the subsequence relation as well as the frequency constraint. More details are given in Section 3. In the following, we present two global constraints that will serve to encode item membership and regular expression constraints (see Section 2.2).

Let $X = \langle X_1, X_2, ..., X_n \rangle$ be a sequence of $n$ variables. Let $V$ be a set of values, $l$ and $u$ be two integers s.t. $0 \leq l \leq u \leq n$, the constraint $\texttt{Among}(X, V, l, u)$ states that each value $a \in V$ should occur at least $l$ times and at most $u$ times in $X$ [4]. Given a deterministic finite automaton $A$, the constraint $\texttt{Regular}(X, A)$ ensures that the sequence $X$ is accepted by $A$ [23].

## 3 Related works

This section provides a critical review of ad hoc methods and CP approaches for SPM and top-$k$ SPM.

### 3.1 Ad hoc methods for SPM

GSP [29] was the first algorithm proposed to extract sequential patterns. It uses a generate-and test approach. It generates candidates of frequent $(k + 1)$-sequences by performing a join on the frequent $k$-sequences. Later, two major classes of methods have been proposed:

- Depth-first search based on a vertical database format e.g. SPADE [37] or SPAM [2]. Each item, and consequently, each sequence is represented using its $id$ list where each $id$ corresponds to an item and a time-stamp. The support of a sequence is then obtained by joining the idlist of its items.
- Projected pattern growth such as PrefixSpan [20] and its extensions, e.g. CloSpan for mining closed sequential patterns [34] or Gap-BIDE [15] tackling the gap constraint.

Several specialised methods have addressed the problem of constrained SPM. Srikant and Agrawal [29] have generalized the scope of sequential pattern mining [1] to include time constraints and sliding time windows. Garofalakis et al. [9] have proposed regular expressions as constraints for SPM. Later, Trasarti et al. [30] has proposed Sequence Mining Automata (SMA), an approach based on a specialized kind of Petri Nets. Two variants of SMA were proposed: SMA-1P (SMA one pass) and SMA-FC (SMA Full Check). SMA-1P processes by means of the SMA all sequences one by one, and enters all resulting valid patterns in a hash table for support counting, while SMA-FC allows frequency based pruning during the scan of the database. CloSpan [34] and Gap-BIDE [15] are both extensions of PrefixSpan to mine closed frequent patterns and closed frequent patterns with gap constraints. cSpade [36] is an extension of SPADE algorithm incorporating constraints (max-gap, max-span, length). Finally, Pei et al. [22] provide a survey for other constraints such as regular expressions, length and aggregates.

However, none of these approaches is generic. Indeed, adding new constraints usually amounts to changing the data-structures used in the core of the algorithm, and combining these new constraints often require new developments.

### 3.2 Existing CP methods for SPM

Following the work of Guns et al. [10] for itemset mining, several methods have been proposed to mine sequential patterns using CP. Coquery et al. have proposed in [6] a first SAT-based model for discovering sequence patterns with explicit wildcards in a single sequence under different types of constraints (e.g. frequency, maximality, closedness). A wildcard represents the presence of exactly one arbitrary symbol in that position in the

sequence. Kemmar et al. [14] have also studied patterns with explicit wildcards, but in a database of sequences. They show how some constraints dealing with individual patterns (e.g. frequency, size, gap, regular expressions) and constraints defining more complex patterns such as relevant subgroups [19] and top-$k$ patterns can be modeled using a CSP. However, the sequential patterns with non-contiguous[2] items are modeled using empty items as wildcards. For instance, if we consider our running example (Table 1), the two patterns $\langle A \diamond C \rangle$ and $\langle A \diamond \diamond C \rangle$ are considered as different. Metivier et al. [16] have proposed a CSP model for SPM. Each sequence is encoded by an automaton capturing all subsequences that can occur in it. This CP approach easily enables to address different constraints on patterns, but it is not enough effective to handle large sequence databases ($\geq$ 500 sequences). Recently, Negrevergne et al. have presented in [18] two CP encodings for SPM. The first one uses a global constraint to encode the subsequence relation (denoted `global-p.f`), while the second one encodes explicitly this relation using additional variables and constraints (denoted `decomposed-p.f`).

All these proposals use **reified constraints** to encode the database. A reified constraint associates a boolean variable to a constraint reflecting whether the constraint is satisfied (value 1) or not (value 0). For each sequence $s$ of $SDB$, a reified constraint, stating whether (or not) the unknown pattern $p$ is a subsequence of $s$, is imposed: $(S_s = 1) \Leftrightarrow (p \preceq s)$. A great consequence is that the encoding of the frequency measure is straightforward: $freq(p) = \sum_{s \in SDB} S_s$. But such an encoding has a major drawback since it requires $(m = \#SDB)$ reified constraints to encode the whole database. This constitutes a strong limitation of the size of the databases that could be managed.

Most of these proposals encode **the subsequence relation** $(p \preceq s)$ using variables $Pos_{s,j}$ $(s \in SDB$ and $1 \leq j \leq \ell)$ to determine a position (or an occurrence) where $p$ occurs in $s$. When only few occurrences are possible, as in sequence patterns with explicit wildcards, this can be performed with a disjunctive constraint over all possible occurrences (see [14] for more details). But for standard sequences (the setting we address in this paper), the number of such occurrences is exponential, thus prohibiting a direct encoding and making the search algorithm computationally expensive. Moreover, it requires a large number of additional variables $(m \times \ell)$.

To address this drawback, Negrevergne et al. have proposed in [18] a global constraint `exists-embedding` to encode the subsequence relation. They use the concept of projected frequency to keep only locally frequent items. However, to recompute and to keep frequent symbols during search, they introduce for every sequence an auxiliary integer variable used to keep items that appear after the current prefix. To avoid searching over infrequent items, a specific search routine (based on domains of auxiliary variables) are used, making the integration quite complex (see Section 4.5 for more details). Moreover, the proposed encoding still relies on reified constraints and requires to impose $m$ `exists-embedding` global constraints. Our approach does not require any extra variables nor a specific branching strategy. Moreover, the global constraint Prefix-Projection will encode the anti-monotonicity of frequency (see Definition 6) in a simple and elegant way, while existing CP methods for SPM have difficulties to handle this property (see Section 4.2(c) for a more detailed study).

So, we propose in the next section the Prefix-Projection global constraint that fully exploits the principle of projected databases to encode both the subsequence relation and the frequency constraint. Prefix-Projection does not require any reified constraints nor any

[2]We say that a pattern is non-contiguous if it contains at least one wildcard before the last item.

extra variables to encode the subsequence relation. As a consequence, usual SPM constraints (see Section 2.2) can be encoded in a straightforward way using directly the elementary constraints and/or the global constraints provided by the CP solver.

### 3.3 Ad hoc methods for top-$k$ SPM

Finding the top-$k$ patterns has been widely considered for itemsets [5, 11]. New approaches for mining top-$k$ frequent closed patterns based on the FP-tree technique have also been proposed [24, 33]. For sequential data, Tzvetkov et al. have proposed in [31] a first algorithm, called `TSP`, for mining top-$k$ sequential patterns or top-$k$ closed sequential patterns of length no less than $\ell_{min}$. `TSP` uses a pattern-growth approach based on `PrefixSpan` [20] to explore the search space of patterns. It proceeds in three steps: (1) initially, build a small, limited number of projected databases for each prefix length $l$ ($l < \ell_{min}$), (2) then, gradually relax the limitation on the number of projected databases that are built, and (3) repeat the mining again. Each time a projected database $SDB|_\alpha$ is reached, where $\#\alpha = (\ell_{min} - 1)$, $SDB|_\alpha$ is computed completely and the mined sequences are used to raise $minsup$. The algorithm stops when all projected databases at level $\ell_{min}$ with support greater than $minsup$ are mined completely. The downside of this approach is that projecting/scanning databases repeatedly is costly, and that cost becomes huge for dense databases. Later, Fournier-Viger et al. have proposed in [8] an efficient algorithm, called `TKS`, for mining top-$k$ sequential patterns. `TKS` uses the vertical database representation and candidate-generation procedure of `SPAM`. It also uses an additional list that maintains at any time the set of patterns that can be extended to generate candidates. This list is exploited for support raising, by extending the pattern having the highest support first. Furthermore, to reduce the number of candidates when extending a sequential pattern, `TKS` records in a hash table the list of items that become infrequent when $minsup$ is raised by the algorithm. Moreover, a second pruning mechanism based on Precedence Map structure is also exploited (see [8] for more details).

Our top-$k$ approach differs from the above algorithms in the raising method. Thanks to the Prefix-Projection global constraint, we propose an effective strategy for initializing the top-$k$ patterns with the most promising ones, so that high support patterns can be derived earlier. Unlike specialized methods like `TSP` or `TKS`, our approach allows to define constraints on top-$k$ patterns as minimum size, item membership and regular expression constraints.

## 4 Prefix-projection global constraint

This section presents the Prefix-Projection global constraint for the SPM problem. Section 4.1 defines the Prefix-Projection global constraint and presents the CSP modeling. Section 4.2 shows how the filtering of global constraint Prefix-Projection can take advantage of the anti-monotonicity of the frequency measure (see Proposition 4) to reduce the variable domains. Sections 4.3 and 4.4 detail respectively the construction of projected databases and the filtering algorithm of Prefix-Projection.

### 4.1 CSP modeling for SPM

(a) **Variables and domains.** Let $P$ be the unknown pattern of size $\ell$ we are looking for. The symbol $\square$ ($\square \notin \mathcal{I}$) stands for an empty item and denotes the end of a sequence. We encode the unknown pattern $P$ of maximum length $\ell$ with a sequence of $\ell$ variables $\langle P_1, P_2, \ldots, P_\ell \rangle$. Each variable $P_j$ represents the item in the $jth$ position of the

sequence. The size $\ell$ of $P$ is determined by the length of the longest sequence of $SDB$. The domains of variables are defined as follows: (i) $D(P_1) = \mathcal{I}$ to avoid the empty sequence, and (ii) $\forall i \in \{2 \dots \ell\}$, $D(P_i) = \mathcal{I} \cup \{\square\}$. Patterns with $k < \ell$ symbols are represented with $k$ items from $\mathcal{I}$; the $(\ell - k)$ last positions are filled with the empty item $\square$. To avoid enumerating the same pattern with $\square$ values in different positions, we impose that $\forall i \in \{2..(\ell - 1)\}$, $(P_i = \square) \rightarrow (P_{i+1} = \square)$. Thus, empty items can only appear at the end.

(b) **Definition of Prefix-Projection.** The global constraint Prefix-Projection encodes both subsequence relation and minimum frequency constraint directly on the data.

**Definition 10** (Prefix-Projection global constraint) Let $P = \langle P_1, P_2, \dots, P_\ell \rangle$ be a pattern of size $\ell$. The Prefix-Projection$(P, SDB, minsup)$ constraint holds if and only if there exists an assignment $\sigma = \langle d_1, ..., d_\ell \rangle \in D(P_1) \times \dots \times D(P_\ell)$ of variables of $P$ such that $sup_{SDB}(\langle d_1, ..., d_\ell \rangle) \geq minsup$.

*Example 17* Consider the sequence database $SDB_1$ given in Table 1 with $minsup = 2$. Let $P = \langle P_1, P_2, P_3 \rangle$ $(\ell = 3)$ with $D(P_1) = \mathcal{I}$ and $D(P_2) = D(P_3) = \mathcal{I} \cup \{\square\}$. Consider 2-length sequential pattern $\langle AB \rangle$. Since $\ell = 3$, we have $\sigma(P_1) = A, \sigma(P_2) = B$, and $\sigma(P_3) = \square$. Prefix-Projection$(P, SDB_1, 2)$ holds since $sup_{SDB_1}(P) \geq 2$.

Proposition 2 establishes a necessary and sufficient condition to ensure that an assignment $\sigma$ satisfies the Prefix-Projection global constraint.

**Proposition 2** (**Prefix-Projection Consistency**) *A Prefix-Projection$(P, SDB, minsup)$ constraint has a solution if and only if there exists an assignment $\sigma = \langle d_1, ..., d_\ell \rangle$ of variables of $P$ s.t. $SDB|_\sigma$ has at least $minsup$ suffixes of $\sigma$: $\#SDB|_\sigma \geq minsup$.*

*Proof* From proposition 1, we have straightforwardly $sup_{SDB}(\sigma) = sup_{SDB|_\sigma}(\langle\rangle) = \#SDB|_\sigma$. Thus, suffixes of $SDB|_\sigma$ are supports of $\sigma$ in the constraint Prefix-Projection$(P, SDB, minsup)$, provided that $\#SDB|_\sigma \geq minsup$. $\square$

Proposition 2 shows that any sequence pruned from $SDB|_\alpha$ does not need to be considered when computing the support of a sequential pattern $\gamma$ grown from $\alpha$ later. Thus, we can stop growing $\alpha$, once we find that $\alpha$ is infrequent.

*Example 18* Let us consider the constraint Prefix-Projection$(P, SDB_1, 3)$ s.t. $P =< P_1, P_2, P_3 >$. Solutions of this constraint is given by the set of assignments $\sigma$ s.t. $sup_{SDB_1}(\sigma) = \#SDB_1|_\sigma \geq 3$, we get the following solutions: $\langle A\square\square \rangle$, $\langle B\square\square \rangle$, $\langle C\square\square \rangle$, $\langle AB\square \rangle$ and $\langle BC\square \rangle$.

