Tomas Bata University in Zlín
Library

An efficient method for mining sequential patterns with indices

Citation

HUYNH, Minh Huy, Loan T.T. NGUYEN, Ngoc Nam PHAM, Zuzana KOMÍNKOVÁ OPLATKOVÁ, Unil YUN, and Bay VO. An efficient method for mining sequential patterns with indices. *Knowledge-Based Systems* [online]. vol. 239, Elsevier, 2022, [cit. 2023-07-03]. ISSN 0950-7051. Available at https://www.sciencedirect.com/science/article/pii/S0950705121010832

DOI

https://doi.org/10.1016/j.knosys.2021.107946

Permanent link

https://publikace.k.utb.cz/handle/10563/1010790

This document is the Accepted Manuscipt version of the article that can be shared via institutional repository.

TBU Publications
Repository of TBU Publications
publikace.k.utb.cz

An efficient method for mining sequential patterns with indices

Huy Minh Huynh[a], Loan T.T. Nguyen[b,c], Nam Ngoc Pham[a], Zuzana Komínková Oplatková[a], Unil Yun[d], Bay Vo[e, *]

[a]*Faculty of Applied Informatics, Tomas Bata University in Zlín, Nám. T.G. Masaryka 5555, Zlín, Czech Republic*

[b]*School of Computer Science and Engineering, International University, Ho Chi Minh City, Viet Nam*

cVietnam National University, Ho Chi Minh City, Viet Nam

[d]*Department of Computer Engineering, Sejong University, Seoul, Republic of Korea*

[e]*Faculty of Information Technology, HUTECH University, Ho Chi Minh City, Viet Nam*

[*]*Corresponding author: E-mail addresses: huynh@utb.cz (H.M. Huynh), nttloan@hcmiu.edu.vn*

Abstract

In recent years, mining informative data and discovering hidden information have become increasingly in demand. One of the popular means to achieve this is sequential pattern mining, which is to find informative patterns stored in databases. Its applications cover different areas and many methods have been proposed. Recently, pseudo-IDLists were proposed to improve both runtime and memory usage in the mining process. However, the idea cannot be directly used for sequential pattern mining as it only works on clickstream patterns, a more distinct type of sequential pattern. We propose adaptations and changes to the original idea to introduce SUI (Sequential pattern mining Using Indices). Comparing SUI with two other state-of-the-art algorithms on six test databases, we show that SUI has effective and efficient performance and memory usage.

Keywords: Pseudo-IDList, Data-IDList, vertical format, sequential pattern mining

1. Introduction

In this current era of the Internet, data generated from online businesses such as commercial websites have grown exponentially. Meaningful and useful data are usually discovered from raw data via some methods of automatic extraction. One form of informative data can be expressed via frequent patterns, and one of the effective methods to discover these is via sequential pattern mining (SPM), which was first presented by Agrawal and Srikant [**1**]. The main purpose is to find common sequential patterns in databases and their ratios. For example, in a supermarket transaction database, 40% of customers first bought ''bread and butter'', then ''jam'', and then ''cheese''.

Many approaches have been proposed for SPM [**2-4**] as well as other branches of the problem such as SPM with constraints [**5,6**], high utility SPM [**7-9**], closed and generator SPM [**10,11**], and mining patterns in multiple sequences [**12**]. Their applications cover various domains, from extracting clickstream-type patterns [**13,14**] such as DNA sequences, event sequences, clickstream sequences on online stores, or the purchase transactions of customers [**15**], to learning resource recommendations [**16**].

Most proposed algorithms can be separated into two common groups, vertical or horizontal, depending on which data structure they use. SPADE [**17**], SPAM [**18**], and the recently improved version CM-SPADE [**19**] are some of the most popular algorithms in the vertical group. FreeSpan [**20**], and PrefixSpan [**21**] are two popular algorithms in this horizontal group. According to [**17-19**], the performance of the vertical group is overall good and generally has some advantages over the other groups (e.g. counting support without the need of re-scanning databases). However, data duplication can occur often in this group. The authors in [**22**] proposed the CUP algorithm to eliminate some of the duplicate data in the mining process. However, the original idea, which is based on pseudo-IDLists, only works for clickstream patterns, which are a special type of sequential pattern. In sequential patterns, two or more events (which are called items) can co-occur in several positions. Nonetheless, clickstreams do not allow such a thing to happen, as events only occur one after another. Thus, it is not possible to use CUP directly for sequential pattern mining. Section 4.4 gives more information about this matter.

