EAI Endorsed Transactions
on Scalable Information Systems

Research Article ICST.ORG

A pattern growth-based sequential pattern mining algorithm called prefixSuffixSpan

Kenmogne Edith Belise[1,*], Tadmon Calvin[1], Nkambou Roger[2]

[1]Faculty of Science, Department of Mathematics and Computer Science, LIFA, Po. Box. 67 Dschang, Cameroon
[2]Computer Science Department, University of Québec at Montréal, 201 avenue du président-Kennedy Montréal (Québec) H2X 3Y7 Canada, Knowledge Management laboratory

Abstract

Sequential pattern mining is an important data mining problem widely addressed by the data mining community, with a very large field of applications. The sequence pattern mining aims at extracting a set of attributes, shared across time among a large number of objects in a given database. The work presented in this paper is directed towards the general theoretical foundations of the pattern-growth approach. It helps in-depth understanding of the pattern-growth approach, current status of provided solutions, and direction of research in this area. In this paper, this study is carried out on a particular class of pattern-growth algorithms for which patterns are grown by making grow either the current pattern prefix or the current pattern suffix from the same position at each growth-step. This study leads to a new algorithm called *prefixSuffixSpan*. Its correctness is proven and experimentations are performed.

Received on 07 November 2016; accepted on 05 December 2016; published on 19 January 2017

Keywords: sequence mining, sequential pattern, pattern-growth direction, pattern-growth ordering, search space, pruning, partitioning.

doi:10.4108/eai.18-1-2017.152103

1. Introduction

A sequence database consists of sequences of ordered elements or events, recorded with or without a concrete notion of time. Sequences are common, occurring in any metric space that facilitates either partial or total ordering. Customer transactions, codons or nucleotides in an amino acid, website traversal, computer networks, DNA sequences and characters in a text string are examples of where the existence of sequences may be significant and where the detection of frequent (totally or partially ordered) subsequences might be useful. Sequential pattern mining has arisen as a technology to discover such subsequences. A subsequence, such as buying first a PC, then a digital camera, and then a memory card, if it occurs frequently in a customer transaction database, is a (frequent) sequential pattern.

Sequential pattern mining [5, 13, 14, 16] is an important data mining problem widely addressed by the data mining community, with a very large field of applications such as finding network alarm patterns, mining customer purchase patterns, identifying outer membraneproteins, automatically detecting erroneous sentences, discovering block correlations in storage systems, identifying plan failures, identifying copy-paste and related bugs in large-scale software code, API specification mining and API usage mining from open source repositories, and Web log data mining. Sequential pattern mining aims at extracting a set of attributes, shared across time among a large number of objects in a given database.

The sequential pattern mining problem was first introduced by Agrawal and Srikant [3] in 1995 based on their study of customer purchase sequences, as follows: *Given a set of sequences, where each sequence consists of a list of events (or elements) and each event consists of a set of items, and given a user-specified minimum support threshold min_sup, sequential pattern mining finds all frequent subsequences, that is, the subsequences whose*

[★]Please ensure that you use the most up to date class file, available from EAI at `http://doc.eai.eu/publications/transactions/latex/`
*Corresponding author. Email: ebkenmogne@gmail.com

occurrence frequency in the set of sequences is no less than min_sup.

Since the first proposal of this new data mining task and its associated efficient mining algorithms, there has been a growing number of researchers in the field and tremendous progress [16] has been made, evidenced by hundreds of follow-up research publications, on various kinds of extensions and applications, ranging from scalable data mining methodologies, to handling a wide diversity of data types, various extended mining tasks, and a variety of new applications.

Improvements in sequential pattern mining algorithms have followed similar trend in the related area of association rule mining and have been motivated by the need to process more data at a faster speed with lower cost. Previous studies have developed two major classes of sequential pattern mining methods : Apriori-based approaches [3, 4, 8–10, 17, 21, 23, 25, 26] and pattern growth algorithms [11, 12, 18–20, 22].

The Apriori-based approach form the vast majority of algorithms proposed in the literature for sequential pattern mining. Apriori-like algorithms depend mainly on the Apriori anti-monotony property, which states the fact that any super-pattern of an infrequent pattern cannot be frequent, and are based on a candidate generation-and-test paradigm proposed in association rule mining [1, 2]. This candidate generation-and-test paradigm is carried out by GSP [3], SPADE [26], and SPAM [4]. Mining algorithms derived from this approach are based on either vertical or horizontal data formats. Algorithms based on the vertical data format involve AprioriAll, AprioriSome and DynamicSome [3], GSP [3], PSP [17] and SPIRIT [8], while those based on the horizontal data format involve SPADE [26], cSPADE [25], SPAM [4], LAPIN-SPAM [23], IBM [21] and PRISM [9, 10]. The generation-and-test paradigm has the disadvantage of repeatedly generating an explosive number of candidate sequences and scanning the database to maintain the support count information for these sequences during each iteration of the algorithm, which makes them computationally expensive. To increase the performance of these algorithms constraint driven discovery can be carried out. With constraint driven approaches systems should concentrate only on user specific or user interested patterns or user specified constraints such as minimum support, minimum gap or time interval etc. With regular expressions these constraints are studied in SPIRIT [8].

To alleviate these problems, the pattern-growth approach, represented by FreeSpan [11], PrefixSpan [18, 19] and their further extensions, namely FS-Miner [6], LAPIN [12, 24], SLPMiner [22] and WAP-mine [20], for efficient sequential pattern mining adopts a divide-and-conquer pattern growth paradigm as follows. Sequence databases are recursively projected into a set of smaller projected databases based on the current sequential patterns, and sequential patterns are grown in each projected database by exploring only locally frequent fragments [11, 19]. The frequent pattern growth paradigm removes the need for the candidate generation and prune steps that occur in the Apriori-based algorithms and repeatedly narrows the search space by dividing a sequence database into a set of smaller projected databases, which are mined separately. The major advantage of projection-based sequential pattern-growth algorithms is that they avoid the candidate generation and prune steps that occur in the Apriori-based algorithms. Unlike Apriori-based algorithms, they grow longer sequential patterns from the shorter frequent ones. The major cost of these algorithms is the cost of forming projected databases recursively. To alleviate this problem, a pseudo-projection method is exploited to reduce this cost. Instead of performing physical projection, one can register the index (or identifier) of the corresponding sequence and the starting position of the projected suffix in the sequence. Then, a physical projection of a sequence is replaced by registering a sequence identifier and the projected position index point. Pseudo-projection reduces the cost of projection substantially when the projected database can fit in main memory.

PrefixSpan [18, 19] and FreeSpan [11] differ at the criteria of partitionning projected databases and at the criteria of growing patterns. FreeSpan creates projected databases based on the current set of frequent patterns without a particular ordering (i.e., pattern-growth direction), whereas PrefixSpan projects databases by growing frequent prefixes. Thus, PrefixSpan follows the unidirectional growth whereas FreeSpan follows the bidirectional growth. Another difference between FreeSpan and PrefixSpan is that the pseudo-projection works efficiently for PrefixSpan but not so for FreeSpan. This is because for PrefixSpan, an offset position clearly identifies the suffix and thus the projected subsequence. However, for FreeSpan, since the next step pattern-growth can be in both forward and backward directions from any position, one needs to register more information on the possible extension positions in order to identify the remainder of the projected subsequences.

The work presented in this paper is directed towards the general theoretical foundations of the pattern-growth approach, and does not look into algorithms specific to closed, maximal or incremental sequences, neither does it investigate special cases of constrained, approximate or near-match sequential pattern mining. It aims at enhancing understanding of the pattern-growth approach, current status of provided solutions, and direction of research in this area. To this end, the important key concepts upon which that approach relies, namely pattern-growth direction, pattern-growth ordering, search space pruning and search space partitioning, are revisited. In this paper, this study is carried out on a particular class of pattern-growth algorithms for which patterns are grown by making grow either the current pattern prefix or the current pattern suffix from the same position at each growth-step. This class contains PrefixSpan and involves both unidirectional and bidirectional growth. Thus, it is a generalization of PrefixSpan. However, it does not contain FreeSpan as it makes grow patterns from any position. Stemming from this theoretical study, we design a new algorithm called *prefixSuffixSpan*. We prove its correctness and perform experimentations.

The rest of the paper is organized as follows. Section 2 presents the formal definition of the problem of sequential pattern mining. Section 3 presents the contribution of the paper. Concluding remarks are given in section 4.

2. Problem statement and Notation

The problem of mining sequential patterns, and its associated notation, can be given as follows:

Let $I = \{i_1, i_2, ..., i_n\}$ be a set of literals, termed **items**, which comprise the alphabet. An **itemset** is a subset of items. A **sequence** is an ordered list of itemsets. A sequence s is denoted by $\prec s_1, s_2, ...s_n \succ$, where s_j is an itemset. s_j is also called an **element** of the sequence, and denoted as $(x_1, x_2, ..., x_m)$, where x_k is an item. For brevity, the brackets are omitted if an element has only one item, i.e. element (x) is written as x. An item can occur at most once in an element of a sequence, but can occur multiple times in different elements of a sequence. The number of instances of items in a sequence is called the length of the sequence. A sequence with length l is called an **l-sequence**. The length of a sequence α is denoted $|\alpha|$. A sequence $\alpha = \prec a_1 a_2 ... a_n \succ$ is called **subsequence** of another sequence $\beta = \prec b_1 b_2 ... b_m \succ$ and β a **supersequence** of α, denoted as $\alpha \subseteq \beta$, if there exist integers $1 \leq j_1 < j_2 < \ ... \ < j_n \leq j_m$ such that $a_1 \subseteq b_{j1}, \ a_2 \subseteq b_{j2}, \ ... \ , \ a_n \subseteq b_{jn}$. Symbol ϵ denotes the **empty sequence**.

We are given a database S of input-sequences. A **sequence database** is a set of tuples of the form $\prec sid, s \succ$ where sid is a **sequence_id** and s a sequence. A tuple $\prec sid, s \succ$ is said to contain a sequence α, if α is a subsequence of s. The **support** of a sequence α in a sequence database S is the number of tuples in the database containing α, i.e.

$$support(S, \alpha) = |\{\prec sid, s \succ | \prec sid, s \succ \in S \ \wedge \ \alpha \subseteq s\}|.$$

It can be denoted as $support(\alpha)$ if the sequence database is clear from the context. Given a user-specified positive integer denoted $min\_support$, termed the **minimum support** or the **support threshold**, a sequence α is called a **sequential pattern** in the sequence database S if $support(S, \alpha) \geq min\_support$. A sequential pattern with length l is called an **l-pattern**. Given a sequence database and the $min\_support$ threshold, **sequential pattern mining** is to find the complete set of sequential patterns in the database.