(c) **Handling other SPM constraints:** Constraints that put restrictions on the structure of the pattern can be encoded in a straightforward way using directly the elementary constraints and/or the global constraints provided by the CP solver. Examples are given bellow:

- *Minimum size constraint*: $minSize(P, \ell_{min}) \equiv \bigwedge_{i=1}^{i=\ell_{min}} (P_i \neq \square)$
- *Maximum size constraint*: $maxSize(P, \ell_{max}) \equiv \bigwedge_{i=\ell_{max}+1}^{i=\ell} (P_i = \square)$

- *Item constraint*: let $V$ be a subset of items, $l$ and $u$ two integers s.t. $0 \leq l \leq u \leq \ell$. $item(P, V) \equiv \bigwedge_{t \in V} \texttt{Among}(P, \{t\}, l, u)$ enforces that items of $V$ should occur at least $l$ times and at most $u$ times in $P$. To forbid items of $V$ to occur in $P$, $l$ and $u$ must be set to 0.
- *Regular expression constraint*: let $A_{reg}$ be the deterministic finite automaton encoding the regular expression $exp$. $reg(P, exp) \equiv \texttt{Regular}(P, A_{reg})$.

However, constraints that impose a restriction on subsequence relation (see Definition 2), by restricting for example the distance allowed between items in the sequence, cannot be directly combined with our Prefix-Projection global constraint. Indeed, changing the *subsequence relation* requires revisiting the filtering algorithm of the Prefix-Projection global constraint. This is the case for the *gap* constraint, which imposes restrictions on the distance between two consecutive items in the patterns (for more details, see [13]).

### 4.2 Consistency checking and filtering

(a) **Maintaining a local consistency.** SPM is a challenging task due to the exponential number of candidates that should be parsed to find the frequent patterns. For instance, we have $O(n^k)$ potential candidate patterns of length at most $k$ in a sequence of length $n$. Furthermore, the NP-hardness of mining maximal[3] frequent sequences was established in [35] by proving the #P-completeness of the problem of counting the number of maximal frequent sequences. Hence, ensuring *Domain Consistency* (DC) for Prefix-Projection i.e., finding, for every variable $P_j$, a value $d_j \in D(P_j)$, satisfying the constraint is NP-hard.

So, the filtering of Prefix-Projection constraint maintains a consistency lower than DC. This consistency is based on specific properties of the projected databases (see Proposition 3), and anti-monotonicity of the frequency constraint (see Proposition 4), and resembles forward-checking regarding Proposition 3. Prefix-Projection is considered as a global constraint, since all variables share the same internal data structures (i.e., pseudo projected databases $\mathcal{PSDB}$) that are maintained and exploited during filtering.

(b) **Detecting inconsistent values.** The following proposition characterizes values of unassigned (i.e. free) variable $P_{i+1}$ that are consistent with the current partial assignment of variables $\langle P_1, ..., P_i \rangle$. In our approach, we assume a fixed search order based on a lexicographic ordering of variables of $P$, i.e., variable $P_1$ is assigned first, then $P_2$, and so on.

**Proposition 3** (Consistent values) *Let* $\sigma$[4] $= \langle d_1, \ldots, d_i \rangle$ *be a consistent partial assignment of* $i$ *variables* $\langle P_1, \ldots, P_i \rangle$ *(i.e., the current frequent prefix), and* $P_{i+1} \in P$ *be a free variable. A value* $d \in D(P_{i+1})$ *appears in a solution for Prefix-Projection*$(P, SDB, minsup)$ *if and only if* $d$ *is a frequent item in* $SDB|_\sigma$*:*

$$\#\{(sid, \gamma) | (sid, \gamma) \in SDB|_\sigma \wedge \langle d \rangle \preceq \gamma\} \geq minsup$$

[3] A sequential pattern $p$ is maximal if there is no sequential pattern $q$ such that $p \preceq q$.

[4] We indifferently denote $\sigma$ by $\langle d_1, \ldots, d_i \rangle$ or by $\langle \sigma(P_1), \ldots, \sigma(P_i) \rangle$.

*Proof* Let $\sigma$ be a consistent partial assignment and $P_{i+1} \in P$ be a free variable. Suppose that value $d \in D(P_{i+1})$ occurs in $SDB|_\sigma$ more than $minsup$. From proposition 1, we have $sup_{SDB}(concat(\sigma, \langle d \rangle)) = sup_{SDB|_\sigma}(\langle d \rangle)$. Hence, the partial assignment $\sigma \cup \langle d \rangle$ satisfies the constraint, so $d \in D(P_{i+1})$ participates in a solution. □

Proposition 4 shows how the Prefix-Projection global constraint exploits the anti-monotonicity of frequency to prune inconsistent values from the domains of free variables in $P$ w.r.t. the current partial assignment $\sigma$.

**Proposition 4** (Prefix-Projection Filtering rules) *Let $\sigma = \langle d_1, \ldots, d_i \rangle$ be a consistent partial assignment of $i$ variables $\langle P_1, \ldots, P_i \rangle$ (i.e., the current frequent prefix), and $P_{i+1} \in P$ be a free variable. All values $d \in D(P_{i+1})$ that are locally not frequent in $SDB|_\sigma$ can be pruned from the domain of variable $P_{i+1}$. Moreover, these values $d$ can also be pruned from the domains of free variables $P_j$ with $j \in [i+2, \ldots, \ell]$.*

*Proof* Let $\sigma = \langle d_1, \ldots, d_i \rangle$ be a consistent partial assignment of $i$ variables $\langle P_1, \ldots, P_i \rangle$. Let $d \in D(P_{i+1})$ s.t. $\sigma' = concat(\sigma, \langle d \rangle)$. Suppose that $d$ is not frequent in $SDB|_\sigma$. According to proposition 1, $sup_{SDB|_\sigma}(\langle d \rangle) = sup_{SDB}(\sigma') < minsup$, thus $\sigma'$ is not frequent. So, $d$ can be pruned from the domain of $P_{i+1}$.

Suppose that the partial assignment $\sigma$ has been extended to $concat(\sigma, \alpha)$, where $\alpha$ corresponds to the assignment of variables $P_j$ (with $j > i$). If $d \in D(P_{i+1})$ is not frequent, it is straightforward that $sup_{SDB|_\sigma}(concat(\alpha, \langle d \rangle)) \leq sup_{SDB|_\sigma}(\langle d \rangle) < minsup$. Thus, if $d$ is not frequent in $SDB|_\sigma$, it will be also not frequent in $SDB|_{concat(\sigma,\alpha)}$. So, $d$ can be pruned from the domains of $P_j$ with $j \in [i+2, \ldots, \ell]$. □

*Example 19* Consider again our running example of Table 1 with $minsup = 2$. Let $P = \langle P_1, P_2, P_3 \rangle$ with $D(P_1) = \mathcal{I}$ and $D(P_2) = D(P_3) = \mathcal{I} \cup \{\square\}$. Let $\sigma$ be a partial assignment s.t. $\sigma(P_1) = A$, $Prefix - Projection(P, SDB, minsup)$ will remove values $A$ and $D$ from $D(P_2)$ and $D(P_3)$, since the only locally frequent items in $SDB_1|_{\langle A \rangle}$ are $B$ and $C$. Consequently, we obtain $D(P_2) = D(P_3) = \{\square, B, C\}$. Now, if we consider $\sigma(P_2) = B$, $Prefix - Projection(P, SDB, minsup)$ will remove $B$ from $D(P_3)$ since $B$ is not frequent in $SDB_1|_{\langle AB \rangle}$. Finally, we get the two consistent values for the variable $P_3$ which are $\square$ and $C$. So, assigning variable $P_3$ leads to two sequential patterns: $\langle AB\square \rangle = \langle AB \rangle$ and $\langle ABC \rangle$.

(c) **Ensuring the anti-monotonicity of frequency.** Proposition 4 guarantees that any value (i.e. item) $d \in D(P_{i+1})$ present but not frequent in $SDB|_\sigma$ does not need to be considered when extending $\sigma$, thus avoiding searching over it. Clearly, our global constraint encodes the anti-monotonicity of frequency in a simple and elegant way, while existing CP methods for SPM have difficulties to exploit efficiently this property. Indeed, they require explicit mechanisms to prove that a given pattern $p$ is infrequent (i.e., to determine if a conflict is encountered because of the frequency constraint) and to avoid future patterns $p'$ such that $p \preceq p'$. For instance, [6] uses nogoods to avoid generating patterns which are super-sequences of infrequent ones. Negrevergne et al. propose in [18] to adapt the propagator associated to `exists-embedding` global constraint so that it exports the items that still appear after the current prefix. This is

achieved by introducing an auxiliary integer variable $X_i$ for every sequence $s$ in the $SDB$, whose domain represents these items. Finally, to avoid searching over infrequent items, they define a custom search routine (brancher) over the $P$ variables. This brancher first computes the local frequency of each item based on the domains of the $X_i$ variables and only branches on the frequent ones (see Section 4.5 for more details). This makes the integration quite complex. Our approach does not require any extra variables nor a specific branching strategy and tackles the frequency constraint as a single constraint performing a global filtering by exploiting the particular structure of frequent patterns.

**Algorithm 2** PROJECTSDB($SDB$, $ProjSDB$, $\alpha$)

**Data**: $SDB$: initial database; $ProjSDB$: projected sequences; $\alpha$: prefix
**begin**
1 $SDB|_\alpha \leftarrow \emptyset$ ;
2 **for** *each pair* $(sid, start) \in ProjSDB$ **do**
3 $s \leftarrow SDB[sid]$ ;
4 $pos_\alpha \leftarrow 1$; $pos_s \leftarrow start$ ;
5 **while** $(pos_\alpha \leq \#\alpha \wedge pos_s \leq \#s)$ **do**
6 **if** $(\alpha[pos_\alpha] = s[pos_s])$ **then**
7 $pos_\alpha \leftarrow pos_\alpha + 1$ ;
8 $pos_s \leftarrow pos_s + 1$ ;
9 **if** $(pos_\alpha = \#\alpha + 1)$ **then**
10 $SDB|_\alpha \leftarrow SDB|_\alpha \cup \{(sid, pos_s)\}$
11 **return** $SDB|_\alpha$ ;

## 4.3 Building the projected databases

The key issue of our approach lies in the construction of the projected databases. When projecting a prefix, instead of storing the whole suffix as a projected subsequence, one can represent each suffix by a pair $(sid, start)$ where $sid$ is the sequence identifier and $start$ is the starting position of the projected suffix in the sequence $sid$. For instance, let us consider the sequence database of Table 1. As shown in Example 15, $SDB|_{\langle A \rangle}$ consists of 3 suffix sequences: $\{(1, \langle BCBC \rangle), (2, \langle BC \rangle), (3, \langle B \rangle)\}$. By using the *pseudo-projection*, $SDB|_{\langle A \rangle}$ can be represented by the following three pairs: $\{(1, 2), (2, 3), (3, 2)\}$. This is the principle of *pseudo-projection*, adopted in `PrefixSpan`, exploited during the filtering step of our Prefix-Projection global constraint. Algorithm 2 details this principle. It takes as input a set of projected sequences $ProjSDB$ and a prefix $\alpha$. The algorithm processes all the pairs $(sid, start)$ of $ProjSDB$ one by one (line 2), and searches for the lowest location of $\alpha$ in the sequence $s$ corresponding to the $sid$ of that sequence in $SDB$ (lines 6-8).

*Example 20* Let us consider the $SDB_1$ of Table 1 with a prefix $\alpha = \langle AB \rangle$. Algorithm 2 processes the four sequences of $SDB_1$. For the first sequence $s_1$, it looks for the lowest position that matches $A$ ($s_1[0] = A$), and then, the lowest one that matches $B$ ($s_1[1] = B$). Since the two items of $\alpha$ appear in $s_1$, then it saves the first entry of the pseudo-projected database $SDB_1|_{\langle AB \rangle}$ which is (1, 2), where 1 represents the sequence identifier and 2 corresponds to the first position in $s_1$ after the current prefix.

**Proposition 5** (**Time complexity of pseudo-projection**) *In the worst case,* PROJECTSDB *processes all the items of all sequences. So, the time complexity is* $O(\ell \times m)$*, with* $m = \#SDB$ *and* $\ell$ *is the length of the longest sequence in* $SDB$.

The worst case space complexity of pseudo-projection is $O(m)$, since we need to store for each sequence only a pair $(sid, start)$, while for the standard projection the space complexity is $O(m \times \ell)$. Clearly, the pseudo-projection takes much less space than the standard projection.