Briefly, the main contributions of this paper are as follows:

- To exploit the idea of pseudo-IDList for SPM, changes and adaptations are proposed to the original CUP algorithm [**22**] to create the SUI algorithm.
- To prove that SUI performs well regarding runtime and memory footprint, evaluation is conducted based on six test databases including three real-life and three synthetic databases.

Table 1 An example of a sequential transaction database.

TSID	Transaction sequences
100	{1, 3, 4}, {1, 2}, {3, 4}, {1, 2, 3}
200	{1, 2}, {1, 3}
300	{1}, {1, 2, 3}
400	{3}, {2, 4}
500	{2, 3}, {1, 2, 3}, {1, 3}

This paper is an extension of work that was reported in [23], there are differences between this version and the one which is published in [**23**], as follows:

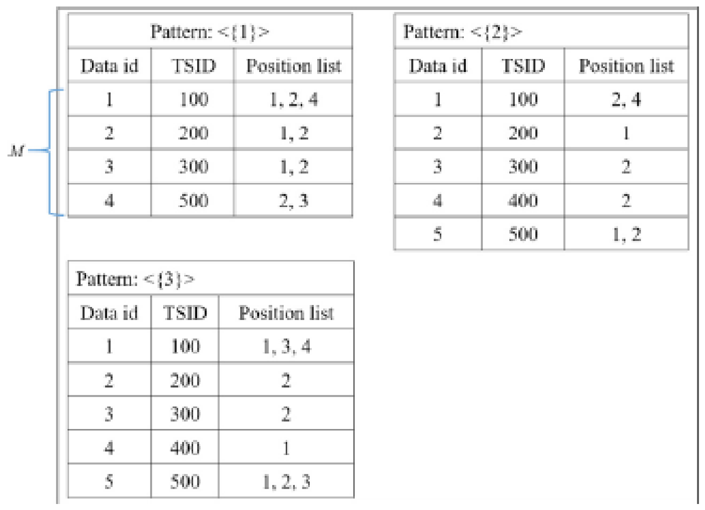
- More detailed information (such as theorems, concepts, and proofs). Because of the page limit of a conference paper, we limited the amount of information that can be presented in [**23**]. In this version, we add in more details such as a related work section to help readers have an overview of other work, a subsection about the downward property to help readers understand the basic pruning technique, a more detailed section on the candidate generation process, the general process of SUI, and estimation of SUI's complexity compared with other algorithms in the same group. Additionally, we also provide theorems and lemma to prove that CUP does not fully work for SPM, so our adaptations and changes of the original idea are needed for SUI to work properly.
- More experimental databases alongside the new scalability experiment section. In the conference version, we only evaluate SUI on three synthetic databases, but in this journal version we evaluate SUI on six databases. Additionally, we present an additional scalability experiment to demonstrate the efficiency, effectiveness, and scalability of SUI to a greater extent.

The remainder of this paper is organized as follows. Some related work and algorithms are presented in **Section 2**. Definitions and basic concepts relating to SPM are introduced in **Section 3**. Our proposed changes, adaptations, data structures, and algorithms are also presented in **Section 3**. The evaluation and experimental results on runtime, memory footprint, and scalability are presented in **Section 4**. Lastly, our conclusions and future work are presented in **Section 5**.