3. Proposed Work

3.1. Pattern-Growth Directions and Orderings

Definition 1 (Pattern-growth direction). A pattern-growth direction is a direction along which patterns could grow. There are two pattern-growth directions, namely left-to-right and right-to-left directions. Do grow a pattern along left-to-right (resp. right-to-left) direction is to add one ore more item to its right (resp. left) hand side.

Definition 2 (Pattern-growth ordering). A pattern-growth ordering is a specification of the order in which patterns should grow. A pattern-growth ordering is said to be unidirectional iff all the patterns should grow along a unique direction. Otherwise it is said to be bidirectional. A pattern-growth ordering is said to be static (resp. dynamic) iff it is fully specified before the beginning of the mining process (resp. iff it is constructed during the mining process).

Definition 3 (Basic-static pattern-growth ordering). A basic-static pattern-growth ordering, also called basic pattern-growth ordering for sake of simplicity, is an ordering which is based on a unique pattern-growth direction, and grow a pattern at the rate of one item per growth-step.

There are two basic-static pattern-growth orderings, namely *left-to-right ordering* (also called *prefix-growth ordering*), which consists in growing a prefix of a pattern at the rate of one item per growth-step at its right hand side, and *right-to-left ordering* (also called *suffix-growth ordering*), which consists in growing a suffix of a pattern at the rate of one item per growth-step at its left hand side.

Definition 4 (Basic-dynamic pattern-growth ordering). A basic-dynamic pattern-growth ordering is an ordering which grow a pattern at the rate of one item per growth-step, and whose pattern-growth direction is determined at the beginning of each growth-step during the mining process. It is denoted $\star$-growth.

Definition 5 (Basic-bidirectional pattern-growth ordering). A basic-bidirectional pattern-growth ordering is an ordering which is based on the two distinct pattern-growth directions, and grow a pattern in each direction at the rate of one item per couple of growth-steps.

There are two basic-bidirectional pattern-growth orderings, namely *prefix-suffix-growth ordering* (i.e. *left-to-right direction* followed by *right-to-left direction*), which consists in growing a pattern at the rate of one item per growth-step during a couple of steps by first growing a prefix (i.e adding of one item at the right-hand side) of that pattern followed by the growing of the corresponding suffix (i.e. adding of one item at the left-hand side), and *suffix-prefix-growth ordering* (i.e *right-to-left direction* followed by *left-to-right direction*), which consists in growing a pattern at the rate of one item per growth-step during a couple of steps by first growing a suffix of that pattern followed by the growing of the corresponding prefix.

Definition 6 (Linear pattern-growth ordering). A linear pattern-growth ordering is a series of compositions of ★-growth, prefix-growth and suffix-growth orderings, and denoted o_0-o_1-o_2 ... o_{n-1}-*growth* for some n, where $o_i \in \{\text{prefix, suffix}, \star\}$ $(0 \le i \le n-1)$. It is said to be static iff $o_i \in \{\text{prefix, suffix}\}$ for all $i \in \{0, 1, 2, \ldots, n-1\}$. Otherwise, it is said to be dynamic.

The o_0-o_1-o_2 ... o_{n-1}-*growth* linear ordering consists in growing a pattern at the rate of one item per growth-step during a series of n growth-steps by growing at step i $(0 \le i \le n-1)$ a prefix (resp. suffix) of that pattern if o_i denotes prefix (resp. suffix). If $o_i \in \{\star\}$, a pattern-growth direction is determined and an item is added to the pattern following that direction. For instance, stemming from the *prefix-suffix-suffix-prefix-growth* static linear ordering, one should grow a pattern in the following order:

- *Growth-step 0*: Add an item to the right hand side of a prefix of that pattern.
- *Growth-step 1*: Add one item to the left hand side of the corresponding suffix of the previous prefix.
- *Growth-step 2*: Repeat step 1.
- *Growth-step 3*: Repeat step 0.
- *Growth-step k* $(k \ge 4)$: Repeat step k mod 4.

The *prefix-suffix-★-prefix-growth* dynamic linear ordering grows patterns as *prefix-suffix-suffix-prefix-growth* ordering except for steps k that satisfy (k mod 4) = 3. During such a particular step, a pattern-growth direction is determined and an item is added to the pattern following that direction.

FreeSpan and PrefixSpan differ at the criteria of growing patterns. FreeSpan creates projected databases based on the current set of frequent patterns without a particular ordering (i.e., pattern-growth direction). Since a length-k pattern may grow at any position, the search for length-(k+1) patterns will need to check every possible combination, which is costly. Because of this, FreeSpan do not follow the linear ordering. However PrefixSpan follows the prefix-growth static ordering as it projects databases by growing frequent prefixes.

Given a database of sequences, an open problem is to find a linear ordering that leads to the best mining performances over all possible linear orderings.

3.2. Search Space Pruning and Partitioning

Definition 7 (Prefix of an itemset). Suppose all the items within an itemset are listed alphabetically. Given an itemset $x = (x_1 x_2 \ \ldots \ x_n)$, another itemset $x\prime = (x\prime_1 x\prime_2 \ \ldots \ x\prime_m)$ $(m \le n)$ is called a prefix of x if and only if $x\prime_i = x_i$ for all $i \le m$. If $m < n$, the prefix is also denoted as $x = (x_1 x_2 \ \ldots \ x_m\_)$.

Definition 8 (The corresponding suffix of a prefix of an itemset). Let $x = (x_1 x_2 \ \ldots \ x_n)$ be a itemset. Let $x\prime = (x_1 x_2 \ \ldots \ x_m)$ $(m \le n)$ be a prefix of x. Itemset $x\prime\prime = (x_{m+1} x_{m+2} \ \ldots \ x_n)$ is called the suffix of x with regards to prefix $x\prime$, denoted as $x\prime\prime = x/x\prime$. We also denote $x = x\prime.x\prime\prime$. Note, if $x = x\prime$, the suffix of x with regards to $x\prime$ is empty. If $1 \le m < n$, the suffix is also denoted as $(\_x_{m+1} x_{m+2} \ \ldots \ x_n)$.

For example, for the itemset $iset = (abcdefgh)$, $(\_efgh)$ is the suffix with regards to the prefix $(abcd\_)$, $iset = (abcd\_).(\_efgh)$, $(abcdef\_)$ is the prefix with regards to suffix $(\_gh)$ and $iset = (abcdef\_).(\_gh)$.

The following definition introduce the dot operator. It permits itemset concatenations and sequence concatenations.

Definition 9 ("." operator). Let e and $e\prime$ be two itemsets that do not contain the underscore symbol (_). Assume that all the items in $e\prime$ are alphabetically sorted after those in e. Let $\gamma = < e_1 \ \ldots \ e_{n-1} a >$ and $\mu = < b e\prime_2 \ \ldots \ e\prime_m >$ be two sequences, where e_i and $e\prime_i$ are itemsets that do not contain the underscore symbol, $a \in \{e, (\_items\ in\ e), (items\ in\ e\_), (\_items\ in\ e\_)\}$ and $b \in \{e\prime, (\_items\ in\ e\prime), (items\ in\ e\prime\_), (\_items\ in\ e\prime\_)\}$. The dot operator is defined as follows.

1. $e.e\prime = ee\prime$
2. $e.(\_items\ in\ e\prime) = (items\ in\ e \cup e\prime)$
3. $e.(items\ in\ e\prime\_) = e(items\ in\ e\prime\_)$
4. $e.(\_items\ in\ e\prime\_) = (items\ in\ e \cup e\prime\_)$
5. $(items\ in\ e\_).e\prime = (items\ in\ e \cup e\prime)$
6. $(items\ in\ e\_).(\_items\ in\ e\prime) = (items\ in\ e \cup e\prime)$
7. $(items\ in\ e\_).(\_items\ in\ e\prime\_) = (items\ in\ e \cup e\prime\_)$
8. $(items\ in\ e\_).(items\ in\ e\prime\_) = (items\ in\ e \cup e\prime\_)$
9. $(\_items\ in\ e).e\prime = (\_items\ in\ e)e\prime$
10. $(\_items\ in\ e).(items\ in\ e\prime\_) = (\_items\ in\ e)(items\ in\ e\prime\_)$
11. $(\_items\ in\ e).(\_items\ in\ e\prime\_) = (\_items\ in\ e \cup e\prime\_)$

12. $(\_items\ in\ e).(\_items\ in\ e\prime) = (\_items\ in\ e \cup e\prime)$
13. $(\_items\ in\ e\_).e\prime = (\_items\ in\ e \cup e\prime)$
14. $(\_items\ in\ e\_).(\_items\ in\ e\prime\_) = (\_items\ in\ e \cup e\prime\_)$
15. $(\_items\ in\ e\_).(items\ in\ e\prime\_) = (\_items\ in\ e \cup e\prime\_)$
16. $(\_items\ in\ e\_).(\_items\ in\ e\prime) = (\_items\ in\ e \cup e\prime)$
17. $\gamma.\mu = \prec e_1\ \dots\ e_{n-1}a.be\prime_2\ \dots\ e\prime_m \succ$

For example, $s = \prec$a(abc)(ac)(efgh)$\succ$ = $\prec$(a).(a_).(_b_).(_c).(a_).(_c).(e_). (_f_).(_g_).(_h)$\succ$ and $s = \prec (a) \succ . \prec (a\_) \succ . \prec (\_b\_) \succ . \prec (\_c) \succ . \prec (a\_) \succ . \prec (\_c) \succ . \prec (e\_) \succ . \prec (\_f\_) \succ . \prec (\_g\_) \succ . \prec (\_h) \succ$.

Definition 10 (Prefix of a sequence). [19] Suppose all the items within an element are listed alphabetically. Given a sequence $\alpha = \prec e_1e_2\ \dots\ e_n \succ$, a sequence $\beta = \prec e\prime_1 e\prime_2\ \dots\ e\prime_m \succ$ $(m \leq n)$ is called a prefix of α if and only if 1) $e\prime_i = e_i$ for all $i \leq m-1$; 2) $e\prime_m \subseteq e_m$; and 3) all the frequent items in $e_m - e\prime_m$ are alphabetically sorted after those in $e\prime_m$. If $e\prime_m \neq \emptyset$ and $e\prime_m \subset e_m$ the prefix is also denoted as $\prec e\prime_1 e\prime_2\ \dots\ e\prime_{m-1}(items\ in\ e\prime_m\_) \succ$.

Definition 11 (The corresponding suffix of a prefix of a sequence). [19] Given a sequence $\alpha = \prec e_1e_2\ \dots\ e_n \succ$. Let $\beta = \prec e_1e_2\ \dots\ e_{m-1}e\prime_m \succ$ $(m \leq n)$ be a prefix of α. Sequence $\gamma = \prec e\prime\prime_m e_{m+1}\ \dots\ e_n \succ$ is called the suffix of α with regards to prefix β, denoted as $\gamma = \alpha/\beta$, where $e\prime\prime_m = e_m - e\prime_m$. We also denote $\alpha = \beta.\gamma$. Note, if $\beta = \alpha$, the suffix of α with regards to β is empty. If $e\prime\prime_m$ is not empty, the suffix is also denoted as $\prec (\_items\ in\ e\prime\prime_m)e_{m+1}\ \dots\ e_n \succ$.