**Algorithm 3** FILTER-PREFIX-PROJECTION($SDB, \sigma, i, P, minsup$)

**Data**: $SDB$: initial database; $\sigma$: current prefix $\langle \sigma(P_1), \ldots, \sigma(P_i) \rangle$; $minsup$: the minimum support threshold; $\mathcal{PSDB}$: internal data structure of PREFIX-PROJECTION for storing pseudo-projected databases

**begin**

1 **if** $(i \geq 2 \wedge \sigma(P_i) = \square)$ **then**
2 **for** $j \leftarrow i + 1$ **to** $\ell$ **do**
3 $P_j \leftarrow \square$;
4 **return** True;
**else**
5 $\mathcal{PSDB}_i \leftarrow$ PROJECTSDB$(SDB, \mathcal{PSDB}_{i-1}, \langle \sigma(P_i) \rangle)$;
6 **if** $(\#\mathcal{PSDB}_i < minsup)$ **then**
7 **return** False ;
**else**
8 $\mathcal{FI} \leftarrow$ GETFREQITEMS$(SDB, \mathcal{PSDB}_i, minsup)$ ;
9 **for** $j \leftarrow i + 1$ **to** $\ell$ **do**
10 **foreach** $a \in D(P_j)\ s.t. (a \neq \square \wedge a \notin \mathcal{FI})$ **do**
11 $D(P_j) \leftarrow D(P_j) - \{a\}$;
12 **return** True;

FUNCTION GETFREQITEMS ($SDB$, $ProjSDB$, $minsup$) ;

**Data**: $SDB$: the initial database; $ProjSDB$: pseudo-projected database; $minsup$: the minimum support threshold; $ExistsItem$, $SupCount$: internal data structures using a hash table for support counting over items;

**begin**

13 $SupCount[] \leftarrow \{0, ..., 0\}$; $F \leftarrow \emptyset$ ;
14 **for** *each pair* $(sid, start) \in ProjSDB$ **do**
15 $ExistsItem[] \leftarrow \{false, ..., false\}$; $s \leftarrow SDB[sid]$ ;
16 **for** $i \leftarrow start$ **to** $\#s$ **do**
17 $a \leftarrow s[i]$ ;
18 **if** $(\neg ExistsItem[a])$ **then**
19 $SupCount[a] \leftarrow SupCount[a] + 1$; $ExistsItem[a] \leftarrow true$;
20 **if** $(SupCount[a] \geq minsup)$ **then**
21 $F \leftarrow F \cup \{a\}$;
22 **return** $F$;

## 4.4 Filtering algorithm

Algorithm 3 describes the pseudo-code of the filtering algorithm of the Prefix-Projection constraint. It is an incremental filtering algorithm that should be run when one of the $P$ variables is assigned according to a lexicographic ordering of variables of $P$ (i.e. $\langle P_1, P_2, \ldots, P_\ell \rangle$). It exploits internal data-structures enabling to enhance the filtering algorithm. More precisely, it uses an incremental data structure, denoted $\mathcal{PSDB}$, that stores the intermediate pseudo-projections of $SDB$, where $\mathcal{PSDB}_i$ ($i \in [0, \ldots, \ell]$) corresponds to the $\sigma$-projected database of the current partial assignment $\sigma = \langle \sigma(P_1), \ldots, \sigma(P_i) \rangle$ (also called prefix) of variables $\langle P_1, \ldots, P_i \rangle$, and $\mathcal{PSDB}_0 = \{(sid, 1) | (sid, s) \in SDB\}$ is the initial pseudo-projected database of $SDB$ (case where $\sigma = \langle\rangle$). It also uses a hash table indexing the items $\mathcal{I}$ into integers $(1 \ldots \#\mathcal{I})$ for an efficient support counting over items (see function GETFREQITEMS). More precisely, suppose we have $n$ items. In the hash-table, the $n$ items $(i_1, ..., i_n)$ are matched with the sequence of integers $(1..n)$ (i.e., item $i_j$ is matched with integer $j$). It enables to have an immediate access to our data structures. In function GETFREQITEMS, for example, $SupCount[a]$ uses the index of $a$ enabling a direct access.

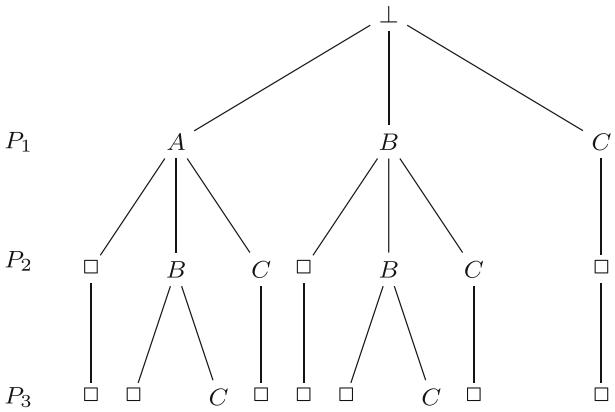



**Fig. 2** The serach tree associated to Section 4.5.1

Algorithm 3 takes as input the database SDB, a minimum support threshold $minsup$, the index $i$ of the last assigned variable $P_i$, the current partial assignment $\sigma = \langle \sigma(P_1), \ldots, \sigma(P_i) \rangle$, and the variables $P$. It starts by checking if the last variable $P_i$ is assigned to $\square$ (line 1). In this case, the end of sequence is reached (since value $\square$ can only appear at the end) and the sequence $\langle \sigma(P_1), \ldots, \sigma(P_i) \rangle$ constitutes a frequent pattern in $SDB$; hence the algorithm sets the remaining $(\ell - i)$ unassigned variables to $\square$ and returns *true* (lines 2-4). Otherwise, the algorithm computes incrementally $\mathcal{PSDB}_i$ from $\mathcal{PSDB}_{i-1}$ by calling function PROJECTSDB (see Algorithm 2). Then, it checks in line 6 whether the current assignment $\sigma$ is a consistent or not for the constraint (see Proposition 2). This is done by computing the size of $\mathcal{PSDB}_i$. If this size is less than $minsup$, we stop growing $\sigma$ and we return *false*.

The lines from 8 to 12 represent the application of proposition 4 on the remaining free variables of $P$. First, the algorithm computes the set of locally frequent items $\mathcal{F}_{\mathcal{I}}$ in $\mathcal{PSDB}_i$ by calling function getFreqItems (line 8). This Function processes all the entries of the pseudo-projected database one by one, counts the number of first occurrences of items $a$ (i.e. $SupCount[a]$) in each entry $(sid, start)$, and keeps only the frequent ones (lines 13-21). This is done by using $ExistsItem$ data structure. After the whole pseudo-projected database has been processed, the frequent items are returned (line 22). Second, lines 9 to 11 remove infrequent items from the domains of free variables $P_j$ with $j \geq (i+1)$, thus avoiding searching over not frequent items.

The next proposition 6 states that FILTER-PREFIX-PROJECTION algorithm establishes domain consistency on the variable $P_{i+1}$ succeeding the current frequent prefix.

**Proposition 6** (**Domain consistency**) *Let $\sigma = \langle d_1, \ldots, d_i \rangle$ be a consistent partial assignment of $i$ variables $\langle P_1, \ldots, P_i \rangle$ (i.e., the current frequent prefix), and $P_{i+1} \in P$ be a free variable. Algorithm* FILTER-PREFIX-PROJECTION *enforces domain consistency of the variable $P_{i+1}$ on the Prefix-Projection constraint.*

*Proof* Since that $\sigma$ is consistent, then from Proposition 2, we have $\#SDB|_\sigma \geq minsup$. Algorithm FILTER-PREFIX-PROJECTION removes from the domain of $P_{i+1}$ all values

which are not frequent in $\#SDB|_\sigma$, and thus keeps only consistent values as provided by Proposition 3. Consequently, FILTER-PREFIX-PROJECTION enforces domain consistency of the variable $P_{i+1}$. □

Proposition 7 states the time and space complexities of FILTER-PREFIX-PROJECTION algorithm.

**Proposition 7** (**complexities of filtering**) *In the worst case, filtering with Prefix-Projection global constraint can be achieved in* $O(m \times \ell + m \times d + \ell \times d)$. *The worst case space complexity of Prefix-Projection is* $O(m \times \ell)$.

*Proof* Let $\ell$ be the length of the longest sequence in $SDB$, $m = \#SDB$, and $d = \#\mathcal{I}$. Computing the pseudo-projected database $\mathcal{PSDB}_i$ can be done in $O(m \times \ell)$: for each sequence $(sid, s)$ of $SDB$, checking if $\sigma$ occurs in $s$ is $O(\ell)$ and there are $m$ sequences. The total complexity of function GETFREQITEMS is $O(m \times (\ell + d))$. Lines 9-11) can be achieved in $O(\ell \times d)$. So, the whole complexity is $O(m \times \ell + m \times (\ell + d) + \ell \times d) = O(m \times \ell + m \times d + \ell \times d)$. The space complexity of the filtering algorithm lies in the storage of the $\mathcal{PSDB}$ internal data structure. For each sequence of length $\ell$, we store only the pseudo-projections of prefixes of that sequence. Since we can have at most $\ell$ prefixes, we have to store $\ell$ pseudo-projected databases in the worst case. In addition, each pseudo-projected database requires $O(m)$. So, the worst case space complexity is $O(m \times \ell)$. □

According to Proposition 6, Prefix-Projection global constraint enumerates all sequential patterns within a backtrack-free search (no fails).

**Proposition 8** (**Backtrack-free**) *Suppose that the variables* $\langle P_1, \ldots, P_\ell \rangle$ *are enumerated following their lexicographic order* $P_1 \rightarrow P_2 \rightarrow \cdots \rightarrow P_\ell$. *Extracting the total number of frequent sequential patterns, noted* $N$, *is backtrack free with a complexity* $O(N \times \ell \times (m \times \ell + m \times d + \ell \times d))$.

*Proof* Using Proposition 6, the adopted ordering of the variables ensures domain consistency on the next variable to enumerate. It follows that extracting the whole sequential patterns is backtrack-free. Since that there is no fail in the search tree, the number of leafs is $N$, and the search tree size is bounded by $O(N \times \ell)$. Since that at each node of the search tree, the solver runs only FILTER-PREFIX-PROJECTION filtering algorithm, the given polynomial complexity, on the number of patterns $N$, follows immediately. □

## 4.5 Running examples

### *4.5.1 Prefix − Projection filtering and search*

Let us take the sequence database given in Table 1 with $minsup = 2$. Let $P = \langle P_1, P_2, P_3 \rangle$ with $D(P_1) = \mathcal{I}$ and $D(P_2) = D(P_3) = \mathcal{I} \cup \{\square\}$. Figure 2 depicts the search tree explored w.r.t. the filtering achieved by Prefix-Projection global constraint. We adopt a variable selection strategy based on the lexicographic ordering of variables: $P_1$ is assigned first, then $P_2$ and finally $P_3$. For value selection strategy, the smallest value in the domain (w.r.t. its lexicographic order) is selected first. Initially, before launching the algorithm

FILTER-PREFIX-PROJECTION, infrequent items are removed from the domains of all variables. Hence, the first variable $P_1$ will be assigned to $A$, $B$ and then $C$. The filtering algorithm is effective if and only if at least the first variable ($P_1$) is assigned. Suppose that $\sigma(P_1) = A$, the algorithm computes the projected database $SDB_1|_{\langle A \rangle}$ from the initial database $SDB_1$ (line 5): $\mathcal{PSDB}_1 = \{(1, 1), (2, 2), (3, 1)\}$. Then, it ensures that the current assignment is consistent (lines 6-7). If the current prefix $\sigma$ appears more than $minsup$ in the database, the frequent items that allow to extend $\sigma = \langle A \rangle$ to a valid sequential patterns are computed (line 8). Consequently, the infrequent items $A$ and $D$ are removed from $D(P_2)$ and $D(P_3)$ (see example 19). There is a special case when a variable is assigned to the symbol $\square$. We suppose that after assigning $P_1$ to $A$, the variable $P_2$ is assigned to $\square$. The pattern is ended by this symbol. Consequently, all variables following $P_2$ must be assigned to $\square$ (line 1). Thus, we get the sequential pattern $\langle A\square\square \rangle = \langle A \rangle$.

### *4.5.2 Comparing prefix-projection vs global-pf*

In this section, we illustrate a comparison between $Prefix - Projection$ global constraint and the Reified Constraint Model proposed in [18] (and detailed in Section 3.2) in terms of modelling and filtering power. For instance, let us take the sequential database given in Table 1 with $minsup = 3$. Let $P = \langle P_1, P_2, P_3, P_4, P_5 \rangle$. Suppose that $\sigma(P_1) = B$ and $D(P_2) = D(P_3) = D(P_4) = D(P_5) = \mathcal{I} \cup \{\square\}$.

Table 2 compares the CP encodings for SPM of the two approaches. As depicted, our approach requires only one constraint and four variables to model both the subsequence and frequency constraints. In contrast, the reified model uses 8 constraints (i.e., $C_1$, ..., $C_8$) and 8 additional variables: 4 boolean variables $S_1$, ..., $S_4$ (one per `exists-embedding` reified global constraint), and 4 auxiliary integer variables $X_1$, ..., $X_4$ used to keep items that appear after the current prefix. Let us detail the behaviour of the filtering and domain reduction done by the two models on the current domains:

`global-p.f`:

- For constraints on the *sequence-end* (see line 1 of the `global-p.f` constraints), as no variable $P_j$ is assigned to $\square$, no reduction is performed on the domains of variables $P$.
- No reduction coming from embedding constraints is perfomed. $S_i$ cannot be reduced because $\langle B \rangle$ appears in all sequences. Consequently, no reduction can be done on variables $P$. Each auxiliary variable $X_i$ ($i \in 1, ..., 4$), takes as a domain the items appearing after the current prefix $\langle B \rangle$: $D(X_1) = \{B, C, \square\}$, $D(X_2) = \{A, B, C, \square\}$, $D(X_3) = \{\square\}$, $D(X_4) = \{C, D, \square\}$. We point out that the domain reduction on $X$ variables is specially dedicated to a custom search routine (brancher) to avoid branching over infrequent items. In this case, only item $C$ is frequent, so, for the next variable, the brancher only branches on this item.
- The frequency constraint does not apply any reduction, since that the $S$ variables are not instantiated.
- Thus, the `global-p.f` does not perform any domain reduction on $P$ variables.

$Prefix - Projection$:

By running the FILTER-PREFIX-PROJECTION filtering algorithm, several inconsistent values will be pruned from the domains of variables $P$. Finally, we get the following new domains: $P_i \in \{C, \square\}$ ($i = 2, ..., 5$).