2. Related work

The original pattern mining problem was frequent itemset mining (FIM), as proposed by Agrawal et al. in **[24]**. The problem is to find out which sets of items are usually bought together by customers through the recorded transactions of retail stores. Those sets of items can then be analyzed to produce valuable information such as association rules that can be interpreted to aid decision-making. FIM has many applications as well as various studies, such as [**25-28**]. However, FIM does not contain the sequential order in which the transactions are made by customers, and the order of the events or transactions is important to certain fields, such as intrusion detection or event log mining. Thus, in 1995 Agrawal and Srikant [**1**] integrated the time order with the transactions to propose sequential pattern mining (SPM), which then became an important problem in the field of pattern mining.

Some of the most well-known classical algorithms for SPM are AprioriAll [**1**], FreeSpan [**20**], PrefixSpan [**21**], SPADE [**17**], and SPAM [**18**]. They can be categorized into two groups that use two different data formats. The first one is the horizontal database format (**Table 1**), which includes AprioriAll, FreeSpan, and PrefixSpan.



Pattern: <{1}>		
Data id	TSID	Position list
1	100	1, 2, 4
2	200	1, 2
3	300	1, 2
4	500	2, 3

Pattern: <{2}>		
Data id	TSID	Position list
1	100	2, 4
2	200	1
3	300	2
4	400	2
5	500	1, 2

Pattern: <{3}>		
Data id	TSID	Position list
1	100	1, 3, 4
2	200	2
3	300	2
4	400	1
5	500	1, 2, 3

Fig. 1. Data-IDLists of frequent 1-patterns (i.e. the vertical format database).

The other two, SPADE and SPAM, are in the vertical database format group (**Fig. 1**). AprioriAll [**1**] is based on the apriori property and is considered the first algorithm for mining sequential patterns. The property states that any frequent pattern cannot contain infrequent sub frequent patterns. The algorithm uses a generate-and-test approach that has multiple passes. In each pass, AprioriAll generates new candidates by appending previous seeds (or previously found frequent patterns) and scans the database for the support counts of the candidates to decide if they are frequent or not. FreeSpan and PrefixSpan employ a different approach from AprioriAll, one called pattern growth. The advantage of this approach is that it does not need to generate and test candidates that do not exist in databases. Both FreeSpan and PrefixSpan also use the concept of projected databases, which are fragments of the original databases that satisfy certain conditions. Each time a new projected database is created, the projected database gets smaller, the candidate patterns grow longer and the support counts via scanning projected databases get faster. PrefixSpan is based on FreeSpan, but it uses prefix-projection. SPADE and SPAM transform the horizontal databases into vertical format, which consists of several data-IDLists. The IDLists record in which sequences and positions the patterns appear. While SPADE uses a list of integer values to represent those IDLists, SPAM uses arrays of bitmaps. The benefit of this approach is that it does not need to re-scan databases again to determine the support counts of patterns. IDLists also get smaller each time they are produced for longer patterns, and thus the process of generating those IDLists gets faster. The vertical group also has other advantages over the horizontal group and has better overall performance.

PrefixSpan, SPADE, and SPAM are base algorithms that have been developed into better ones, such as CM-SPAM, and CM-SPADE [**19**]. CM-SPADE and CM-SPAM [**19**] use the idea of cooccurrence information maps, which are pre-populated frequent patterns, to help prune infrequent candidates. CM-SPAM and CM-SPADE are considered faster than their base algorithms, SPADE, and SPAM, especially CM-SPADE, according to [**19**]. PrefixSpan, SPADE, and SPAM are also adapted into more specific algorithms for more diverse problems, such as GenPrefixSpan **[29]** or cSPADE [**30**].