For example, for the sequence $s = \prec a(abc)(ac)(efgh) \succ$, $\prec (ac)(efgh) \succ$ is the suffix with regards to the prefix $\prec a(abc) \succ$, $\prec (\_bc)(ac)(efgh) \succ$ is the suffix with regards to the prefix $\prec$aa$\succ$, $\prec (\_c)(ac)(efgh) \succ$ is the suffix with regards to the prefix $\prec a(ab) \succ$, and $\prec a(abc)(a\_) \succ$ is the prefix with regards to the suffix $\prec (\_c)(efgh) \succ$.

Given three sequences, y, α and $\alpha\prime$, we denote $spc(y, \alpha)$ (resp. $ssc(y, \alpha\prime)$) the shortest prefix (resp. suffix) of y containing α (resp. $\alpha\prime$). If no prefix (resp. suffix) of y contains α (resp. $\alpha\prime$) $spc(y, \alpha)$ (resp. $ssc(y, \alpha\prime)$) does not exist. If the two sequences $spc(y, \alpha)$ and $ssc(y, \alpha\prime)$ exist and do not overlap in sequence y, there exists a sequence $y_{\alpha,\alpha\prime}$ such that $y = spc(y, \alpha).y_{\alpha,\alpha\prime}.ssc(y, \alpha\prime)$. Hence, we have the following definition.

Definition 12 (Canonical sequence decomposition). Given three sequences, y, α and $\alpha\prime$ such that $spc(y, \alpha)$ and $ssc(y, \alpha\prime)$ exist and do not overlap in y, equation $y = spc(y, \alpha).y_{\alpha,\alpha\prime}.ssc(y, \alpha\prime)$ is the canonical decomposition of y following prefix α and suffix $\alpha\prime$. The left, middle and right parts of the decomposition are respectively $spc(y, \alpha)$, $y_{\alpha,\alpha\prime}$ and $ssc(y, \alpha\prime)$.

For example, consider sequence $s = \prec a(abc)(ac)(efgh) \succ$, we have $spc(s, \prec a \succ) = \prec a \succ$, $spc(s, \prec (ab) \succ) = \prec a(ab) \succ$, $spc(s, \prec (ac) \succ) = \prec a(abc) \succ$, $ssc(s, \prec (c)(e) \succ) = \prec (c)(efgh) \succ$, $ssc(s, \prec a \succ) = \prec (ac)(efgh) \succ$, $ssc(s, \prec (bc) \succ) = \prec (\_bc)(ac)(efgh) \succ$, $s = spc(s, \prec (ab) \succ). \prec (\_c)(a\_) \succ .ssc(s, \prec (c)(e) \succ)$ and $s = spc(s, \prec (ac) \succ).\epsilon.ssc(s, \prec a \succ)$. The two sequences $spc(s, \prec (ab) \succ)$ and $spc(s, \prec (ab) \succ$ overlap in sequence s as two sets of the index positions of their items in s are not disjoint.

Stemming from the canonical decompositions of sequences following prefix α and suffix $\alpha\prime$, we define two sets of the sequence database S as follows. We denote $S_{\alpha,\alpha\prime}$ the set of subsequences of S prefixed with α and suffixed with $\alpha\prime$ which are obtained by replacing the left and right parts of canonical decompositions respectively with α and $\alpha\prime$. We have $S_{\alpha,\alpha\prime} = \{\prec sid, \alpha.y_{\alpha,\alpha\prime}.\alpha\prime \succ\ |\ \prec sid, y \succ\ \in\ S$ and $y = spc(y, \alpha).y_{\alpha,\alpha\prime}.ssc(y, \alpha\prime)\}$. We denote $S^{\alpha,\alpha\prime}$ the set of subsequences which are obtained by removing the left and right parts of canonical decompositions. We have $S^{\alpha,\alpha\prime} = \{\prec sid, y_{\alpha,\alpha\prime} \succ\ |\ \prec sid, y \succ\ \in\ S$ and $y = spc(y, \alpha).y_{\alpha,\alpha\prime}.ssc(y, \alpha\prime)\}$. We also have $S = S_{\epsilon,\epsilon}$ and $S = S^{\epsilon,\epsilon}$ as ϵ denotes the empty sequence.

Definition 13 (Extension of the "." operator). Let S be a sequence database and let α be a sequence that may contain the underscore symbol (_). The dot operator is extended as follows. $\alpha.S = \{\prec sid, \alpha.s \succ\ |\ \prec sid, s \succ \in S\}$ and $S.\alpha = \{\prec sid, s.\alpha \succ\ |\ \prec sid, s \succ \in S\}$.

Corollary 1 (Associativity of the "." operator). The dot operator is associative, i.e. given a sequence database S and three sequences α, $\alpha\prime$ and $\alpha\prime\prime$ that may contain the underscore symbol (_), we have.

1. $(\alpha.\alpha\prime).\alpha\prime\prime = \alpha.(\alpha\prime.\alpha\prime\prime)$
2. $\alpha.(\alpha\prime.S) = (\alpha.\alpha\prime).S$
3. $(S.\alpha).\alpha\prime = S.(\alpha.\alpha\prime)$
4. $(\alpha.S).\alpha\prime = \alpha.(S.\alpha\prime)$

Proof. It is straightforward from the dot operation definition. □

We have the following lemmas.

Lemma 1 (The support of z in $S^{\alpha,\alpha\prime}$ is that of its counterpart in S). [15] Given a sequence database S and two sequences α and $\alpha\prime$, for any sequence y prefixed with α and suffixed with $\alpha\prime$, i.e. $y = \alpha.z.\alpha\prime$ for some sequence z, we have support(S, y)=$support(S^{\alpha,\alpha\prime}, z)$.

Proof. Consider the function f from dataset $S_{\alpha,\alpha\prime}$ to dataset $S^{\alpha,\alpha\prime}$ which assigns tuple $\prec sid, y_{\alpha,\alpha\prime} \succ \in S^{\alpha,\alpha\prime}$ to tuple $\prec sid, spc(y, \alpha).y_{\alpha,\alpha\prime}.ssc(y, \alpha\prime) \succ \in S_{\alpha,\alpha\prime}$, where tuple $\prec sid, y \succ \in S$ and sequence y admits a canonical decomposition following prefix α and suffix $\alpha\prime$.

Let's prove that function f is injective. Consider two tuples of S $< sid, y >$ and $< sid\prime, y\prime >$, each having a canonical decomposition following prefix α and suffix $\alpha\prime$. Assume that $f(< sid, spc(y, \alpha).y_{\alpha,\alpha\prime}.ssc(y, \alpha\prime) >) = f(< sid\prime, spc(y\prime, \alpha).y\prime_{\alpha,\alpha\prime}.ssc(y\prime, \alpha\prime) >)$. This implies that $< sid, y_{\alpha,\alpha\prime} >=< sid\prime, y\prime_{\alpha,\alpha\prime} >$, which in turn implies that $sid = sid\prime$. This implies that tuple $< sid, y >$ is equal to $< sid\prime, y\prime >$ as the identifier of any tuple is unique. It comes that $y = y\prime$. Thus $< sid, spc(y, \alpha).y_{\alpha,\alpha\prime}.ssc(y, \alpha\prime) >=< sid\prime, spc(y\prime, \alpha).y\prime_{\alpha,\alpha\prime}.ssc(y\prime, \alpha\prime) >$. Therefore function f is injective.

Let's prove that function f is surjective. Consider $< sid, z_{\alpha,\alpha\prime} >\in S^{\alpha,\alpha'}$, where $< sid, z >$ belongs to S and admits a canonical decomposition following prefix α ans suffix $\alpha\prime$. From the definition of function f, $f(< sid, spc(z, \alpha).z_{\alpha,\alpha\prime}.ssc(z, \alpha\prime) >) =< sid, z_{\alpha,\alpha\prime} >$. This means that $< sid, z_{\alpha,\alpha\prime} >\in S^{\alpha,\alpha'}$ admits a pre-image in $S_{\alpha,\alpha\prime}$. Thus function f is surjective.

Function f is bijective because it is injective and surjective. Let consider a sequence y prefixed with α and suffixed with $\alpha\prime$, i.e. $y = \alpha.z.\alpha\prime$ for some sequence z. Denote $S(y) = \{< sid, s > | < sid, s >\in S \quad \wedge \quad y \subseteq s\}$. Recall that $support(S, y) = |S(y)|$. The definition of $S(y)$ means that it is the set of sequences of S having a canonical decomposition following prefix α and suffix $\alpha\prime$ and containing sequence z in their middle part. It comes that $S(y) = \{< sid, s > | < sid, s >\in S_{\alpha,\alpha\prime} \ \wedge \ z \subseteq s_{\alpha,\alpha\prime}\}$. This implies that $f(S(y)) = \{< sid, s_{\alpha,\alpha\prime} > | < sid, s >\in S_{\alpha,\alpha\prime} \ \wedge \ z \subseteq s_{\alpha,\alpha\prime}\}$. We have $|S(y)| = |f(S(y))|$, as function f is bijective. Therefore $support(S, y) = |S(y)| = |f(S(y))| = |support(S^{\alpha,\alpha'}, z)|$. Hence the lemma. □

Lemma 2 (What does set α.patterns($S^{\alpha,\alpha'}$).$\alpha\prime$ denote for patterns(S) ?). The complete set of sequential patterns of S which are prefixed with α and suffixed with $\alpha\prime$ is equal to $\alpha.patterns(S^{\alpha,\alpha'}).\alpha\prime$, where function *patterns* denotes the complete set of sequential patterns of its unique argument.

Proof. A similar proof is provided in [15]. Let x be a sequence. Assume that $x \in \alpha.patterns(S^{\alpha,\alpha'}).\alpha\prime$. This means that $x = \alpha.z.\alpha\prime$ for some $z \in patterns(S^{\alpha,\alpha'})$. From lemma 1, we have $support(S^{\alpha,\alpha'}, z) = support(S, \alpha.z.\alpha\prime)$. It comes that, x is also a sequential pattern in S as z is a sequential pattern in $S^{\alpha,\alpha'}$. Thus, $\alpha.patterns(S^{\alpha,\alpha'}).\alpha\prime$ is included in the set of sequential patterns of S which are prefixed with α and suffixed with $\alpha\prime$.