**Table 2** CP encodings for SPM: Prefix-Projection vs `global-p.f`

| | Prefix-Projection | `global-p.f` |
|---|---|---|
| | $P = \langle P_1, P_2, \ldots, P_5 \rangle$ % unknown pattern | $P = \langle P_1, P_2, \ldots, P_5 \rangle$ % unknown pattern |
| Variables | $D(P_1) = \{B\}$, | $D(P_1) = \{B\}$, |
| & | $D(P_2) = \{A, B, C, D, \square\}$ | $D(P_2) = D(P_3) = D(P_4) = D(P_5) = \{A, B, C, D, \square\}$ |
| Domains | $D(P_3) = \{A, B, C, D, \square\}$ | $S = \langle S_1, S_2, S_3, S_4 \rangle$ % boolean variables for reified covering constraints |
| | $D(P_4) = \{A, B, C, D, \square\}$ | $X = \langle X_1, X_2, X_3, X_4 \rangle$ % Auxiliary variables to avoid infrequent symbols |
| | $D(P_5) = \{A, B, C, D, \square\}$ | $D(X_1) = D(X_2) = D(X_3) = D(X_4) = \{A, B, C, D, \square\}$ |
| | | $\forall j \in 2 \ldots 4, C_{j-1} : P_j = \square \rightarrow P_{j+1} = \square$ % constraints on the sequence-end |
| | | $C_4$: `exists-embedding`$(P, SDB[1], S_1, X_1)$ % embedding constraints |
| Constraints | $Prefix - Projection(P, SDB, minsup)$ | $C_5$: `exists-embedding`$(P, SDB[2], S_2, X_2)$ % in the sense $S_i \longleftrightarrow \exists e \; s.t. P \preceq SDB[i]$ |
| | | $C_6$: `exists-embedding`$(P, SDB[3], S_3, X_3)$ |
| | | $C_7$: `exists-embedding`$(P, SDB[4], S_4, X_4)$ |
| | | $C_8$: $S_1 + S_2 + S_3 + S_4 \geq 3$ % minimum frequency constraint |

## 5 Mining the top-*k* sequential patterns

An important problem common not only to sequential patterns, but to frequent patterns in general, concerns the huge size of the output, from which it is difficult for the user to retrieve relevant informations. Consequently, for practical data mining, it is important to reduce the size of the output. This is often done by fine-tuning the $minsup$ threshold. However, in practice, it is difficult for users to provide an appropriate threshold. To address this issue, it was proposed to redefine the problem of mining sequential patterns as the problem of mining top-$k$ sequential patterns [31], where $k$ is the desired number of frequent patterns to be mined. top-$k$ patterns are the $k$ patterns optimizing an interestingness measure $m$, where $m$ usually refers to the frequency measure or any other measure. In this section, we consider the problem of mining top-$k$ sequential patterns of minimum size $\ell_{min}$. First, we describe a general algorithm for mining top-$k$ sequential patterns based on two steps: initializing a list of patterns $\mathcal{M}$ ordered by the support and raising the support threshold during search. Then, we detail two strategies named `top-k-init-BL` and `top-k-init-IPL` related to the first step of top-$k$ SPM. Both strategies exploit the Prefix-Projection global constraint to explore the pattern search space, however space traversal is different.

**Algorithm 4** `top-k-PP` $(\Phi, SDB, \ell_{min}, \ell, k)$

**Data**: $\Phi$: CSP; $SDB$: initial database; $minsup$: internal variable for the minimum support threshold; $\ell_{min}$: the minimum size of a pattern; $\ell$: the maximum size of a pattern; $k$ ($\geq 1$): an integer.
**Result**: $\mathcal{M}$: the set of all top-$k$ sequential patterns of minimal size $\ell_{min}$.

```
begin
1   M ← ∅ ;
2   minsup ← 0 ;
3   M ← top-k-init(Φ, SDB, minsup, ℓ_min, ℓ, k) ;
4   if (|M| = k) then
5       top-k-SupportRaising(Φ, SDB, M, ℓ_min, k) ;
6   return M

Procedure top-k-SupportRaising (Φ, SDB, M, ℓ_min, k) ;
begin
7   Set minsup to the lowest support of patterns in M ;
8   while (P = solverNext(Φ, SDB, minsup) ∧ |P| ≥ ℓ_min) do
9       add (P,M) ;
10      top-k-update (M, P, minsup, k) ;
11      Set minsup to the lowest support of patterns in M;

Procedure top-k-update (M, P, minsup, k) ;
begin
12  if (sup(P) > minsup) then
13      while ((|M| > k) and (∃ P' ∈ M ∧ sup(P') = minsup)) do
14          remove (P',M) ;
```

**Definition 11** (**top-$k$ sequential patterns**) Let $k$ and $\ell_{min}$ be positive integers. A sequential pattern $p$ is a top-$k$ sequential pattern of minimum size $\ell_{min}$ if there exists no more than $(k-1)$ sequential patterns of minimum size $\ell_{min}$ with supports greater than its support.

*Example 21* For the dataset of Table 1, with $k = 5$ and $\ell_{min} = 1$, the $top$-5 sequential patterns should be: $\langle B \rangle : 4$, $\langle A \rangle : 3$, $\langle C \rangle : 3$, $\langle AB \rangle : 3$, and $\langle BC \rangle : 3$.

In this problem, the minimum support threshold $minsup$ usually used in sequential pattern mining is not known upfront, while the minimum size $\ell_{min}$ can be set to 1 if one is interested in patterns of arbitrary size. So, an algorithm for top-$k$ SPM cannot use a fixed $minsup$ threshold to prune the search space. Therefore, the problem is more difficult. In

consequence, inferences on $minsup$ can be made based on patterns identified so far during search.

### 5.1 The top-$k$-PP algorithm

Algorithm 4 describes a general approach for computing top-$k$ sequential patterns of minimum size $\ell_{min}$. It takes as input a CSP $\Phi$, a sequence database $SDB$, and two strictly positive integers $k$ and $\ell_{min}$. It has as output the set $\mathcal{M}$ of top-$k$ sequential patterns contained in $SDB$.

Let solverNext($\Phi$, $SDB$, $minsup$) be a function which searches for a next valid solution (i.e., valid sequential pattern satisfying the constraints system $\Phi$). top-$k$-PP exploits the Prefix-Projection global constraint to explore the pattern search space. It performs two basic steps: ($i$) initializing a list of patterns $\mathcal{M}$ ordered by the support (line 3), and ($ii$) raising the support threshold during search (line 5). The first step aims at computing the $k$ first sequential patterns (i.e. a set of potential top-$k$ patterns) of minimum size $\ell_{min}$ and insert them in a list of patterns $\mathcal{M}$ ordered according to their support. This internal list is used to maintain the top-$k$ patterns found until now. Our objective in this step is to find the most promising patterns so that the second step will be faster while not missing any top-$k$ pattern.

**Support threshold raising step** Since the $minsup$ threshold is unknown, the mining process should start with $minsup = 0$, raise it progressively during the process, and then use the raised $minsup$ to prune unpromising branches in the search space. Procedure top-k-SupportRaising details the $minsup$-raising process. At the beginning, $minsup$ is set to 0 (line 2). As soon as $k$ sequential patterns with size no less than $\ell_{min}$ are collected, $minsup$ is set to the support of the least frequent pattern $\mathcal{M}$ (line 7). Thereafter, each time a new frequent pattern $P$ of minimum size $\ell_{min}$ is found (line 8), it is inserted in $\mathcal{M}$ (line 9), patterns that are not top-$k$ are removed from $\mathcal{M}$ (cf. procedure top-k-update, line 10), and $minsup$ is raised to the support of the least frequent pattern in $\mathcal{M}$ (line 11). The search stops when no more pattern with a support higher than or equal to $minsup$ can be generated, which means that it has found the top-$k$ sequential patterns of minimum size $\ell_{min}$.

**Algorithm 5** Strategies for initializing $\mathcal{M}$ with the $k$ first feasible solutions

```
   Function top-k-init-BL (Φ, SDB, minsup, ℓ_min, ℓ, k) ;
   begin
1      for i ← 1 to k do
2          if (P = solverNext(Φ, SDB, minsup) ∧ |P| ≥ ℓ_min) then
3              add (P,M) ;
4          else return M ;
       return M
   Function top-k-init-IPL (Φ, SDB, minsup, ℓ_min, ℓ, k) ;
   begin
5      while (|M| < k and ℓ_min ≤ ℓ) do
6          while (P = solverNext(Φ, SDB, minsup) ∧ |P| = ℓ_min) do
7              add (P,M) ;
8              if (|M| > k) then
9                  Set minsup to the lowest support of patterns in M ;
10                 top-k-update (M, P, minsup, k) ;
11         Increase ℓ_min by one and restart search ;
       return M
```

### 5.2 Strategies for initializing the top-$k$ list $\mathcal{M}$

The great interest of $minsup$-raising process of Algorithm 4 is that raising dynamically $minsup$ during search will refine the pruning condition leading to more and more powerful pruning of the search space. However, such approach is largely dependent on the way the $minsup$ is initiated. If $minsup$ is initiated with very low value, the search space will be huge and it is likely to find many patterns with pretty low support. This will lead to the slow raise of $minsup$. Thus, the major issue is how to raise $minsup$ as quickly as possible. As $minsup$ is initialized to the support of the least frequent pattern in $\mathcal{M}$ computed during the first step of Algorithm 4, the way these $k$ first patterns are generated can greatly impact the $minsup$-raising process. In the following, we describe two strategies for initializing top-$k$ list $\mathcal{M}$ (see Algorithm 5).

a) **Base-line strategy.** Function `top-k-init-BL`[5] gives the pseudo-code of the basic strategy. In this strategy, we search for the $k$ first sequential patterns of minimum size $\ell_{min}$ using a depth-first traversal of the search space (lines 1-3). Whenever a new sequential pattern is identified (line 2), it gets inserted to the list of patterns $\mathcal{M}$ ordered according to their support. Once $k$ patterns are found, the exploration of the search tree continue with the procedure `top-k-SupportRaising`. This strategy has poor performances because the search space is too large (see Section 6.8). It is thus necessary to find patterns with high support first in order to raise $minsup$ more quickly.

b) **Generating the most promising patterns.** To raise $minsup$ more quickly, one can first mine the most promising patterns so that high support patterns can be derived earlier, which can be used to prune low-support patterns. We propose a new strategy that first considers patterns of small size and increases gradually the size of sequential patterns to be mined until $k$ patterns are generated. The reasoning behind this strategy is that patterns of small size are likely to be more frequent in $SDB$. Function `top-k-init-IPL`[6] depicts the pseudo-code of our strategy. `top-k-PP-IPL` strategy can be considered as a level-wise approach which generates in each step patterns by incrementing the previous pattern's length. At the beginning, we search for all frequent patterns of size equal to $\ell_{min}$ (lines 6-10), and we stop the extraction when there is no more patterns with a support higher than or equal to $minsup$. If the number of patterns extracted of size $\ell_{min}$ is less than $k$, then we restart search by increasing $\ell_{min}$ by one. The process is repeated until $k$ frequent patterns are found (line 5). Contrary to the base-line strategy, once $k$ patterns are found, we restart the search before continuing with the procedure `top-k-SupportRaising`. We pointed out that the $\ell_{min}$ parameter allows to fix the minimum length of the generated patterns. Although this parameter is considered in the definition of top-$k$ patterns, and therefore as an input parameter of our approach `top-k-PP-IPL`, it has no impact on the provided solutions, but on the efficiency of solving. This is guaranteed by setting $\ell_{min}$ to 1 and adding the minimum size constraint $minSize(P, \ell_{min})$ (see Section 4.1 (c)).

c) **A running example.** Table 3 illustrates the execution of `top-k-init-IPL` on the $SDB$ of Table 1 with $k = 5$ and $\ell_{min} = 1$. In the first iteration (i.e., $\ell_{min} = 1$), four patterns are generated. As $|\mathcal{M}| < 5$, we look for patterns of size 2. A first pattern $\langle BB \rangle$

[5] We note by `top-k-PP-BL` the top-$k$ algorithm using as first step the `top-k-init-BL` strategy.

[6] We note by `top-k-PP-IPL` the top-$k$ algorithm using as first step the `top-k-init-IPL` strategy.

**Table 3** Running `top-k-init-IPL` on Table 1 ($k = 5$ and $\ell_{min} = 1$). The numbers in columns 2 and 3 represent the support of each frequent sequence

| Steps of the algorithm | Candidates | $\mathcal{M}$: list of top-$k$ patterns ordered according to their support | Minsup |
|---|---|---|---|
| Initialization ($\ell_{min} = 1$) | $\langle B\rangle : 4$, $\langle C\rangle : 3$, $\langle A\rangle : 3$, $\langle D\rangle : 1$ | $\langle B\rangle : 4$, $\langle C\rangle : 3$, $\langle A\rangle : 3$, $\langle D\rangle : 1$ | 0 |
| | $\langle BB\rangle : 2$ | $\langle B\rangle : 4$, $\langle C\rangle : 3$, $\langle A\rangle : 3$, $\langle BB\rangle : 2$, $\langle D\rangle : 1$ | 0 |
| Initialization ($\ell_{min} = 2$) | $\langle BC\rangle : 3$ | $\langle B\rangle : 4$, $\langle C\rangle : 3$, $\langle A\rangle : 3$, $\langle BC\rangle : 3$, $\langle BB\rangle : 2$ | 2 |
| | $\langle AB\rangle : 3$ | $\langle B\rangle : 4$, $\langle C\rangle : 3$, $\langle A\rangle : 3$, $\langle BC\rangle : 3$, $\langle AB\rangle : 3$ | 3 |
| Support raising ($\ell_{min}$) $\geq 3$ | $\emptyset$ | $\langle B\rangle$: 4, $\langle C\rangle : 3$, $\langle A\rangle$: 3, $\langle BC\rangle$: 3, $\langle AB\rangle$: 3 | 3 |

with a support 2 is found, and then inserted in $\mathcal{M}$. A second pattern $\langle BC \rangle$ with a support 3 is generated, and again inserted in $\mathcal{M}$. As there are more than ($k = 5$) patterns in $\mathcal{M}$, pattern $\langle D \rangle$ is removed from $M$, and $minsup$ is raised to the support of $\langle BB \rangle$ (i.e., 2). Now, a third pattern of size 2 is generated (i.e., pattern $\langle AB \rangle$ with a support 3). It is inserted in $\mathcal{M}$, pattern $\langle BB \rangle$ is removed from $M$ (i.e. it is not a $top$-5 pattern), and $minsup$ is raised to 3. As there is no more pattern of size 2 with a support higher than or equal to $minsup$, the initialization step of Algorithm 4 terminates. In this example, the mining is completed after the first step because the support threshold is raised to 3 and there are no patterns with support greater than or equal to 3. For comparison, Table 4 shows the results obtained by `top-k-PP-BL` on the $SDB$ of Table 1. As we can see, after the initialization step, $\mathcal{M} = \{\langle B \rangle, \langle BC \rangle, \langle BB \rangle, \langle BBC \rangle, \langle BCB \rangle\}$, and $minsup = 1$. For the second step, as $minsup$ is set to a low value, `top-k-PP-BL` needs to explore many patterns with pretty low support to get the $top$-5 sequential patterns, thus demonstrating the interest of `top-k-init-IPL` strategy as compared to the base-line strategy.