Now, assume that x is a sequential pattern of S which is prefixed with α and suffixed with $\alpha\prime$. We have $x = \alpha.z.\alpha\prime$ for some sequence z. From lemma 1, we have $support(S^{\alpha,\alpha'}, z) = support(S, \alpha.z.\alpha\prime)$. It comes that, z is also a sequential pattern in $S^{\alpha,\alpha'}$ as x is a sequential pattern in S. This means that $z \in patterns(S^{\alpha,\alpha'})$. Thus, the complete set of sequential patterns of S which are prefixed with α and suffixed with $\alpha\prime$ is included in $\alpha.patterns(S^{\alpha,\alpha'}).\alpha\prime$. Hence the lemma. □

Lemma 3 (Sequence decomposition lemma). Let $\beta =< e\prime_1 e\prime_2 \ \dots \ e\prime_m >$ be a sequence such that $\beta = \gamma.\mu$ for some non-empty prefix γ and some non-empty suffix μ. Either $\gamma =< e\prime_1 \ \dots \ e\prime_k >$ and $\mu =< e\prime_{k+1} \ \dots \ e\prime_m >$ for some integer k or $\gamma =< e\prime_1 \ \dots \ e\prime_{k-1}\gamma_{k\_} >$, $\mu =< \_\mu_k e\prime_{k+1} \ \dots \ e\prime_m >$, $e\prime_k = \gamma_{k\_} \cup \_\mu_k$, all the items in $\gamma_{k\_}$ are alphabetically before those in $\_\mu_k$ (this implies that $\gamma_{k\_} \cap \_\mu_k = \emptyset$), $\gamma_{k\_} \neq \emptyset$ and $\mu_{k\_} \neq \emptyset$ for some integer k such that $1 \leq k \leq m$.

Proof. Let $\beta =< e\prime_1 e\prime_2 \ \dots \ e\prime_m >= \gamma.\mu$ where $\gamma \neq \epsilon$ and $\mu \neq \epsilon$. According to definitions 10 and 11, $\gamma =< e\prime_1 \ \dots \ e\prime_{k-1}\gamma_{k\_} >$, $\mu =< \_\mu_k e\prime_{k+1} \ \dots \ e\prime_m >$, $e\prime_k = \gamma_{k\_} \cup \_\mu_k$ and all the items in $\gamma_{k\_}$ are alphabetically before those in $\_\mu_k$ for some integer k ($1 \leq k \leq m$). We have the following cases:

- *Case 1*: $k = 1$. This means that $\gamma =< \gamma_{1\_} >$ and $\mu =< \_\mu_1 e\prime_2 \ \dots \ e\prime_m >$. We have $\gamma_{1\_} \neq \emptyset$ as $\gamma \neq \epsilon$. We also have $\_\mu_1 \neq e\prime_1$ as the contrary, i.e. $\_\mu_1 = e\prime_1$, implies that $\gamma = \epsilon$. If $\_\mu_1 = \emptyset$, $\gamma_{1\_} = e\prime_1$ and it comes that $\gamma =< e\prime_1 >$ and $\mu =< e\prime_2 \ \dots \ e\prime_m >$, which corresponds to the first half of the claim of the lemma. Otherwise, we have $\gamma_{1\_} \neq \emptyset$ and $\mu_{1\_} \neq \emptyset$, which leads to the second half of the claim of the lemma

- *Case 2*: $k = m$. This means that $\gamma =< e\prime_1 \ \dots \ e\prime_{m-1}\gamma_{m\_} >$ and $\mu =< \_\mu_m >$. We have $\_\mu_m \neq \emptyset$ as $\mu \neq \epsilon$. We also have $\gamma_{m\_} \neq e\prime_m$ as the contrary, i.e. $\gamma_{m\_} = e\prime_m$, implies that $\mu = \epsilon$. If $\gamma_{m\_} = \emptyset$, $\_\mu_m = e\prime_m$ and it comes that $\gamma =< e\prime_1 \ \dots \ e\prime_{m-1} >$ and $\mu =< e\prime_m >$, which corresponds to the first half of the claim of the lemma. Otherwise, we have $\gamma_{m\_} \neq \emptyset$ and $\_\mu_m \neq \emptyset$, which leads to the second half of the claim of the lemma

- *Case 3*: $k \neq 1, k \neq m$ and $\gamma_{k\_} = \emptyset$. This implies that $\mu_{k\_} = e\prime_k$. It comes that $\gamma =< e\prime_1 \ \dots \ e\prime_{k-1} >$ and $\mu =< e\prime_k \ \dots \ e\prime_m >$, which corresponds to the first half of the claim of the lemma.

- *Case 4*: $k \neq 1$, $k \neq m$ and $\_\mu_k = \emptyset$. This case is similar to case 3. We have $\gamma_{k\_} = e\prime_k$. This implies that $\gamma =< e\prime_1 \ \dots \ e\prime_k >$ and $\mu =< e\prime_{k+1} \ \dots \ e\prime_m >$, which corresponds to the first half of the claim of the lemma.

- *Case 5*: $k \neq 1$, $k \neq m$, $\gamma_{k\_} \neq \emptyset$ and $\_\mu_k \neq \emptyset$. This leads to the second half of the claim of the lemma.

□

Definition 14 (Static and dynamic search-space partitioning). A

search space partition is said to be static i it is fully specified before the beginning of the mining process. It is said to be dynamic iff it is constructed during the mining process.

Lemma 4 (Search-space partitioning based on prefix and/or suffix). We have the following.

1. Let $\{x_1, x_2, \ldots, x_n\}$ be the complete set of length-1 sequential patterns in a sequence database S. The complete set of sequential patterns in S can be divided into n disjoint subsets in two different ways:

   (a) *Prefix-item-based search-space partitioning* [19]: The i-th subset $(1 \le i \le n)$ is the set of sequential patterns with prefix x_i.

   (b) *Suffix-item-based search-space partitioning* [19]: The i-th subset $(1 \le i \le n)$ is the set of sequential patterns with suffix x_i.

2. Let α be a length-l sequential pattern and $\{\beta_1, \beta_2, \ldots, \beta_p\}$ be the set of all length-(l+1) sequential patterns with prefix α. Let $\alpha\prime$ be a length-$l\prime$ sequential pattern and $\{\gamma_1, \gamma_2, \ldots, \gamma_q\}$ be the set of all length-$(l\prime + 1)$ sequential patterns with suffix $\alpha\prime$. We have:

   (a) *Prefix-based search-space partitioning* [19]: The complete set of sequential patterns with prefix α, except for α itself, can be divided into p disjoint subsets. The i-th subset $(1 \le i \le p)$ is the set of sequential patterns prefixed with β_i.

   (b) *Suffix-based search-space partitioning* [19]: The complete set of sequential patterns with suffix $\alpha\prime$, except for $\alpha\prime$ itself, can be divided into q disjoint subsets. The j-th subset $(1 \le j \le q)$ is the set of sequential patterns suffixed with γ_j.

   (c) *Prefix-suffix-based search-space partitioning* [15]: The complete set of sequential patterns with prefix α and suffix $\alpha\prime$, and of length greater or equal to $l + l\prime + 1$, can be divided into p or q disjoint subsets. In the first partition, the i-th subset $(1 \le i \le p)$ is the set of sequential patterns prefixed with β_i and suffixed with $\alpha\prime$. In the second partition, the j-th subset $(1 \le j \le q)$ is the set of sequential patterns prefixed with α and suffixed with γ_j.

Proof. Parts (1.a) and (2.a) of the lemma are proven in [19]. The proof of parts (1.b) and (2.b) of the lemma is similar to the proof of parts (1.a) and (2.a). Thus, we only show the correctness of part (2.c).

Let μ be a sequential pattern of length greater or equal to $l + l\prime + 1$, with prefix α and with suffix $\alpha\prime$, where α is of length l and $\alpha\prime$ is of length $l\prime$. The length-(l+1) prefix of μ is a sequential pattern according to an Apriori principle which states that a subsequence of a sequential pattern is also a sequential pattern. Furthermore, α is a prefix of the length-(l+1) prefix of μ, according to the definition of prefix. This implies that there exists some i $(1 \le i \le p)$ such that β_i is the length-(l+1) prefix of μ. Thus μ is in the i-th subset of the first partition. On the other hand, since the length-k prefix of a sequence is unique, the subsets are disjoint and this implies that μ belongs to only one determined subset. Thus, we have (2.c) for the first partition. The proof of (2.c) for the second partition is similar. Therefore we have the lemma. □

Corollary 2 (Partitioning S with sets $x_i.patterns(S^{x_i,\epsilon})$ and $patterns(S^{\epsilon,x_i}).x_i$). [15] Let $\{x_1, x_2, \ldots, x_n\}$ be the complete set of length-1 sequential patterns in a sequence database S. The complete set of sequential patterns in S can be divided into n disjoint subsets in two different ways:

1. *Prefix-item-based search-space partitioning* : The i-th subset $(1 \le i \le n)$ is $x_i.patterns(S^{x_i,\epsilon})$, where function *patterns* denotes the set of sequential patterns of its unique argument.

2. *Suffix-item-based search-space partitioning* : The i-th subset $(1 \le i \le n)$ is $patterns(S^{\epsilon,x_i}).x_i$.

Proof. According to part 1.(a) of lemma 4, the i-th subset is the set of sequential patterns which are prefixed with x_i. From lemma 2, this subset is $x_i.patterns(S^{x_i,\epsilon})$. Similarly, according to part 1.(b) of lemma 4, the i-th subset is the set of sequential patterns suffixed with x_i. From lemma 2, this subset is $patterns(S^{\epsilon,x_i}).x_i$. □

Lemma 5 (A linear ordering induces a recursive pruning and partitioning). [15] A linear ordering induces a recursive pruning and partitioning of the search space. The recursive partitioning is static if the linear ordering is static and dynamic otherwise.

Proof. Let us consider the initial sequence database S, two integer numbers l and $l\prime$, a length-l sequential pattern α, a length-$l\prime$ sequential pattern $\alpha\prime$, and a linear ordering $L_0 = o_0\text{-}o_1\text{-}o_2 \ldots o_{n-1}\text{-}growth$. Note that $\epsilon.S^{\epsilon,\epsilon}.\epsilon = S$ is the starting database of the recursive pruning and partitioning of the search space. In the following, we show how L_0 induces a recursive pruning and partitioning of $\alpha.S^{\alpha,\alpha\prime}.\alpha\prime$.

- *Case 1:* $o_0 \in \{\text{prefix}\}$. Let $\{\beta_1.\alpha\prime, \beta_2.\alpha\prime, \ldots, \beta_p.\alpha\prime\}$ be the set of all length-$(l + l\prime + 1)$ sequential patterns with respect to database $\alpha.S^{\alpha,\alpha\prime}.\alpha\prime$, prefixed with α and suffixed with $\alpha\prime$. From lemma 3, either $\beta_i = \alpha. \prec (x_i) \succ$ or $\beta_i = \alpha. \prec$

$(\_x_i) \succ$, where x_i is an item and $1 \leq i \leq p$. This implies that $X = \{\prec x_1 \succ, \ \prec x_2 \succ, \ \ldots, \ \prec x_p \succ\}$ is the complete set of length-1 sequential patterns with respect to database $S^{\alpha,\alpha'}$. It comes that any item that does not belong to X is not frequent with respect to $S^{\alpha,\alpha'}$. Thus, any sequence that contains an item that does not belong to X is not frequent with respect to $S^{\alpha,\alpha'}$ according to an Apriori principle which states that any supersequence of an infrequent sequence is also infrequent. Because of this, all the infrequent items with respect to $S^{\alpha,\alpha'}$ are removed from the z part (also called the middle part) of all sequence $\alpha.z.\alpha\prime \in \alpha.S^{\alpha,\alpha'}.\alpha\prime$. This pruning step leads to a new sequence database $\alpha.S\prime^{\alpha,\alpha'}.\alpha\prime$ whose middle parts of sequences do not contain infrequent items with respect to $S^{\alpha,\alpha'}$. Then, $\alpha.S\prime^{\alpha,\alpha'}.\alpha\prime$ is partitioned according to part (2.c) of lemma 4. The i-th sub-database $(1 \leq i \leq p)$ of $\alpha.S\prime^{\alpha,\alpha'}.\alpha\prime$, denoted $\alpha.x_i.S\prime^{\alpha.x_i,\alpha'}.\alpha\prime$, is the set of subsequences of $\alpha.S\prime^{\alpha,\alpha'}.\alpha\prime$ with prefix $\beta_i = \alpha.x_i$ and with suffix $\alpha\prime$. Each sub-database is in turn recursively pruned and partitioned according to $L_1 = o_1\text{-}o_2 \ \ldots \ o_{n-1}\text{-}growth$ linear ordering.

- *Case 2:* $o_0 \in \{\text{suffix}\}$. Let $\{\alpha.\gamma_1, \alpha.\gamma_2, \ \ldots, \ \alpha.\gamma_p\}$ be the set of all length-$(l + l\prime + 1)$ sequential patterns with respect to database $\alpha.S^{\alpha,\alpha'}.\alpha\prime$, prefixed with α and suffixed with $\alpha\prime$. From lemma 3, either $\gamma_i = \prec (x_i) \succ .\alpha\prime$ or $\gamma_i = \prec (x_i\_) \succ .\alpha\prime$ $(1 \leq i \leq p)$. As in case 1, $\alpha.S\prime^{\alpha,\alpha'}.\alpha\prime$ is partitioned according to part (2.c) of lemma 4. The i-th sub-database $(1 \leq i \leq p)$ of $\alpha.S\prime^{\alpha,\alpha'}.\alpha\prime$, denoted $\alpha.S\prime^{\alpha,x_i.\alpha'}.x_i.\alpha\prime$, is the set of subsequences of $\alpha.S\prime^{\alpha,\alpha'}.\alpha\prime$ with prefix α and with suffix $\gamma_i = x_i.\alpha\prime$. As in case 1, each sub-database is in turn recursively pruned and partitioned according to $L_1 = o_1\text{-}o_2 \ \ldots \ o_{n-1}\text{-}growth$ linear ordering.

- *Case 3:* $o_0 \in \{\star\}$. A pattern-growth direction is determined during the mining process. Then, $\alpha.S^{\alpha,\alpha'}.\alpha\prime$ is recursively pruned and partitioned as in case 1 if the determined direction is *left-to-right* and as in case 2 otherwise.

From definitions 6 and 14 it is easy to see that the recursive partitioning is static if the linear ordering is static and dynamic otherwise. □

3.3. A Pattern-growth algorithm based on linear orderings

In this section, we translate the study made in sections 3.1 and 3.2 into a function called *prefixSuffixSpan*. It is presented in algorithm 1. The initial call of *prefixSuffixSpan* (1) takes as arguments the initial database S, the empty sequence ϵ as the current prefix and suffix values, a linear ordering $o = o_0\text{-}o_1\text{-}o_2 \ldots o_{n-1}\text{-}growth$, the index of the pattern-growth direction o_0 in o, i.e. 0, and the support threshold, (2) searches for the complete list $X = \{x_1, \ x_2, \ \ldots, \ x_p\}$ of all the length-1 sequential patterns of S, (4) saves $\alpha.x_i.\alpha\prime$ as a new sequential pattern for each pattern x_i found, assuming that the current prefix and suffix values are respectively α and $\alpha\prime$. (5) constructs, following corollary 2, a new database $S^{x_i,\epsilon}$ (resp. S^{ϵ,x_i}) for each length-1 pattern x_i found if $o_0 = prefix$ (resp. $o_0 = suffix$), and (6) makes a recursive call per new constructed database with arguments (6.1) $\alpha.x_i$ as the new current prefix value if $o_0 = prefix$ and α otherwise, (6.2) $x_i.\alpha\prime$ as the new current suffix value if $o_0 = suffix$ and $\alpha\prime$ otherwise, (6.3) o as the pattern-growth ordering, (6.4) the index of the pattern-growth direction o_1 in o, i.e. 1, and (6.5) the support threshold.

Function *prefixSuffixSpan* recursively generates sub-databases from a partition of the current database following corollary 2. We consider that database S is of depth 0. A generated database is of depth d if it has been constructed using d length-1 patterns. Such a database is denoted $S(x_1, \ x_2, \ ..., \ x_d)$, where $x_1, \ x_2, \ ... \ , \ x_d$ are the length-1 patterns used to construct that database step by step in this order. In the behaviour of *prefixSuffixSpan*, $S(x_1)$ is generated from the initial database S, $S(x_1, x_2)$ is generated from $S(x_1)$, more generally $S(x_1, \ x_2, \ ..., \ x_i)$ is generated from $S(x_1, \ x_2, \ ..., \ x_{i-1})$ where $i < d$ and $S(x_1, \ x_2, \ ..., \ x_d)$ is generated from $S(x_1, \ x_2, \ ..., \ x_{d-1})$. Thus $S(x_1, \ x_2, \ ..., \ x_d)$ is consructed in d steps, where step 1 corresponds to the construction of $S(x_1)$ from S and step i corresponds to the construction of $S(x_1, \ x_2, \ ..., \ x_i)$ from $S(x_1, \ x_2, \ ..., \ x_{i-1})$. In terms of *prefixSuffixSpan* calls, step 1 corresponds to the initial function call $\textsc{prefixSuffixSpan}(S, \ \epsilon, \ \epsilon, \ o, \ 0, \ min\_support)$ and step i corresponds to the function call $\textsc{prefixSuffixSpan}(S(x_1, \ x_2, \ ..., \ x_{i-1}), \ \alpha, \ \alpha\prime, \ o, \ i - 1, \ min\_support)$. We consider that this last function call is of depth $i - 1$. Similarly, we consider that the initial call is of depth 0. For sake of simplicity, we assume that if $d = 0$, $S(x_1, \ x_2, \ ..., \ x_d)$ denotes the initial sequence database S, i.e. $S(x_1, \ x_2, \ ..., \ x_d) = S$. We have the following lemmas and corollaries.

Lemma 6 (Veracity of algorithm 2). Given a depth d call $\textsc{prefixSuffixSpan}(S(x_1, \ x_2, \ ..., \ x_d), \ \alpha, \ \alpha\prime, \ o, \ d, \ min\_support)$, where $o = o_0\text{-}o_1\text{-}o_2 \ldots o_{d+1} \ldots o_{n-1}\text{-}growth$ and o_{-1} denotes either "prefix" or "suffix", the function call $\textsc{prefixSuffixArguments}(\alpha, \alpha\prime, o_{d-1}, X)$ of algorithm 2, where $X = \{x_1, \ x_2, \ ..., \ x_d\}$, provides the values of prefix α and suffix $\alpha\prime$.

Proof. We prove the result by induction on the value of depth. If depth $d = 0$, we have $S(x_1, \ x_2, \ ..., \ x_d) = S$ and the result is true as the first *prefixSuffixSpan* call takes the empty sequence ϵ as the current

Algorithm 1 (prefixSuffixSpan) PrefixSuffix-growth sequential pattern mining. The initial call is PREFIXSUFFIXSPAN($S, \epsilon, \epsilon, o, 0, min\_support$)

```
1: function PREFIXSUFFIXSPAN(Dataset $S$, Prefix $\alpha$, Suffix $\alpha\prime$, Ordering $o$, int $position$, float $min\_support$)
2:     $direction \leftarrow$ GETTHEGROWTHDIRECTION($o$, $position$)
3:     if (direction =='*') then
4:         $direction \leftarrow$ GETTHEGROWTHDIRECTION()
5:     end if
6:     $X \leftarrow$ FINDALLLENGTHONEPATTERN($S$, $direction$, $min\_support$)
7:     Comment: $X = \{x_1, x_2, \ldots, x_p\}$ is obtained by scanning all the sequences of $S$ following the pattern-growth direction $direction$. Length-1 pattern $x_i$ is either of the form $<\_item>$ or $<item\_>$ or $<\_item\_>$ or $<item>$, where $item$ denotes an item.
8:     Comment: The following loop Append successively $x_i$ and $\alpha\prime$ to $\alpha$ to form a sequential pattern.
9:     for all $x_i \in X$ do
10:        SAVESEQUENTIALPATTERN($\alpha.x_i.\alpha\prime$)
11:    end for
12:    $nextPos \leftarrow$ GETTHENEXTPOSITION($o, position$).
13:    if ($direction ==$ "$prefix$") then
14:        for all $x_i \in X$ do
15:            PREFIXSUFFIXSPAN($S^{x_i,\epsilon}$, $\alpha.x_i$, $\alpha\prime$, $o$, $nextPos$, $min\_support$)
16:        end for
17:    else                                   ▷ $direction ==$ "$suffix$"
18:        for all $x_i \in X$ do
19:            PREFIXSUFFIXSPAN($S^{\epsilon,x_i}$, $\alpha$, $x_i.\alpha\prime$, $o$, $nextPos$, $min\_support$)
20:        end for
21:    end if
22: end function
```