## 6 Experimental evaluation

### 6.1 Benchmark datasets

We chose several real database benchmarks of large size, publicly available from [7]. These datasets have varied characteristics and represent different application domains: web click stream, texts from books, protein sequences and bio-medical text. Table 5 reports for each dataset, the number of sequences $\#SDB$, the number of items $\#\mathcal{I}$, the average size of sequence $avg(\#s)$ and the maximum sequence length. The Leviathan dataset is a conversion

**Table 4** Running `top-k-PP-BL` on Table 1 ($k = 5$ and $\ell_{min} = 1$)

| Steps of the algorithm | Candidates | $\mathcal{M}$: list of top-$k$ patterns ordered according to their support | Minsup |
|---|---|---|---|
| Initialization step | $\langle B \rangle : 4$ | $\langle B \rangle$: 4 | 0 |
| | $\langle BB \rangle : 2$ | $\langle B \rangle : 4, \langle BB \rangle$: 2 | 0 |
| | $\langle BBC \rangle : 2$ | $\langle B \rangle : 4, \langle BB \rangle : 2, \langle BBC \rangle$: 2 | 0 |
| | $\langle BC \rangle : 3$ | $\langle B \rangle : 4, \langle BC \rangle$: 3, $\langle BB \rangle : 2, \langle BBC \rangle : 2$ | 0 |
| | $\langle BCB \rangle : 1$ | $\langle B \rangle : 4, \langle BC \rangle : 3, \langle BB \rangle : 2, \langle BBC \rangle : 2, \langle BCB \rangle$: 1 | 1 |
| Support raising step | $\langle BCBC \rangle : 1$ | | |
| | $\langle BCC \rangle : 1$ | | |
| | $\langle BCD \rangle : 1$ | $\langle B \rangle : 4, \langle BC \rangle : 3, \langle BB \rangle : 2, \langle BBC \rangle : 2, \langle BCB \rangle : 1,$ | |
| | $\langle BA \rangle : 1$ | $\langle BCBC \rangle : 1, \langle BCC \rangle : 1, \langle BCD \rangle : 1, \langle BA \rangle : 1,$ | 1 |
| | $\langle BAB \rangle : 1$ | $\langle BAB \rangle : 1, \langle BABC \rangle : 1, \langle BAC \rangle : 1, \langle BD \rangle : 1$ | |
| | $\langle BABC \rangle : 1$ | | |
| | $\langle BAC \rangle : 1$ | | |
| | $\langle BD \rangle : 1$ | | |
| | $\langle C \rangle : 3$ | $\langle B \rangle : 4, \langle BC \rangle : 3, \langle C \rangle$: 3, $\langle BB \rangle : 2, \langle BBC \rangle : 2$ | 2 |
| | $\langle A \rangle : 3$ | $\langle B \rangle : 4, \langle BC \rangle : 3, \langle C \rangle : 3, \langle A \rangle$: 3, $\langle BB \rangle : 2, \langle BBC \rangle : 2$ | 2 |
| | $\langle AB \rangle : 3$ | $\langle B \rangle : 4, \langle BC \rangle : 3, \langle C \rangle : 3, \langle A \rangle : 3, \langle AB \rangle$: 3 | 3 |

**Table 5** Dataset Characteristics

| Dataset | $\#SDB$ | $\#\mathcal{I}$ | avg ($\#s$) | $\max_{s \in SDB}$ ($\#s$) | Type of data |
|---|---|---|---|---|---|
| Leviathen | 5834 | 9025 | 33.81 | 100 | book |
| Kosarak | 69999 | 21144 | 7.97 | 796 | web click stream |
| FIFA | 20450 | 2990 | 34.74 | 100 | web click stream |
| BIBLE | 36369 | 13905 | 21.64 | 100 | bible |
| Protein | 103120 | 24 | 482 | 600 | protein sequences |
| data-200K | 200000 | 20 | 50 | 86 | synthetic dataset |
| PubMed | 17527 | 19931 | 29 | 198 | bio-medical text |

of the novel Leviathan by Thomas Hobbes (1651) as a sequence database (each word is an item). It contains 5834 sequences and 9025 distinct items. The average number of items per sequence is 33.8. The Kosarak dataset is a very large dataset containing 990,000 sequences of click-stream data obtained from an hungarian news portal. The dataset in its original format can be found at http://fimi.ua.ac.be/data/. For our experiments, we selected a subset of 69,999 sequences. The FIFA dataset contains sequences of click stream from the website of FIFA World Cup 98. It has 2990 distinct items (webpages). The average sequence length is 34.74 items. This dataset was created by processing a part of the web logs from the world cup. The BIBLE dataset is a conversion of the Bible into a sequence database (each word is an item). It contains 36369 sequences and 13905 distinct items. The average length of a sequence is 21.6 items. The Protein dataset contains DNA sequence records. It has 103120 sequences over an alphabet of 24 symbols, and average sequence length equals to 482. The data-200K is a synthetic dataset generated using the IBM dataset generator. These two datasets can be downloaded from http://www-kdd.isti.cnr.it/SMA/. The PubMed dataset is obtained from biomedical texts [3]. We create a corpus using HUGO[7] and Orphanet dictionaries to query the database to get sentences describing genes and rare diseases.

## 6.2 Experimental protocol

The implementation of our approach was coded in C++ using the `Gecode` solver.[8] All experiments were conducted on a machine with a processor Intel X5670 and 24 GB of memory, running the Linux operating system. A time limit of 1 hour has been used. For each dataset, we varied the $minsup$ threshold until the method is not able to complete the extraction of all patterns within the time limit. For example, in the case of the Protein dataset, we chose high values of $minsup$ ($minsup \geq 99.978$ %). Otherwise, a timeout is obtained due to the size of the dataset which exceeds 100,000 sequences. $\ell$ was set to the length of the longest sequence of $SDB$. The implementation and the datasets used in our experiments are available online.[9]

Our objective is (1) to compare our approach to existing CP methods as well as to state-of-the-art methods for SPM in terms of scalability which is a major issue of existing

[7] www.genenames.org

[8] http://www.gecode.org

[9] https://sites.google.com/site/prefixprojection4cp/

CP methods, (2) to show the flexibility of our approach allowing to handle different constraints simultaneously, (3) to evaluate the performance of both top-$k$-PP-BL and top-$k$-PP-IPL against state-of-the-art methods for top-$k$ SPM in terms of efficiency and scalability. We compare our approach (indicated by PP) with:

1. two CP encodings [18], the most efficient CP methods for SPM: global-p.f and decomposed-p.f;
2. state-of-the-art methods for SPM: PrefixSpan and cSpade;
3. SMA [30] for SPM under regular expressions.
4. state-of-the-art methods for top-$k$ SPM: TSP [31] and TKS [8].

We used the author's cSpade[10] and PrefixSpan[11] implementations for SPM. and the SMA implementation[12] for SPM under regular expressions. The two algorithms TKS and TSP used in our experiments are part of the SPMF[13] data mining framework [7]. The implementation[14] of the two CP encodings was carried out in the Gecode solver. All methods have been executed on the same machine.

### 6.3 Comparing PP with existing CP methods for SPM

First we compare PP with the two CP encodings global-p.f and decomposed-p.f (see Section 3.2). CPU times (in logscale for BIBLE, Kosarak and PubMed) of the three methods are shown on Fig. 3. We also report the number of extracted patterns in Fig. 4. First, as expected, the lower *minsup* is, the larger the number of extracted patterns. Second, decomposed-p.f is the least performer method. On all the datasets, it fails to complete the extraction within the time limit for all values of *minsup* we considered. Third, PP largely dominates global-p.f on all the datasets: PP is more than an order of magnitude faster than global-p.f. The gains in terms of CPU times are greatly amplified for low values of *minsup*. On BIBLE (resp. PubMed), the speed-up is 84.4 (resp. 33.5) for *minsup* equal to 1 %. Another important observation that can be made is that, on most of the datasets (except BIBLE and Kosarak), global-p.f is not able to mine for patterns at very low frequency within the time limit. For example on FIFA, PP is able to complete the extraction for values of *minsup* up to 6 % in 1,457 seconds, while global-p.f fails to complete the extraction for *minsup* less than 10 %. Another observation is that Prefix-Projection extracts the sequantial patterns in a backtrack-free manner, where all solutions can be enumerated without any fail. This result is confirmed by the number of failures always equal to zero (backtrack-free).

To complement the results given by Fig. 3, Table 6 reports for different datasets and different values of *minsup*, the number of calls to the propagate routine of Gecode (column 5), and the number of nodes of the search tree (column 6). First, PP explores less nodes than global-p.f. But, the difference is not huge (gains of 45 % and 33 % on FIFA and BIBLE respectively). Second, our approach is very effective in terms of number of

[10] http://www.cs.rpi.edu/zaki/www-new/pmwiki.php/Software/

[11] http://illimine.cs.uiuc.edu/software/prefixspan-mining-sequential-patterns-efficiently-prefix-projected-pattern-growth/

[12] http://www-kdd.isti.cnr.it/SMA/

[13] http://www.philippe-fournier-viger.com/spmf/

[14] https://dtai.cs.kuleuven.be/CP4IM/cpsm/

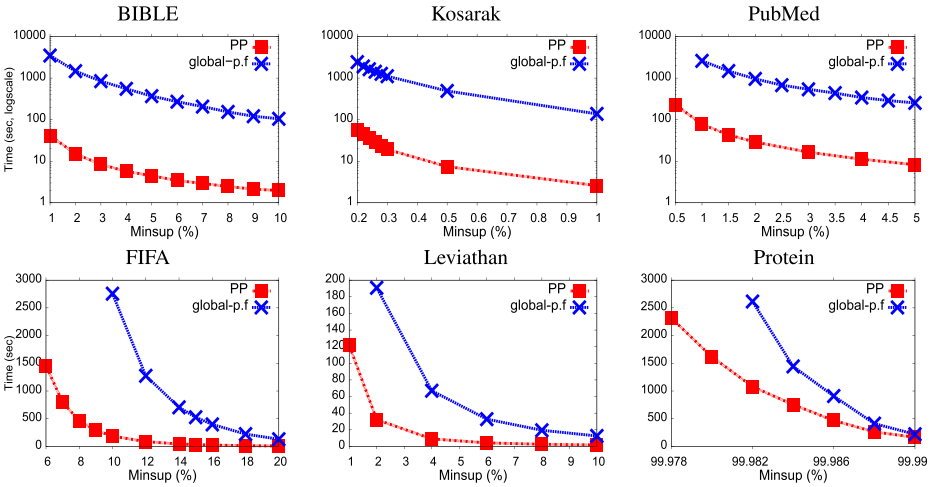



**Fig. 3** Comparing `PP` with `global-p.f` for SPM on real-life datasets: CPU times

propagations. For `PP`, the number of propagations remains small (in thousands for small values of *minsup*) compared to `global-p.f` (in millions). This is due to the huge number of reified constraints used in `global-p.f` to encode the subsequence relation. On the contrary, our Prefix-Projection global constraint does not require any reified constraints nor any extra variables to encode the subsequence relation. Finally, we observe that the number of nodes is slightly different from the number of propagations, because Gecode calls the filtering algorithm each time a new variable is assigned even with the empty symbol. There are some nodes where the filtering algorithm is called many times, because there are more than one variable for which the domain is reduced to a singleton.