Algorithm 2 Computation of the current prefix and suffix values of a prefixSuffixSpan call of depth d with $S(x_1, x_2, ..., x_d)$ as the database (of depth d).

```
1: function PREFIXSUFFIXARGUMENTS(Prefix $\alpha$, Suffix $\alpha\prime$, Direction $o_i$, ListOfLengthOnePatterns $X$)
2:     $\alpha \leftarrow \epsilon$
3:     $\alpha\prime \leftarrow \epsilon$
4:     for all $i \in$ {integer k such that $1 \le k \le d$} do   ▷ This set is empty if $d = 0$.
5:         if $o_{i-1} ==$ "$prefix$" then
6:             $\alpha \leftarrow \alpha.x_i$
7:         else                                ▷ $o_{i-1} ==$ "$suffix$"
8:             $\alpha\prime \leftarrow x_i.\alpha\prime$
9:         end if
10:    end for
11: end function
```

prefix and suffix values. Assume that the result is true up to depth d. Stemming from this, we prove in the following that the result is also true for depth $(d+1)$. To this end, we consider a dataset of depth $(d+1)$ denotes $S(x_1, x_2, ..., x_d, x_{d+1})$. It is constructed either by statement 15 or 19 of algorithm 1 during the execution of the *prefixSuffixSpan* call having $S(x_1, x_2, ..., x_d)$ as the dataset argument. Denote α and $\alpha\prime$ the values of prefix and suffix arguments related to that *prefixSuffixSpan* call. From statement 15 of algorithm 1, the current prefix value for depth $(d+1)$ is $\alpha.x_i$ if $o_d =$ "$prefix$" and α otherwise. Similarly, from statement 19 of algorithm 1, the current prefix value for depth $(d+1)$ is $x_i.\alpha\prime$ if $o_d =$ "$suffix$" and $\alpha\prime$ otherwise. Futhermore, the function call PREFIXSUFFIXARGUMENTS($\alpha, \alpha\prime, o_{d-1}, X$) of algorithm 2 provides the values of α and $\alpha\prime$ as concatenations of length-1 sequences belonging to $X = \{x_1, x_2, \ldots, x_d\}$ as we have assumed that the lemma is true for depth d. Therefore the values of prefix and suffix arguments for the *prefixSuffixSpan* call having $S(x_1, x_2, ..., x_d, x_{d+1})$ as the database argument are also provided by the function call PREFIXSUFFIXARGUMENTS(α, $\alpha\prime$, o_d, $X \cup \{x_{d+1}\}$) of algorithm 2 as concatenations of length-1 sequences belonging to $X \cup \{x_{d+1}\}$. Hence the lemma. □

Corollary 3 (The sizes of the prefix and suffix of a *prefixSuffixSpan* call). Given a depth d call PREFIXSUFFIXSPAN($S(x_1, x_2, ..., x_d)$, α, $\alpha\prime$, o, d, $min\_support$), with $o = o_0$-o_1-o_2 ... o_{d+1} ... o_{n-1}-$growth$, the lengths of prefix α and suffix $\alpha\prime$ are respectively $|\alpha| = |\{o_i|o_i = "prefix" \wedge i \leq d-1\}|$ and $|\alpha\prime| = |\{o_i|o_i = "suffix" \wedge i \leq d-1\}|$.

Proof. Consider a depth d call PREFIXSUFFIXSPAN($S(x_1,\ x_2,\ ...,\ x_d)$, α, $\alpha\prime$, o, d, $min\_support$), with $o = o_0$-o_1-o_2 ... o_{d+1} ... o_{n-1}-$growth$. According to lemma 6, the values of prefix α and suffix $\alpha\prime$ are provided by the function call PREFIXSUFFIXARGUMENTS($\alpha, \alpha\prime, o_{d-1}, X$) of algorithm 2, where $X = \{x_1,\ x_2,\ ...,\ x_d\}$. From statements 6 (resp. 8) of algorithm 2, the length of α (resp. $\alpha\prime$) is equal to the number of pattern-growth directions belonging to $\{o_1,\ o_2,\ ...,\ o_{d-1}\}$ which are equal to "prefix" (resp. "suffix"). Hence the corollary. □

Lemma 7 (The support of z in a depth-d set is that of its S'counterpart). Given a depth d database $S(x_1,\ x_2,\ ...,\ x_d)$ obtained from the initial database S and a linear ordering $o = o_0$-o_1-o_2 ... o_{d+1} ... o_{n-1}-$growth$, we have $support(S(x_1,\ x_2,\ ...,\ x_d), z) = support(S, \alpha.z.\alpha\prime)$ for any sequence z, where prefix α and suffix $\alpha\prime$ are provided by the function call PREFIXSUFFIXARGUMENTS($\alpha, \alpha\prime, o_{d-1}, X$) and $X = \{x_1,\ x_2,\ ...,\ x_d\}$.

Proof. We prove the result by induction on the depth value. Let z denotes a sequence. If depth $d = 0$, we have $S(x_1,\ x_2,\ ...,\ x_d) = S$, $\alpha = \epsilon$ and $\alpha\prime = \epsilon$. It comes that, $support(S(x_1,\ x_2,\ ...,\ x_d), z) = support(S, \alpha.z.\alpha\prime)$ and the result is true for this case. Now, assume that the result is true up to depth d. Stemming from this, we prove in the following that the result is also true for depth $(d+1)$. To this end, we consider a dataset of depth $(d+1)$ denotes $D\prime = S(x_1,\ x_2,\ ...,\ x_d,\ x_{d+1})$. It is constructed either by statement 15 or 19 of algorithm 1 during the execution of the *prefixSuffixSpan* call having $D = S(x_1,\ x_2,\ ...,\ x_d)$ as the dataset argument. Denote α and $\alpha\prime$ the prefix and suffix values related to D following algorithm 2. Similarly denote α_1 and $\alpha\prime_1$ the prefix and suffix values related to $D\prime$ following algorithm 2. We have $\alpha_1 = \alpha.x_{d+1}$ if $o_d = "prefix"$ and $\alpha_1 = \alpha$ otherwise. Similarly, $\alpha\prime_1 = x_{d+1}.\alpha\prime$ if $o_d = "suffix"$ and $\alpha\prime_1 = \alpha\prime$ otherwise. From statements 15 and 19, $D\prime = D^{x_{d+1},\epsilon}$ if $o_d = "prefix"$ and $D\prime = D^{\epsilon,x_{d+1}}$ if $o_d = "suffix"$.

Assume that $o_d = "prefix"$. This implies that $support(D\prime, z) = support(D, x_{d+1}.z)$ according to lemma 1. Futhermore, $support(D, x_{d+1}.z) = support(S, \alpha.x_{d+1}.z.\alpha\prime)$ from the induction assumption. It comes $support(D\prime, z) = support(S, \alpha.x_{d+1}.z.\alpha\prime) = support(S, \alpha_1.z.\alpha\prime_1)$. Thus the lemma holds in this case.

Similarly, assume that $o_d = "suffix"$. This implies that $support(D\prime, z) = support(D, z.x_{d+1})$ according to lemma 1. Futhermore, $support(D, z.x_{d+1}) = support(S, \alpha.z.x_{d+1}.\alpha\prime)$ from the induction assumption. It comes $support(D\prime, z) = support(S, \alpha.z.x_{d+1}.\alpha\prime) = support(S, \alpha_1.z.\alpha\prime_1)$. Thus the lemma also holds in this second case. Therefore the lemma holds. □

Corollary 4 (*prefixSuffixSpan* tells the truth). If *prefixSuffixSpan* says that a sequence s is a pattern, then s is really a sequential pattern of the initial sequence database S.

Proof. Consider a depth d call PREFIXSUFFIXSPAN($S(y_1,\ y_2,\ ...,\ y_d)$, α, $\alpha\prime$, o, d, $min\_support$), with $o = o_0$-o_1-o_2 ... o_{d+1} ... o_{n-1}-$growth$. According to lemma 6, the values of prefix α and suffix $\alpha\prime$ are provided by algorithm 2. Denote $D = S(y_1,\ y_2,\ ...,\ y_d)$. Statement 10 of algorithm 1 saves $\alpha.x_i.\alpha\prime$ as a pattern, where x_i is a length-1 sequential pattern of D, i.e. $support(D, x_i) \geq min\_support$. From lemma 7, $support(D, x_i) = support(S, \alpha.x_i.\alpha\prime)$. This implies that $support(S, \alpha.x_i.\alpha\prime) \geq min\_support$. Therefore $\alpha.x_i.\alpha\prime$ is a sequential pattern in S. Hence the Corollary. □

Lemma 8 (Any pattern of size d induces a depth-d database). Given an ordering $o = o_0$-o_1-o_2 ... o_{d+1} ... o_{n-1}-$growth$ and a sequential pattern $x = x_1.x_2$ ... x_d of the initial database S which is decomposed in terms of a product of length-1 sequential patterns, there exists a depth d database $D = S(x_{i1},\ x_{i2},\ ...,\ x_{id})$, where the x_{ij}'s, $1 \leq j \leq d$, are distinct length-1 sequential patterns belonging to $X = \{x_1,\ x_2,\ ...\ x_d\}$ and the values of prefix α and suffix $\alpha\prime$ related to the *prefixSuffixSpan* call having D as the database are $\alpha = x_1.x_2 ... x_p$ and $\alpha\prime = x_{p+1}.x_{p+2} ... x_d$ respectively, with $p = |\{o_i|o_i = "prefix" \wedge i \leq d-1\}|$.

Proof. Consider an ordering $o = o_0$-o_1-o_2 ... o_{d+1} ... o_{n-1}-$growth$ and a sequential pattern x of the initial database S. We prove the result by induction on the length of x denoted d. Assume that $|x| = 1$, i.e. $d = 1$. Consider the execution of the initial function call PREFIXSUFFIXSPAN(S, ϵ, ϵ, o, 0, $min\_support$). During that execution, statement 6 of algorithm 1 generates x as a length-1 sequential pattern as it finds all the length-1 sequential patterns of the initial database S. Futhermore, from statement 15 of algorithm 1, we have $S(x) = S^{x,\epsilon}$, $\alpha = x$ and $\alpha\prime = \epsilon$ if $o_0 = "prefix"$. Similarly, from statement 19, we have $S(x) = S^{\epsilon,x}$, $\alpha = \epsilon$ and $\alpha\prime = x$ if $o_0 = "suffix"$. Thus, the lemma is true if $|x| = 1$.

Now, assume that the result is true up to rank d, i.e. for any sequential pattern belonging to S whose length is lower or equal to d. Consider a length-(d+1) sequential pattern $x\prime$ of the initial database S. The following equation $x\prime = x\prime_1.x\prime_2 ... x\prime_p.x\prime_{p+1}.x\prime_{p+2} ... x\prime_{d+1}$, with $p = |\{o_i|o_i = "prefix" \wedge i \leq d-1\}|$, decomposes $x\prime$ in terms of a product of length-1 sequential patterns which can be divided into three parts. The left part is $x\prime_1.x\prime_2 ... x\prime_p$, the middle part is $x\prime_{p+1}$ and the right part is $x\prime_{p+2} ... x\prime_{d+1}$. Consider the subsequence of $x\prime$, denoted x, obtained by applying the dot operator with the left and right parts of $x\prime$ as operands. We have $x = x\prime_1.x\prime_2 ... x\prime_p.x\prime_{p+2} ... x\prime_{d+1}$.

Sequence x is a sequential pattern according to an Apriori principle which states that a subsequence of a sequential pattern is also a sequential pattern. From the induction assumption, the lemma is true for subsequence x of $x\prime$ as $|x| = d$. Thus, there exists a *prefixSuffixSpan* call having $D = S(x_{i1}, x_{i2}, ..., x_{id})$ as the database argument, $\alpha = x\prime_1.x\prime_2 \ ... \ x\prime_p$ as the prefix value and $\alpha\prime = x\prime_{p+2}.x\prime_{p+3} \ ... \ x\prime_{d+1}$ as the suffix value, where $p = |\{o_i | o_i = "prefix" \wedge 0 \leq i \leq d-1\}|$ and x_{ij}'s, $1 \leq j \leq d$, are distinct length-1 sequential patterns belonging to $X = \{x\prime_1, x\prime_2, ... x\prime_p, x\prime_{p+2}, ... x\prime_{d+1}\}$. Note that prefix α and suffix $\alpha\prime$ correspond respectively to the left and right parts of sequence $x\prime$, and it comes that $x\prime = \alpha.x\prime_{p+1}.\alpha\prime$. Note also that the function call corresponds to PREFIXSUFFIXSPAN($S(x_{i1}, x_{i2}, ..., x_{id}), \alpha, \alpha\prime, o, d, min\_support$).

Futhermore, according to lemma 7, we have $support(S(x_{i1}, x_{i2}, ..., x_{id}), x\prime_{p+1}) = support(S, \alpha.x\prime_{p+1}.\alpha\prime)$. It comes that $support(S(x_{i1}, x_{i2}, ..., x_{id}), x\prime_{p+1}) = support(S, x\prime)$ as $x\prime = \alpha.x\prime_{p+1}.\alpha\prime$, and this implies that $x\prime_{p+1}$ is a length-1 sequential pattern of $D = S(x_{i1}, x_{i2}, ..., x_{id})$ as $x\prime$ is a sequential pattern of S. This implies that during the execution of the function call PREFIXSUFFIXSPAN($S(x_{i1}, x_{i2}, ..., x_{id}), \alpha, \alpha\prime, o, d, min\_support$), statement 6 of algorithm 1 generates $x\prime_{p+1}$ as a length-1 sequential pattern as it finds all the length-1 sequential patterns of D. Thus, during the execution, statement 15 of algorithm 1 makes a *prefixSuffixSpan* recursive call with $D\prime = S(x_{i1}, x_{i2}, ..., x_{id}, x_{p+1}) = D^{x\prime_{p+1}, \epsilon}$ as the database argument, $\alpha_1 = \alpha.x\prime_{p+1} = x\prime_1.x\prime_2 \ ... \ x\prime_{p+1}$ as the prefix value and $\alpha\prime_1 = \alpha\prime = x\prime_{p+2}.x\prime_{p+3} \ ... \ x\prime_{d+1}$ as the suffix value if $o_d = "prefix"$. In this first case, we have $|\{o_i | o_i = "prefix" \wedge 0 \leq i \leq d\}| = (p+1)$, and the lemma holds. Similarly, statement 19 of algorithm 1 makes a *prefixSuffixSpan* recursive call with $D\prime = S(x_{i1}, x_{i2}, ..., x_{id}, x_{p+1}) = D^{\epsilon, x\prime_{p+1}}$ as the database argument, $\alpha_1 = \alpha = x\prime_1.x\prime_2 ... x\prime_p$ as the prefix value and $\alpha\prime_1 = x\prime_{p+1}.\alpha\prime = x\prime_{p+1}.x\prime_{p+2} \ ... \ x\prime_{d+1}$ as the suffix value if $o_d = "suffix"$. In this second case, we have $|\{o_i | o_i = "prefix" \wedge 0 \leq i \leq d\}| = p$, and the lemma holds. Therefore we have the lemma. □

Corollary 5 (*prefixSuffixSpan* discovers all the patterns). Algorithm *prefixSuffixSpan* declares all sequence which is a sequential pattern as so.

Proof. Consider a length-d sequential pattern $x = x_1.x_2 ... x_d$ of the initial database S and a linear ordering $o = o_0\text{-}o_1\text{-}o_2 \ ... \ o_{d+1} \ ... \ o_{n-1}\text{-}growth$. From lemma 8, there exists a depth d database $D = S(x_{i1}, x_{i2}, ..., x_{id})$, where the x_{ij}'s, $1 \leq j \leq d$, are distinct length-1 sequential patterns belonging to $X = \{x_1, x_2, ... x_d\}$ and the values of prefix α and suffix $\alpha\prime$ related to the *prefixSuffixSpan* call having D as the database are $\alpha = x_1.x_2 \ ... \ x_p$ and $\alpha\prime = x_{p+1}.x_{p+2} \ ... \ x_d$ respectively, with $p = |\{o_i | o_i = "prefix" \wedge i \leq d-1\}|$. This function call is PREFIXSUFFIXSPAN($S(x_{i1}, x_{i2}, ..., x_{id}), \alpha, \alpha\prime, o, d, min\_support$). It is launched either by statement 15 of algorithm 1 if $o_{d-1} = "prefix"$ or by statement 19 otherwise, i.e. if $o_{d-1} = "suffix"$, during the execution of the previous function call, i.e. PREFIXSUFFIXSPAN($S(x_{i1}, x_{i2}, ..., x_{i(d-1)}), \alpha_2, \alpha\prime_2, o, d-1, min\_support$). The prefix value α and the suffix value $\alpha\prime$ are calculated during the execution of that previous function call as follows. From statement 15 of algorithm 1, we have $\alpha = \alpha_2.x_p$ and $\alpha\prime = \alpha\prime_2$ if $o_{d-1} = "prefix"$. Similarly, from statement 19 of algorithm 1, we have $\alpha = \alpha_2$ and $\alpha\prime = x_{p+1}.\alpha\prime_2$ if $o_{d-1} = "suffix"$. Futhermore, during the execution of that previous function call, statement 6 of algorithm 1 saves $\alpha_2.x_p.\alpha\prime_2$ as a sequential pattern. Therefore $x = \alpha.\alpha\prime = \alpha_2.x_p.\alpha\prime_2$ is save as a sequential pattern. Hence the corollary. □

We have the following theorem.

Theorem 1 (Veracity of *prefixSuffixSpan*). A sequence is a pattern is and only if *prefixSuffixSpan* says so.

Proof. It is straightforward from corollaries 4 and 5. □

3.4. Experimental results

The data set used here is collected from the webpage of SPMF software [7]. This webpage (http://www.philippe-fournier-viger.com/spmf/index.php) provides large data sets in SPMF format that are often used in the data mining litterature for evaluating and comparing algorithm performance.

Experiments were performed on the real-life. The first data set is *LEVIATHAN*. It contains 5834 sequences and 9025 distinct items. The second data set is *Kosarak*. It is a very large data set containing 990000 sequences of click-stream data from an hungarian news portal. The third data set is *BIBLE*. It is a conversion of the Bible into a sequence database (each word is an item). It contains 36 369 sequences and 13905 distinct items. The fourth data set is BMSWebView2 (Gazelle). It is called here *BMS2*. It contains 59601 sequences of clickstream data from e-commerce and 3340 distinct items.

These dataset in its original format can be found at http://fimi.ua.ac.be/data/. A SPMF format is provided at http://www.philippe-fournier-viger.com/spmf/index.php.

All experiments are done on a 4-cores of 2.16GHz Intel(R) Pentium(R) CPU N3530 with 4 gigabytes main memory, running Ubuntu 14.04 LTS. The algorithms are implemented in Java and grounded on SPMF software [7].

The experiments consisted of running the pattern-growth algorithms related to the left-to-right and

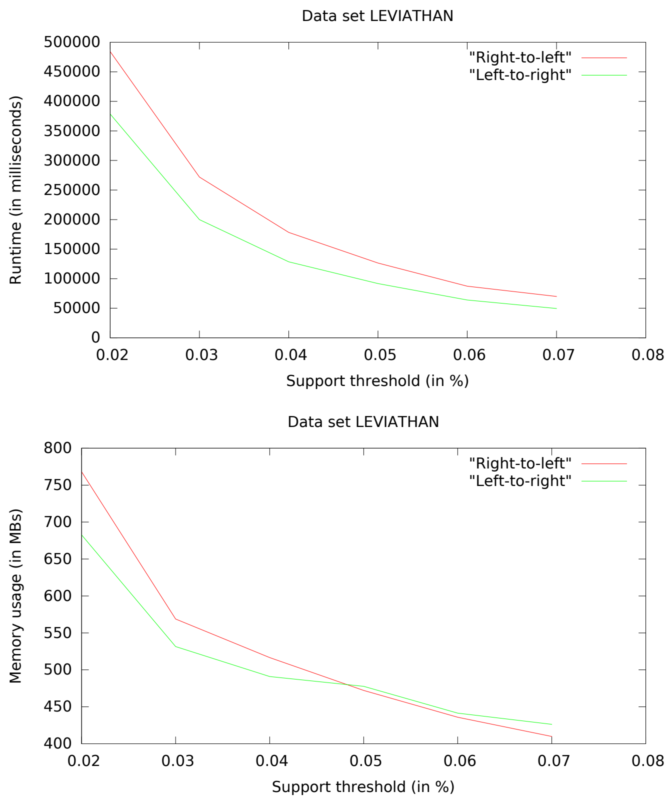

Figure 1. Performances of left-to-right and right-to-left pattern-growth orderings on the real-life data set LEVIATHAN. The left-to-right pattern-growth ordering is 1.27 – 1.4 times faster, and requires less memory if the support threshold is less than 0.05 and a little more memory otherwise.