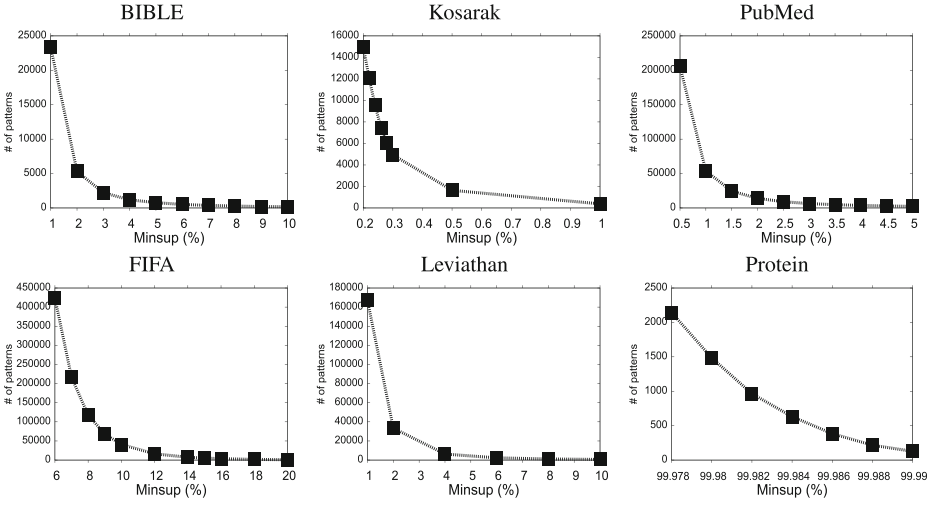



**Fig. 4** Comparing `PP` with `global-p.f` for SPM on real-life datasets: number of patterns

**Table 6** PP vs. global-p.f

| Dataset | *minsup*(%) | #PATTERNS | CPU times (s) | | #PROPAGATIONS | | #NODES | |
|---|---|---|---|---|---|---|---|---|
| | | | PP | global-p.f | PP | global-p.f | PP | global-p.f |
| BIBLE | 10 | 174 | 1.98 | 105.01 | 363 | 4189140 | 235 | 348 |
| | 8 | 274 | 2.47 | 153.61 | 575 | 5637671 | 362 | 548 |
| | 6 | 508 | 3.45 | 270.49 | 1065 | 8592858 | 669 | 1016 |
| | 4 | 1185 | 5.7 | 552.62 | 2482 | 15379396 | 1575 | 2371 |
| | 2 | 5311 | 15.05 | 1470.45 | 11104 | 39797508 | 7048 | 10605 |
| | 1 | 23340 | 41.4 | 3494.27 | 49057 | 98676120 | 31283 | 46557 |
| Kosarak | 1 | 384 | 2.59 | 137.95 | 793 | 8741452 | 482 | 769 |
| | 0.5 | 1638 | 7.42 | 491.11 | 3350 | 26604840 | 2087 | 3271 |
| | 0.3 | 4943 | 19.25 | 1111.16 | 10103 | 56854431 | 6407 | 9836 |
| | 0.28 | 6015 | 22.83 | 1266.39 | 12308 | 64003092 | 7831 | 11954 |
| | 0.24 | 9534 | 36.54 | 1635.38 | 19552 | 81485031 | 12667 | 18966 |
| | 0.2 | 15010 | 57.6 | 2428.23 | 30893 | 111655799 | 20055 | 29713 |
| PubMed | 5 | 2312 | 8.26 | 253.16 | 4736 | 15521327 | 2833 | 4619 |
| | 4 | 3625 | 11.17 | 340.24 | 7413 | 20643992 | 4428 | 7242 |
| | 3 | 6336 | 16.51 | 536.96 | 12988 | 29940327 | 7757 | 12643 |
| | 2 | 13998 | 28.91 | 955.54 | 28680 | 50353208 | 17145 | 27910 |
| | 1 | 53818 | 77.01 | 2581.51 | 110133 | 124197857 | 65587 | 107051 |
| FIFA | 20 | 938 | 8.16 | 129.54 | 1884 | 11649290 | 1025 | 1873 |
| | 18 | 1743 | 13.39 | 222.68 | 3502 | 19736442 | 1922 | 3486 |
| | 16 | 3578 | 24.39 | 396.11 | 7181 | 35942314 | 3923 | 7151 |
| | 14 | 7313 | 44.08 | 704 | 14691 | 65522076 | 8042 | 14616 |
| | 12 | 16323 | 86.46 | 1271.84 | 32820 | 126187396 | 18108 | 32604 |
| | 10 | 40642 | 185.88 | 2761.47 | 81767 | 266635050 | 45452 | 81181 |
| Leviathan | 10 | 651 | 1.78 | 12.56 | 1366 | 2142870 | 849 | 1301 |
| | 8 | 1133 | 2.57 | 19.44 | 2379 | 3169615 | 1487 | 2261 |
| | 6 | 2300 | 4.27 | 32.85 | 4824 | 5212113 | 3008 | 4575 |
| | 4 | 6286 | 9.08 | 66.31 | 13197 | 10569654 | 8227 | 12500 |
| | 2 | 33387 | 32.27 | 190.45 | 70016 | 33832141 | 43588 | 66116 |
| | 1 | 167189 | 121.89 | – | 350310 | – | 217904 | – |
| Protein | 99.99 | 127 | 165.31 | 219.69 | 264 | 26731250 | 172 | 221 |
| | 99.988 | 216 | 262.12 | 411.83 | 451 | 44575117 | 293 | 390 |
| | 99.986 | 384 | 467.96 | 909.47 | 805 | 80859312 | 514 | 679 |
| | 99.984 | 631 | 753.3 | 1443.92 | 1322 | 132238827 | 845 | 1119 |
| | 99.982 | 964 | 1078.73 | 2615 | 2014 | 201616651 | 1284 | 1749 |
| | 99.98 | 2143 | 2315.65 | – | 4485 | – | 2890 | – |

## 6.4 Comparing PP with ad hoc methods for SPM

Our second experiment compares PP with state-of-the-art methods for SPM. Figure 5a shows the CPU times of the three methods. First, cSpade obtains the best performance on all datasets (except on Protein). However, PP exhibits a similar behavior as cSpade,

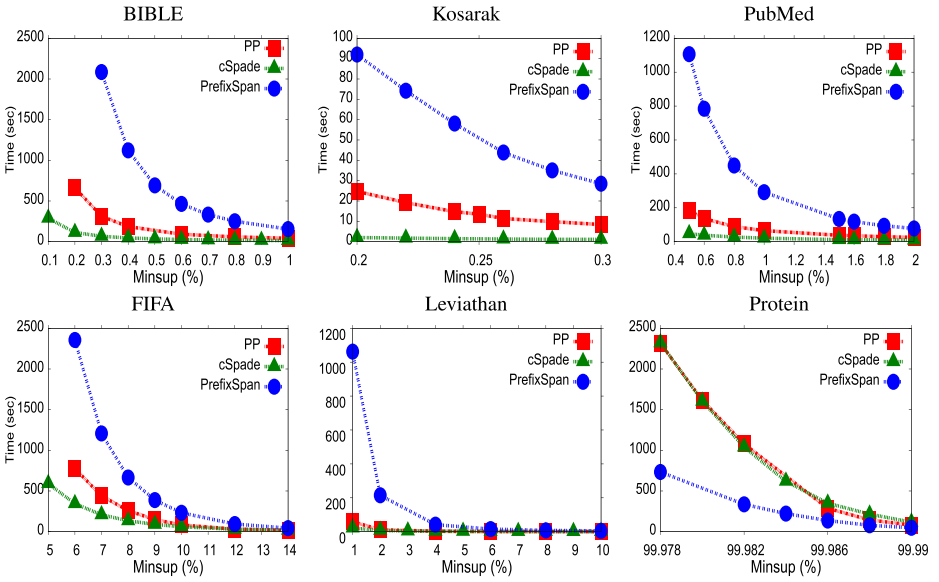



**Fig. 5** Comparing Prefix-Projection with state-of-the-art algorithms for SPM

but it is less faster (not counting the highest values of *minsup*). The behavior of `cSpade` on Protein is due to the vertical representation format that is not appropriated in the case of databases having large sequences and small number of distinct items, thus degrading the performance of the mining process.

Second, `PP` which also uses the concept of projected databases, clearly outperforms `PrefixSpan` on all datasets (except on Protein). This is due to the incremental data structures adopted in PP enabling a quick computation of the projected databases. We point out also another fact: when instantiating some item position in the pattern, PP reduces all the next positions, whereas PrefixSpan reduces only the next position. On FIFA, both approaches remain feasible until 6 % within the time limit. However, Prefix-Projection is 3 times faster than `PrefixSpan` for *minsup* equal to 6 %. These results clearly demonstrate that our approach competes well with state-of-the-art methods for SPM on large datasets and achieves scalability while it is a major issue of existing CP approaches.

### 6.5 SPM under size and item constraints

Our third experiment aims at assessing the interest of pushing simultaneously different types of constraints. We impose on the PubMed dataset usual constraints such as *the minimum frequency* and the *minimum size* constraints and other useful constraints expressing some linguistic knowledge such as *the item constraint*. The goal is to retain sequential patterns which convey linguistic regularities (e.g., gene - rare disease relationships) [3].

- *The size constraint* allows to remove patterns that are too small w.r.t. the number of items (number of words) to be relevant patterns. We tested this constraint with $\ell_{min}$ set to 3.
- *The item constraint* imposes that the extracted patterns must contain the item GENE and the item DISEASE.

As no ad hoc method exists for this combination of constraints we only compare PP with global-p.f. Figure 6 shows the CPU times and the number of sequential patterns extracted with and without constraints. First, pushing simultaneously the two constraints enables to reduce significantly the number of patterns. Moreover, the CPU times for PP decrease slightly whereas for global-p.f (with and without constraints), they are almost the same. This is probably due to the significant number of propagations performed between the $m$ exists-embedding reified global constraints and the two constraints, in order to reach the domain fixpoint (i.e., the variables domain where no value can be filtered). Conversely, in our model, there are only three constraints to propagate for which the fixpoint can be reached more quickly. Second (see Table 7), when considering the two constraints, PP clearly dominates global-p.f (speed-up value up to 51.5). Moreover, the number of propagations performed by PP remains very small as compared to global-p.f.

Figure 6c compares the two methods under the minimum size constraint for different values of $\ell_{min}$, with $minsup$ fixed to 1 %. Once again, PP is always the most performer method (speed-up value up to 53.1). These results also confirm what we observed previously, namely the weak communication between reified global constraints and constraints imposed on patterns (i.e., size and item constraints).

### 6.6 SPM under regular constraints

Our forth experiment compares PP-REG against two variants of SMA: SMA-1P (SMA one pass) and SMA-FC (SMA Full Check). Two datasets are considered from [30]: one synthetic dataset (data-200k), and one real-life dataset (Protein).

– For data-200k, we used two RE:

  1. RE10 $\equiv A^*B(B|C)D^*EF^*(G|H)I^*$
  2. RE14 $\equiv A^*(Q|BS^*(B|C))D^*E(I|S)^*(F|H)G^*R$

– For Protein, we used RE2 $\equiv (S|T)\ .\ (R|K)$ representing *Protein kinase C phosphorylation* (where . represents any symbol).

Figure 7 reports CPU-times comparison. On the synthetic dataset, our approach is very effective. For RE14, our method is more than an order of magnitude faster than SMA. On Protein, the gap between the 3 methods shrinks, but our method remains effective. For the particular case of RE2, the Regular constraint can be substituted by restricting the domain of the first and third variables to $\{S, T\}$ and $\{R, K\}$ respectively (denoted as PP-SRE), thus improving performances.

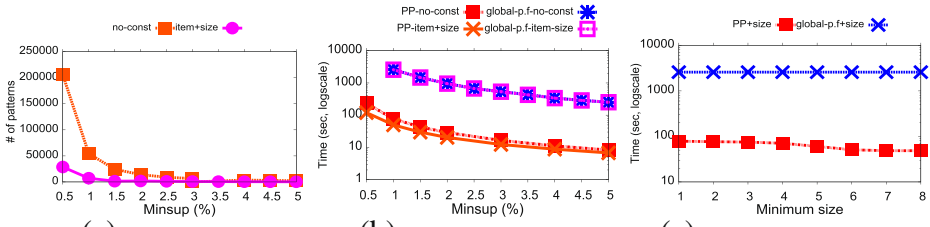



**Fig. 6** Comparing PP with global-p.f under minimum size and item constraints on PubMed

**Table 7** PP vs. global-p.f under minimum size and item constraints

| Dataset | *minsup* (%) | #PATTERNS | CPU times (s) | | #PROPAGATIONS | | #NODES | |
|---|---|---|---|---|---|---|---|---|
| | | | PP | global-p.f | PP | global-p.f | PP | global-p.f |
| PubMed | 5 | 279 | 6.76 | 252.36 | 7878 | 12234292 | 2285 | 4619 |
| | 4 | 445 | 8.81 | 339.09 | 12091 | 16475953 | 3618 | 7242 |
| | 3 | 799 | 12.35 | 535.32 | 20268 | 24380096 | 6271 | 12643 |
| | 2 | 1837 | 20.41 | 953.32 | 43088 | 42055022 | 13888 | 27910 |
| | 1 | 7187 | 49.98 | 2574.42 | 157899 | 107978568 | 52508 | 107051 |

## 6.7 Scalability of prefix-projection approach

To evaluate the scalability of Prefix-Projection global constraint, we used the six real-life datasets and replicated it from 1 to 20 times. We fixed the minimum support threshold at three different values for each dataset. Figure 8 illustrates how the CPU times of Prefix-Projection varies with different replication factors (i.e. dataset sizes). We can see that the CPU times increase (almost) linearly when the replication factor (i.e. number of sequences) increases. This indicates that Prefix-Projection scales well with the size of dataset. For example, for dataset FIFA with *minsup* at 10 %, the CPU time increases from 185 seconds to about 2500 seconds when the number of sequences increases 16 times. Moreover, we can also observe that the CPU times decrease significantly as the minimum support increases. This phenomena can be partially explained by the fact that the number of mined patterns increases dramatically with the decreasing of minimum support. For example, for dataset FIFA with *minsup* at 10 %, there are totally 40,642 sequential patterns, while there are only 5,110 sequential patterns with *minsup* at 15 %.

## 6.8 Top-*k* mining evaluation

In this section, we report a performance study of top-*k* SPM algorithms over the datasets in Table 5. First, we compare the efficiency of the two top-*k*-PP algorithms with two state-of-the-art algorithms for top-*k* SPM, TSP and TKS, for different values of *k*. Second, we study the interest of pushing simultaneously different types of constraints like item and size constraints into top-*k* sequential pattern mining. Third, we evaluate the performance of top-*k*-PP-IPL in terms of scalability. Last, we compare the performance of top-*k*-PP-IPL with PP and cSpade for the scenario where the user would tune

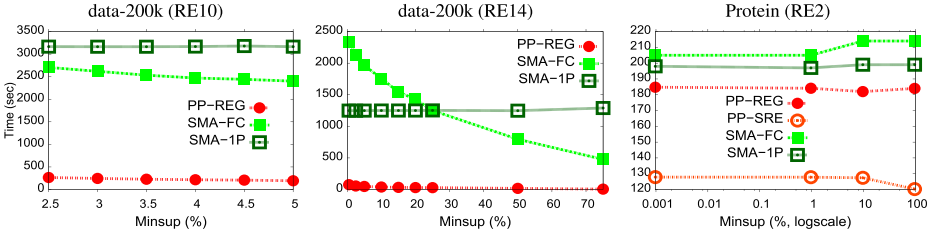



**Fig. 7** Comparing Prefix-Projection with SMA for SPM under RE constraint

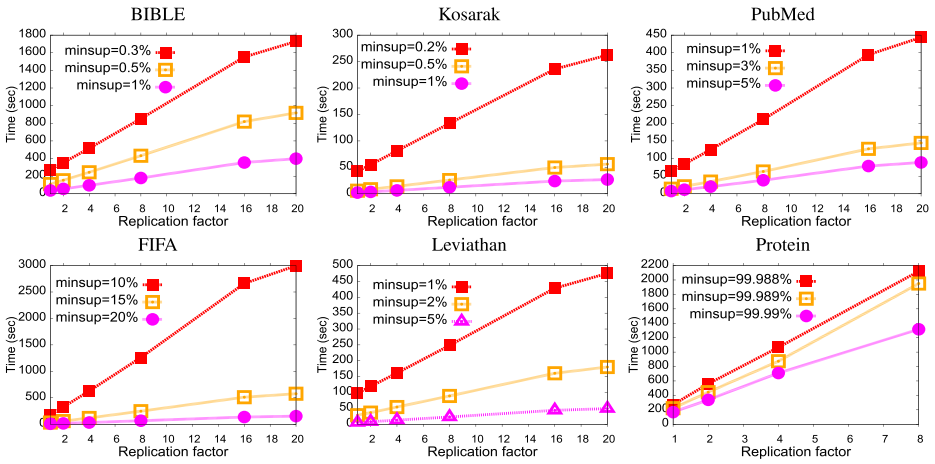



**Fig. 8** Scalability of Prefix-Projection global constraint on real-life datasets

`PP` or `cSpade` with the final minimum support threshold to generate the top-$k$ sequential patterns.