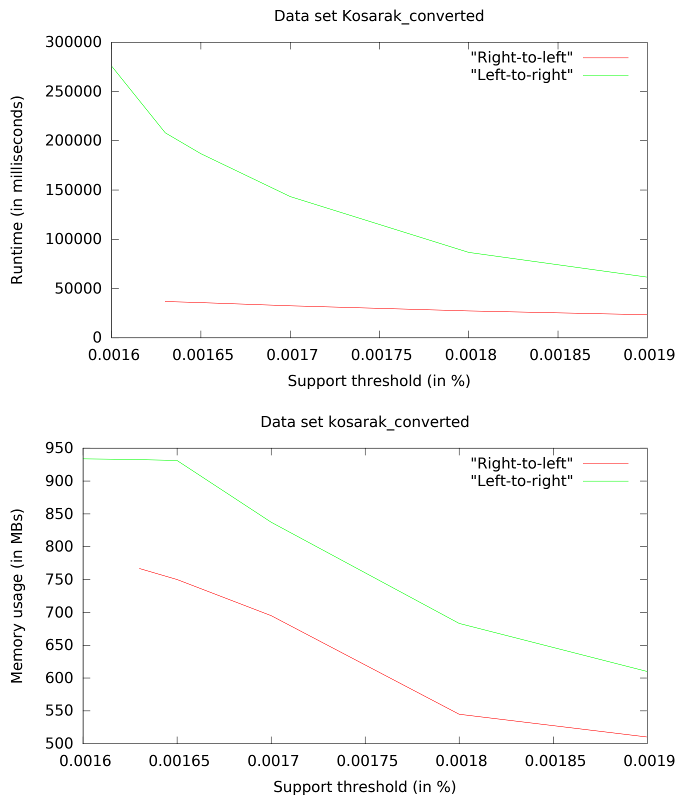

Figure 2. Performances of left-to-right and right-to-left pattern-growth orderings on the real-life data set kosarak_converted. The right-to-left pattern-growth ordering is 2.6 – 5.6 times faster and requires almost 1.2 times less memory than the other direction.

the right-to-left orderings. on each data set while decreasing the support threshold until algorithms became too long to execute or ran out of memory. The performances are presented in figures 1, 2, 3 and 4. These figures show that the order in which patterns grow has a significant influence on the performances.

4. Conclusion

Sequential pattern mining is an important data mining problem with broad applications. However, it is also a challenging problem since the mining may have to generate or examine a combinatorially explosive number of intermediate subsequences. It has been a focused theme in data mining research for over a decade. Abundant literature has been dedicated to this research and tremendous progress has been made, ranging from efficient and scalable algorithms for frequent itemset mining to numerous research frontiers, such as sequential pattern mining, structured pattern mining, correlation mining, associative classification, and frequent pattern-based clustering, as well as their broad applications.

In this article, an overview is provided on the current status of pattern growth-based sequential pattern mining algorithms. The important key concepts of the pattern-growth approach are revisited, formally defined and extended. A new class of pattern-growth algorithms inspired from a new class of pattern-growth orderings, called linear ordering, is introduced. Issues of this new class of pattern-growth algorithms related to search space pruning and partitioning are investigated. Stemming from this theoretical study, a new algorithm called *prefixSuffixSpan* is designed. Its correctness is proven and related experimental results are presented.

References

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. pages 207–216, 1993.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499, 1994.

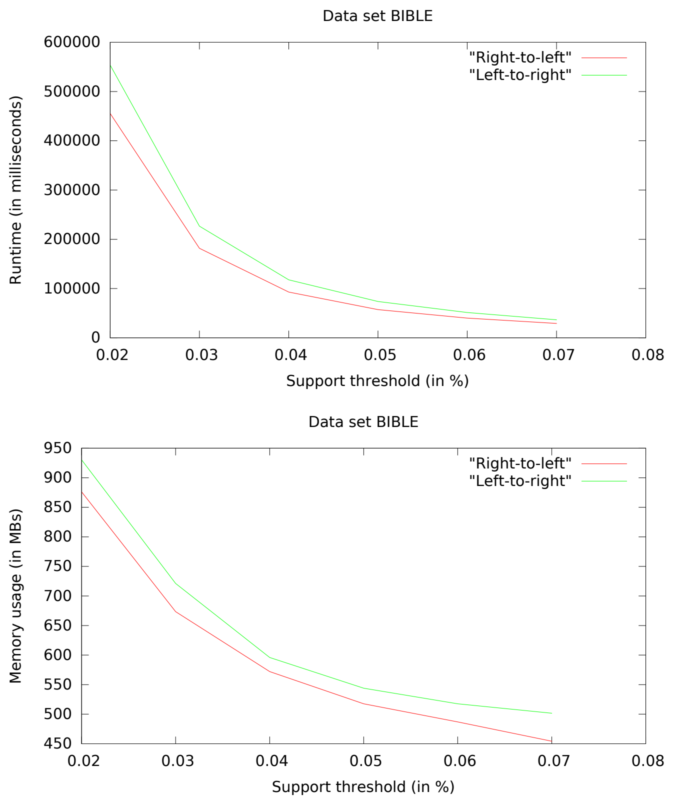

Figure 3. Performances of left-to-right and right-to-left pattern-growth orderings on the real-life data set BIBLE. The right-to-left pattern-growth ordering is $1.21 - 1.25$ times faster and requires almost $1.04 - 1.10$ times less memory than the other ordering.

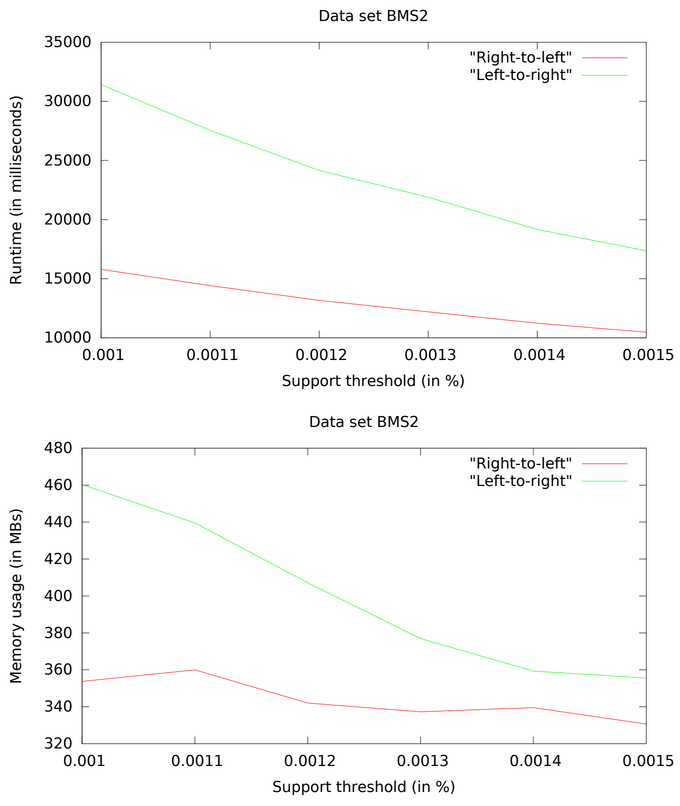

Figure 4. Performances of left-to-right and right-to-left pattern-growth orderings on the real-life data set BMS2. The right-to-left pattern-growth ordering is $1.5 - 2$ times faster and requires almost $1.07 - 1.3$ times less memory than the other ordering.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan*, pages 3–14, 1995.

[4] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. 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*, pages 429–435, 2002.

[5] T. Dam, K. Li, P. Fournier-Viger, and Q. Duong. An efficient algorithm for mining top-rank-k frequent patterns. *Appl. Intell.*, 45(1):96–111, 2016.

[6] M. El-Sayed, C. Ruiz, and E. A. Rundensteiner. Fs-miner: efficient and incremental mining of frequent sequence patterns in web logs. In *Sixth ACM CIKM International Workshop on Web Information and Data Management (WIDM 2004), Washington, DC, USA, November 12-13, 2004*, pages 128–135, 2004.

[7] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C. Wu, and V. S. Tseng. SPMF: a java open-source pattern mining library. *Journal of Machine Learning Research*, 15(1):3389–3393, 2014.

[8] M. N. Garofalakis, R. Rastogi, and K. Shim. SPIRIT: sequential pattern mining with regular expression constraints. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 223–234, 1999.

[9] K. Gouda, M. Hassaan, and M. J. Zaki. Prism: A primal-encoding approach for frequent sequence mining. In *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), October 28-31, 2007, Omaha, Nebraska, USA*, pages 487–492, 2007.

[10] K. Gouda, M. Hassaan, and M. J. Zaki. Prism: An effective approach for frequent sequence mining via prime-block encoding. *J. Comput. Syst. Sci.*, 76(1):88–102, 2010.

[11] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. Hsu. Freespan: frequent pattern-projected sequential pattern mining. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA, USA, August 20-23, 2000*, pages 355–359, 2000.

[12] C. Hsieh, D. Yang, and J. Wu. An efficient sequential pattern mining algorithm based on the 2-sequence matrix. In *Workshops Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 583–591, 2008.

[13] J. Huang, M. Peng, H. Wang, J. Cao, G. Wang, and X. Zhang. A probabilistic method for emerging topic tracking in microblog stream. *Accepted by World Wide Web on Feb, 2016.*

[14] E. Kabir, A. Mahmood, H. Wang, and A. Mustafa. Microaggregation sorting framework for k-anonymity statistical disclosure control in cloud computing. *IEEE Transactions on Cloud Computing*, PP(99):1–1, 2015.

[15] E. B. Kenmogne. The impact of the pattern-growth ordering on the performances of pattern growth-based sequential pattern mining algorithms. *Computer and Information Science. To appear.*

[16] N. R. Mabroukeh and C. I. Ezeife. A taxonomy of sequential pattern mining algorithms. *ACM Comput. Surv.*, 43(1):3, 2010.

[17] F. Masseglia, F. Cathala, and P. Poncelet. The PSP approach for mining sequential patterns. In *Principles of Data Mining and Knowledge Discovery, Second European Symposium, PKDD '98, Nantes, France, September 23-26, 1998, Proceedings*, pages 176–184, 1998.

[18] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In *Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany*, pages 215–224, 2001.

[19] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Trans. Knowl. Data Eng.*, 16(11):1424–1440, 2004.

[20] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu. Mining access patterns efficiently from web logs. In *Knowledge Discovery and Data Mining, Current Issues and New Applications, 4th Pacific-Asia Conference, PADKK 2000, Kyoto, Japan, April 18-20, 2000, Proceedings*, pages 396–407, 2000.

[21] L. Savary and K. Zeitouni. Indexed bit map (IBM) for mining frequent sequences. In *Knowledge Discovery in Databases: PKDD 2005, 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, Porto, Portugal, October 3-7, 2005, Proceedings*, pages 659–666, 2005.

[22] M. Seno and G. Karypis. Slpminer: An algorithm for finding frequent sequential patterns using length-decreasing support constraint. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*, pages 418–425, 2002.

[23] Z. Yang and M. Kitsuregawa. LAPIN-SPAM: an improved algorithm for mining sequential pattern. In *Proceedings of the 21st International Conference on Data Engineering Workshops, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, page 1222, 2005.

[24] Z. Yang, Y. Wang, and M. Kitsuregawa. LAPIN: effective sequential pattern mining algorithms by last position induction for dense databases. In *Advances in Databases: Concepts, Systems and Applications, 12th International Conference on Database Systems for Advanced Applications, DASFAA 2007, Bangkok, Thailand, April 9-12, 2007, Proceedings*, pages 1020–1023, 2007.

[25] M. J. Zaki. 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*, pages 422–429, 2000.

[26] M. J. Zaki. SPADE: an efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.