### *6.8.1 Influence of parameter k*

In order to determine the effect of the parameter $k$ on the CPU times of our two `top-k-PP` algorithmsv (`top-k-PP-IPL` and `top-k-PP-BL`), we varied $k$ from 100 to 3000 (except for PubMed where $k$ varies from 100 to 10000). We compare the performance of our two algorithms with `TSP` and `TKS`. Figure 9 and Table 8 show detailed results. We can clearly observe that the greater $k$ is, the longer the CPU time is. It is worth noting that all of the four methods generate the same top-$k$ sequential patterns and obtain the same support of the

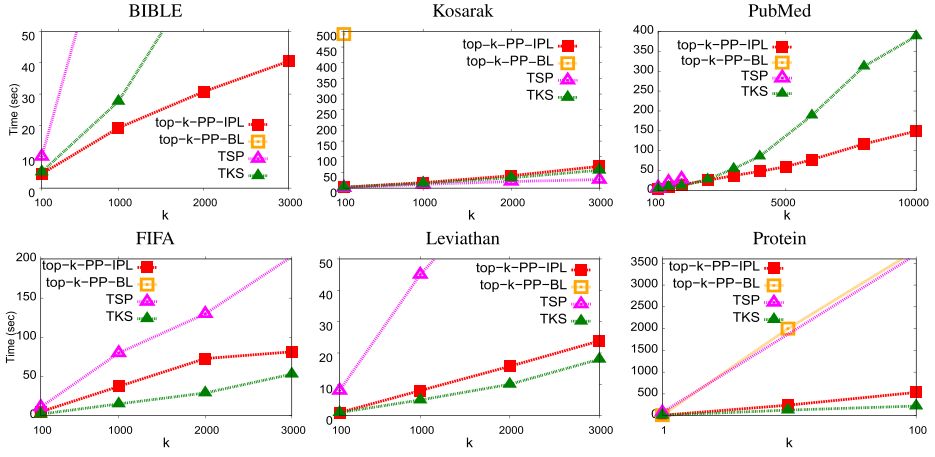



**Fig. 9** Comparing `top-k-PP-IPL` and `top-k-PP-BL` with `TSP` and `TKS` for top-$k$ sequential patterns on real-life datasets

**Table 8** Comparing top-$k$ approaches in terms of CPU times

| Dataset | $k$ | CPU times (s) | | | | Final $minsup$ |
|---|---|---|---|---|---|---|
| | | top-$k$-PP-BL | top-$k$-PP-IPL | TKS | TSP | |
| BIBLE | 1000 | – | 19.22 | 27.67 | 97.707 | 1593 |
| | 2000 | – | 30.75 | 70.786 | 198.212 | 1146 |
| | 3000 | – | 40.49 | 125.578 | 302.580 | 950 |
| Kosarak | 1000 | – | 17.44 | 15.781 | 12.351 | 442 |
| | 2000 | – | 39.57 | 33.958 | 21.490 | 317 |
| | 3000 | – | 69.29 | 57.806 | 27.501 | 261 |
| PubMed | 100 | – | 3.46 | 5.659 | 28.317 | 3745 |
| | 6000 | – | 63.35 | 189.649 | – | 585 |
| | 10000 | – | 126.29 | 388.865 | – | 410 |

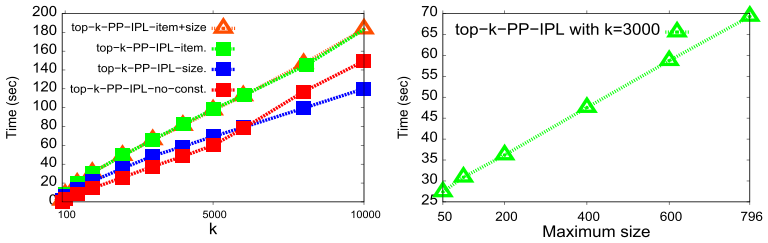
least frequent pattern in top-$k$ list $\mathcal{M}$. From these curves, we can also draw the following observations:

- First, as expected, top-$k$-PP-IPL clearly outperforms the base-line approach top-$k$-PP-BL on all the datasets. On all datasets (except Kosarak and Protein), top-$k$-PP-BL fails to complete the extraction process within the time limit. Table 9 compares the two methods in terms of reached threshold after the initialization step as well as in terms of numbers of generated candidates on Kosarak dataset for different values of $k$. We can see that the number of candidates generated by top-$k$-PP-BL is much higher than the one by top-$k$-PP-IPL. Moreover, top-$k$-PP-IPL reaches higher threshold values after the initialization step. These numbers show that top-$k$-PP-IPL significantly outperforms top-$k$-PP-BL. The Table also shows that the threshold value decreases when $k$ increases. It is because the larger the $k$ value is, the lower the threshold value needs to be to return more patterns.
- Second, top-$k$-PP-IPL largely dominates TSP on all the datasets (except Kosarak where TSP is the best one). Moreover, top-$k$-PP-IPL has better scalability than TSP w.r.t. $k$. On the PubMed dataset, TSP failed to extract all top-$k$ sequential patterns for ($k > 2000$) within the time limit. For $k = 1000$, top-$k$-PP-IPL is an order of magnitude faster than TSP. On the Protein dataset, TSP is not able to complete the extraction even for small values of $k$.
- Third, top-$k$-PP-IPL clearly outperforms TKS on two datasets (speed-up value up to 3.38), while TKS is better on FIFA. On Kosarak and Leviathan, the two methods behave very similarly with a slight advantage for TKS. Recall that TKS is known as the most effective algorithm for mining top-$k$ sequential patterns. However, it uses quite a lot of memory because it needs to consider several "search paths" at the same time in memory to find the top-$k$ patterns. This phenomena is amplified when the length of sequences and the number of items are too large, as for the two datasets BIBLE and PubMed. This partially explains the poor performance of TKS on these datasets.

These results clearly highlight the interest of combining our increasing pattern length strategy with our global constraint for this task which is much harder than sequential pattern mining since $minsup$ has to be raised dynamically.

**Table 9** Comparing top-*k*-PP-IPL with top-*k*-PP-BL in terms of (a) reached threshold after the initialization step and (b) number of candidates generated for different values of *k* on Kosarak dataset

| *k* | *minsup* after initialization step | | # CANDIDATES | | CPU times (s) | | Final *minsup* |
|---|---|---|---|---|---|---|---|
| | top-*k*-PP-BL | top-*k*-PP-IPL | top-*k*-PP-BL | top-*k*-PP-IPL | top-*k*-PP-BL | top-*k*-PP-IPL | |
| 10 | 1 | 4569 | 104 | 17 | 418.13 | 1.280 | 6463 |
| 50 | 1 | 745 | 2836 | 130 | 423.01 | 2.08 | 2407 |
| 100 | 1 | 460 | 43105 | 265 | 491.29 | 2.83 | 1629 |




(a) Results on Pubmed: with (minimum size ($\ell_{min} = 3$) and/or item) constraints and without constraints ($\ell_{min} = 1$).

(b) Varying the value of $\ell$ on Kosarak.

**Fig. 10** Results of `top-k-PP-IPL` under constraints (*CPU times*)

### *6.8.2 Influence of item and size constraints on top-k approach*

To asses the interest of pushing simultaneously different types of constraints in `top-k-PP-IPL` algorithm, we imposed on PubMed dataset both *item* and *minimum size* constraints as those used in Section 6.5. To our best knowledge, there is no existing algorithm that mines top-$k$ sequential patterns for this combination of constraints, we thus evaluate how sensitive `top-k-PP-IPL` is to these two constraints when varying the value of $k$. Figure 12 shows the CPU times of `top-k-PP-IPL` algorithm when the two constraints are pushed simultaneously or individually on the PubMed dataset for $\ell_{min}$ fixed at 3 and $k$ ranging from 100 to 10000. First, `top-k-PP-IPL`'s performance decreases marginally when the two constraints are pushed simultaneously or individually for high values of $k$. Second, when considering separately each constraint, the size constraint seems to have less impact on the CPU times as compared to the item constraint. This suggests that item constraint is more difficult to satisfy. Such results are consistent since the mining task is much harder because we look for the $k$ best sequential patterns satisfying each constraint or their combination, even if the number of patterns decreases. We can also notice that, the CPU time increases linearly as $k$ increases.

We also evaluated how `top-k-PP-IPL` algorithm behaves when the maximum size of pattern $\ell$ vary. Figure 10b shows the results obtained on dataset Kosarak. In the experiments, we fixed $k$ at 3000 and varied $\ell$[15] from 50 to 796. We can clearly observe that the greater $\ell$ is, the longer the CPU time is, because the number of candidate patterns is also increasing. Moreover, the CPU time increases linearly with the linearly increasing of the maximum size of a pattern. For example, at $\ell = 50$, the CPU time was reduced by half. Thus, $\ell$, which also represents the number of variables in our CP model, has an important effect on the CPU time.

### *6.8.3 Scalability of top-k approach*

To evaluate the scalability of `top-k-PP-IPL` algorithm, we followed the same experimental protocol as in Section 6.7. Figure 11 shows the CPU times obtained for two values of $k$: 1000 and 2000 (for Protein, $k$ was set to 50 and 100 in order to complete the extraction within the time limit of one hour). Once again, `top-k-PP-IPL` scales well with the size of dataset.

[15] Recall that in all experiments, $\ell$ was fixed to the maximum size of sequences in the database.

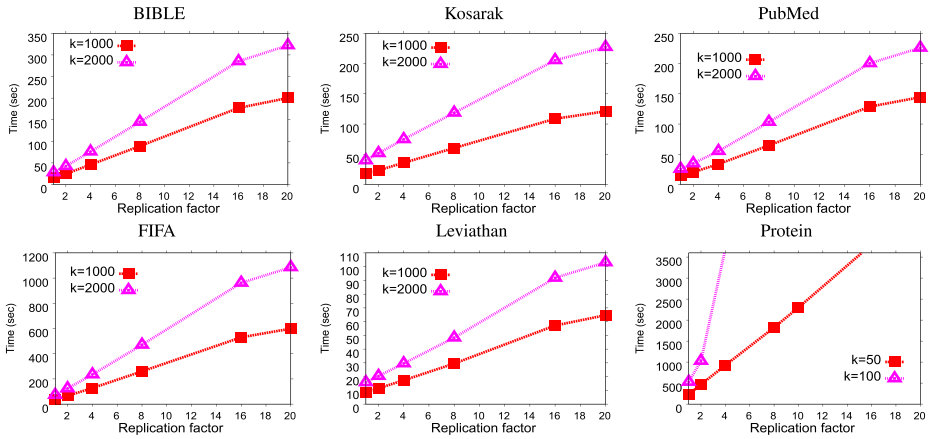



**Fig. 11** Scalability of top-$k$-PP-IPL on real-life datasets

### *6.8.4 Performance comparison of top-k-PP-IPL with PP and cSpade*

Our last experiment compares the efficiency of top-$k$-PP-IPL against PP and cSpade in the case where the two last algorithms are tuned with the optimal minimum support threshold (which is difficult to obtain in practice) so that they can generate the same top-$k$ sequential patterns for a user-specified $k$ value (under a condition of $\ell_{min}$). These optimal *minsup* are obtained by running top-$k$-PP-IPL for each $k$ value. Figure 12 shows the CPU times of the three algorithms. For all datasets, except PubMed, PP and cSpade largely dominate top-$k$-init-IPL. For the PubMed dataset, the curves show that when $k$ is above 4000, top-$k$-PP-IPL starts to outperform PP, while cSpade retains its good performance. These results clearly demonstrate that top-$k$ sequential pattern mining is a much harder problem than sequential pattern mining since *minsup* has to be raised dynamically, starting from 0.

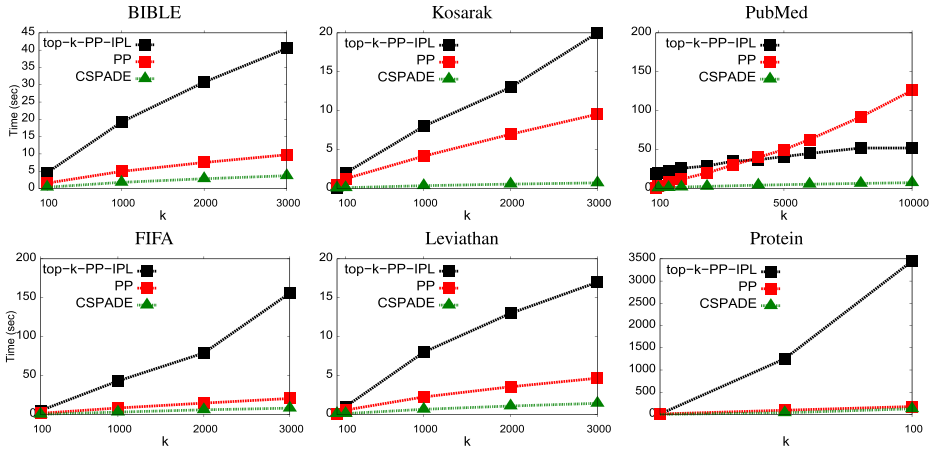



**Fig. 12** Comparing top-$k$-PP-IPL with PP and cSpade

## 7 Conclusion

We have proposed the global constraint Prefix-Projection for sequential pattern mining. Prefix-Projection encapsulates both the subsequence relation imposed over the sequences of the database as well as the frequency constraint into a single constraint. Its filtering strongly relies on the principle of projected databases adopted by pattern growth approaches. Unlike existing CP approaches, to the best of our knowledge, it is the first global constraint which does not require any reified constraints to extract frequent patterns. Prefix-Projection filtering algorithm enforces domain consistency on the variable succeeding the current frequent prefix in polynomial time. When this global constraint is integrated into a CP solver, it enables to handle several constraints simultaneously. Some of them like size, item membership and regular expression are considered in this paper.

In this paper, we have also investigated the task of mining top-$k$ sequential patterns optimizing the frequency measure. We showed how our Prefix-Projection global constraint can be exploited to achieve this task efficiently. More precisely, we have proposed an effective strategy for initializing the top-$k$ patterns with the most promising ones so that high support patterns can be derived earlier. Moreover, we studied the effect of adding constraints for mining top-$k$ sequential patterns as minimum size, item and regular expression constraints, and showed how they can be combined with our Prefix-Projection global constraint.

Experiments performed on several real-life datasets show that our approach clearly outperforms existing CP approaches and competes well with the state-of-the-art methods on large datasets for mining frequent sequential patterns, sequential patterns under various constraints, and top-$k$ sequential patterns. It is worth noticing that our approach achieves scalability while it is a major issue of existing CP approaches.

Although Prefix-Projection is well suited for constraints on patterns, it would require to be adapted to handle other constraints like aggregates, or constraints on subsequence relation like duration or gap constraints [13]. Another direction that deserves to be considered is to extend our approach for sequences of itemsets. Another promising line of research we intend to investigate is to tackle constraints defined on set of patterns [25]. In this setting, the interest of a pattern is evaluated w.r.t. a set of patterns. Examples include finding closed patterns, relevant subgroups [19], and skypatterns [28]. Given a set of measures, skypatterns are patterns based on a Pareto-dominance relation for which no measure can be improved without degrading the others. Such patterns are highly interesting because they do not require any threshold on the measures and the dominance relation gives to the skypatterns a global interest with semantics easily understood by the user. A relevant subgroup consists of patterns that discriminate the positive dataset from the negative one. These patterns are interesting in many domains like disease likelihood prediction, discovering patterns in gene expression data, comparison of protein, and so on. These types of constrains involving set of patterns can be efficiently handled by CP techniques [17, 26] exploiting our global constraint.

## References

1. Agrawal, R., & Srikant, R. (1995). Mining sequential patterns, In Yu, P.S., & Chen, A.L.P. (Eds.) *Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan. pp. 3–14. IEEE Computer Society*. doi:10.1109/ICDE.1995.380415.
2. Ayres, J., Flannick, J., Gehrke, J., & Yiu, T. (2002). Sequential pattern mining using a bitmap representation. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge*

*Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada. pp. 429–435. ACM*. doi:10.1145/775047.775109.
3. Béchet, N., Cellier, P., Charnois, T., & Crémilleux, B. (2012). Sequential pattern mining to discover relations between genes and rare diseases. In *CBMS*.
4. Beldiceanu, N., & Contejean, E. (1994). Introducing global constraints in CHIP. *Journal of Mathematical and Computer Modelling*, *20*(12), 97–123.
5. Cheung, Y., & Fu, A.W. (2004). Mining frequent itemsets without support threshold: With and without item constraints. *IEEE Transactions on Knowledge and Data Engineering*, *16*(9), 1052–1069.
6. Coquery, E., Jabbour, S., Saïs, L., & Salhi, Y. (2012). A sat-based approach for discovering frequent, closed and maximal patterns in a sequence, In Raedt, L.D., Bessière, C., Dubois, D., Doherty, P., Frasconi, P., Heintz, F., & Lucas, P.J.F. (Eds.) *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31, 2012. Frontiers in Artificial Intelligence and Applications, vol. 242, pp. 258–263. IOS Press*. doi:10.3233/978-1-61499-098-7-258.
7. Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C., & Tseng, V. (2014). SPMF: A java Open-Source pattern mining library. *J. of Machine Learning Resea.*, *15*, 3389–3393.
8. Fournier-Viger, P., Gomariz, A., Gueniche, T., Mwamikazi, E., & Thomas, R. (2013). TKS: efficient mining of top-k sequential patterns, In Motoda, H., Wu, Z., Cao, L., Zaïane, O.R., Yao, M., & Wang, W. (Eds.) *Advanced Data Mining and Applications, 9th International Conference, ADMA 2013, Hangzhou, China, December 14-16, 2013, Proceedings, Part I. Lecture Notes in Computer Science, vol. 8346, pp. 109–120. Springer*. doi:10.1007/978-3-642-53914-5_10.
9. Garofalakis, M.N., Rastogi, R., & Shim, K. (2002). Mining sequential patterns with regular expression constraints. *IEEE Trans. Knowl. Data Eng.*, *14*(3), 530–552. doi:10.1109/TKDE.2002.1000341.
10. Guns, T., Nijssen, S., & Raedt, L.D. (2011). Itemset mining: A constraint programming perspective. *Artif. Intell.*, *175*(12-13), 1951–1983. doi:10.1016/j.artint.2011.05.002.
11. Han, J., Wang, J., Lu, Y., & Tzvetkov, P. (2002). Mining top-k frequent closed patterns without minimum support. In *Proceedings of the 2002 IEEE international conference on data mining (ICDM 2002), 9-12 december 2002, maebashi city, Japan* (pp. 211–218).
12. Kemmar, A., Loudni, S., Lebbah, Y., Boizumault, P., & Charnois, T. (2015). PREFIX-PROJECTION global constraint for sequential pattern mining, In Pesant, G. (Ed.) *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9255, pp. 226–243. Springer*. doi:10.1007/978-3-319-23219-5_17.
13. Kemmar, A., Loudni, S., Lebbah, Y., Boizumault, P., & Charnois, T. (2016). A global constraint for mining sequential patterns with GAP constraint. In *Integration of AI and OR techniques in constraint programming - 13th international conference, CPAIOR 2016, banff, AB, Canada, May 29 - June 1, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9676, pP. 198–215. Springer*.
14. Kemmar, A., Ugarte, W., Loudni, S., Charnois, T., Lebbah, Y., Boizumault, P., & Crémilleux, B. (2014). Mining relevant sequence patterns with cp-based framework. In *26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10-12, 2014. pp. 552–559. IEEE Computer Society*. doi:10.1109/ICTAI.2014.89.
15. Li, C., Yang, Q., Wang, J., & Li, M. (2012). Efficient mining of gap-constrained subsequences and its various applications. *ACM Trans. Knowl. Discov. Data*, *6*(1), 2:1–2:39.
16. Métivier, J.P., Loudni, S., & Charnois, T. (2013). A constraint programming approach for mining sequential patterns in a sequence database. In *ECML/PKDD Workshop on languages for data mining and machine learning*.
17. Négrevergne, B., Dries, A., Guns, T., & Nijssen, S. (2013). Dominance programming for itemset mining, In Xiong, H., Karypis, G., Thuraisingham, B.M., Cook, D.J., & Wu, X. (Eds.) *2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013. pp. 557–566. IEEE Computer Society*. doi:10.1109/ICDM.2013.92.
18. Négrevergne, B., & Guns, T. (2015). Constraint-based seque nce mining using constraint programming, In Michel, L. (Ed.) *Integration of AI and OR Techniques in Constraint Programming - 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9075, pp. 288–305. Springer*. doi:10.1007/978-3-319-18008-3_20.
19. Novak, P.K., Lavrac, N., & Webb, G.I. (2009). Supervised descriptive rule discovery: a unifying survey of contrast set, emerging pattern and subgroup mining. *Journal of Machine Learning Research*, *10*, 377–403.

20. Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., & Hsu, M. (2001). Prefixspan: Mining sequential patterns by prefix-projected growth, In Georgakopoulos, D., & Buchmann, A. (Eds.) *Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany. pp. 215–224. IEEE Computer Society*. doi:10.1109/ICDE.2001.914830.
21. Pei, J., Han, J., Mortazavi-Asl, B., & Zhu, H. (2000). Mining access patterns efficiently from web logs, In Terano, T., Liu, H., & Chen, A.L.P. (Eds.) *Knowledge Discovery and Data Mining, Current Issues and New Applications, 4th Pacific-Asia Conference, PADKK 2000, Kyoto, Japan, April 18-20, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1805, pP. 396–407. Springer*. doi:10.1007/3-540-45571-X_47.
22. Pei, J., Han, J., & Wang, W. (2002). Mining sequential patterns with constraints in large databases. In *Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management, McLean, VA, USA, November 4-9, 2002. pp. 18–25. ACM*. doi:10.1145/584792.584799.
23. Pesant, G. (2004). A regular language membership constraint for finite sequences of variables, In Wallace, M. (Ed.) *CP'04. LNCS, vol. 2239, pp. 482–495. Springer*.
24. Pyun, G., & Yun, U. (2014). Mining top-k frequent patterns with combination reducing techniques. *Applied Intelligence*, *41*(1), 76–98.
25. Raedt, L.D., & Zimmermann, A. (2007). Constraint-based pattern set mining. In *Proceedings of the Seventh SIAM International Conference on Data Mining, April 26-28, 2007, Minneapolis, Minnesota, USA. pp. 237–248. SIAM*. doi:10.1137/1.9781611972771.22.
26. Rojas, W.U., Boizumault, P., Loudni, S., Crémilleux, B., & Lepailleur, A. (2014). Mining (soft-) skypatterns using dynamic CSP, In Simonis, H. (Ed.) *Integration of AI and OR Techniques in Constraint Programming - 11th International Conference, CPAIOR 2014, Cork, Ireland, May 19-23, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8451, pp. 71–87. Springer*. doi:10.1007/978-3-319-07046-9_6.
27. Rossi, F., van Beek, P., & Walsh, T. (Eds.) (2006). *Handbook of Constraint Programming*. New York: Elsevier Science Inc.
28. Soulet, A., Raïssi, C., Plantevit, M., & Crémilleux, B. (2011). Mining dominant patterns in the sky, In Cook, D.J., Pei, J., Wang, W., Zaïane, O.R., & Wu, X. (Eds.) *11th IEEE International Conference on Data Mining, ICDM 2011, Vancouver, BC, Canada, December 11-14, 2011. pp. 655–664. IEEE Computer Society*. doi:10.1109/ICDM.2011.100.
29. Srikant, R., & Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements, In Apers, P.M.G., Bouzeghoub, M., & Gardarin, G. (Eds.) *Advances in Database Technology - EDBT'96, 5th International Conference on Extending Database Technology, Avignon, France, March 25-29, 1996, Proceedings. Lecture Notes in Computer Science*, (Vol. 1057 pp. 3–17): Springer. doi:10.1007/BFb0014140.
30. Trasarti, R., Bonchi, F., & Goethals, B. (2008). Sequence mining automata: A new technique for mining frequent sequences under regular expressions. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy. pp. 1061–1066. IEEE Computer Society*. doi:10.1109/ICDM.2008.111.
31. Tzvetkov, P., Yan, X., & Han, J. (2003). In *TSP: mining top-k closed sequential patterns. In: Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003), 19-22 December 2003, Melbourne, Florida, USA. pp. 347–354. IEEE Computer Society*. doi:10.1109/ICDM.2003.1250939.
32. Wang, J., & Han, J. (2004). BIDE: efficient mining of frequent closed sequences, In Özsoyoglu, Z.M., & Zdonik, S.B. (Eds.) *Proceedings of the 20th International Conference on Data Engineering, ICDE 2004, 30 March - 2 April 2004, Boston, MA, USA. pp. 79–90. IEEE Computer Society*. doi:10.1109/ICDE.2004.1319986.
33. Wang, J., Han, J., Lu, Y., & Tzvetkov, P. (2005). TFP: an efficient algorithm for mining top-k frequent closed itemsets. *IEEE Trans. Knowl. Data Eng.*, *17*(5), 652–664. doi:10.1109/TKDE.2005.81.
34. Yan, X., Han, J., & Afshar, R. (2003). Clospan: Mining closed sequential patterns in large databases, In Barbará, D., & Kamath, C. (Eds.) *Proceedings of the Third SIAM International Conference on Data Mining, San Francisco, CA, USA, May 1-3, 2003. pp. 166–177. SIAM*. doi:10.1137/1.9781611972733.15.
35. Yang, G. (2006). Computational aspects of mining maximal frequent patterns. *Theoretical Computer Science*, *362*(1-3), 63–85.
36. Zaki, M.J. (2000). Sequence mining in categorical domains: Incorporating constraints. In *Proceedings of the 2000 ACM CIKM International Conference on Information and Knowledge Management, McLean, VA, USA, November 6-11, 2000. pp. 422–429. ACM*. doi:10.1145/354756.354849.
37. Zaki, M.J. (2001). SPADE: an efficient algorithm for mining frequent sequences. *Machine Learning*, *42*(1/2), 31–60. doi:10.1023/A:100765250231.