Various applications have been proposed based on SPM. In the bioinformatics field, SPM is used for analyzing and understanding biological data such as motif discovery with regard to DNA sequences [**31,32**]. In [**32**], the authors proposed the DFSG algorithm based on the characteristics of protein sequences. In [**31**], the authors proposed SPAM_SNG, extended from SPAM. The motif discovery of DNA sequences is similar to n-gram mining, but with a dynamic min-max gap constraint. In the security domain, sequential frequent patterns have been used to aid the process of intrusion detection or the analysis of security attacker patterns [**33**]. In e-learning, sequential patterns are used to help predict student results or learn the navigating patterns of students [**34**]. For event log mining, sequential patterns are incorporated into systems to mine interesting patterns and analyze the relevant behaviors of workers' daily activities [**35**]. For healthcare systems, SPM is incorporated to help interactive visualization and on-demand analysis of clinical events of patients' medical conditions [**36**]. With the growth of smart devices (e.g., smartphones, sensors, and portable computers) in the Internet-of-Things (IoT) environment, the authors in [**37**] proposed HUSP-Spark to carry out high-utility sequential pattern mining for very large datasets, which are often encountered in such environments. A similar problem to IoT is the Internet of connected vehicles. As it includes uncertainty factors, the authors in [**38**] introduced a new type of pattern called high expected utility sequential patterns, incorporating both uncertainty and utility, to aid the quality evaluation of patterns. The authors then designed two data structures called PUL-Chain and EUL-Chain and integrated them to create the HEUSPM algorithm to mine high expected utility sequential patterns.

Based on SPM, diverse problems have been proposed to fit into different criteria. To provide more concise representative patterns, researchers proposed the closed and generator pattern mining methods [**11,39**]. Those types of patterns allow us to generate all the frequent patterns in the databases. Researchers also proposed more ways to limit the patterns found, such as inter-constraint [**6**], maximal constraint [**40**], or gap constraint approaches [**41**]. Another issue is high utility SPM or weighted SPM. Researchers reason that each element in patterns should have different degrees of importance so that the patterns found can be more useful and adjusted to user preferences. They thus incorporated the concept of importance under the name of either high utility [**42-45**] or weights [**46-48**]. Some works have also aimed to improve the quality of rules mined from sequential patterns [**49**]. The idea is to mine partial-order rules common among sequences instead of the strictly ordering rules.

A more distinct problem that deviates from SPM is clickstream pattern mining (CPM), in which each sequence contains itemsets with only one item. In fact, some of the above-mentioned works for substring mining, bioinformatics, healthcare systems, event log mining, and intrusion detection are CPM. One disadvantage of SPM horizontal algorithms is that they can contain duplicate information while executing the mining processes. For example, a pattern X contains a position list $<1, 4, 5, 6>$ and its super pattern Y contains a position list $<4, 5, 6>$, $<4, 5, 6>$ is a sublist of $<1, 4, 5, 6>$, which is duplicate information. CUP [22] was thus recently proposed for CPM, which exploits indices to reduce the amount of duplicate information during the mining process. This separates the data structures into two types, data-IDLists and pseudo-IDLists. While the former holds the real data, the latter contains indices that serve to retrieve the real data based on their referenced data-IDLists. The related experiments show that CUP outperforms the state-of-the-art algorithms PrefixSpan and CM-SPADE. In this paper, we extend the idea in CUP [**22**] to propose SUI, an algorithm for mining sequential patterns.

3. Problem definitions

In this section, we define the SPM problem and present basic concepts and definitions.

Let I = {i_1, i_2, ..., in} be a set of integer values. Each $i_1 \in I$ symbolizes an item (e.g. a pair of shoes). **A sequence** $s = <E_1, E_2,..., E_m>$ $(1 \leq i \leq m)$ is a list of itemsets in an order, where **an itemset** E_i e s is a subset of I and m is the **sequence size** of s. For example, let I = {1, 2, 3, 4, 5}, two possible 3-itemsets can be {2, 4, 5} and {1, 2, 5}.

When a customer purchases items in a store, **a user transaction sequence** is generated according to the customer's orders. A sequence is enclosed with "<" and ">" symbols while "{" and "}" enclose an itemset. For example, $<$ {2, 3}, {1, 2, 3}, {1, 3}> is a transaction sequence with three itemsets. $<$ {2, 3}, {1, 2, 3}, {1, 3}> means that a user purchased 2, and 3 together, then 1, 2, and 3, and lastly 1, and 3. A sequence containing l items is called an l-sequence, and l is its **sequence length**. For example, $<$ {2, 3}, {1, 2, 3}, {1, 3}> is a 7-sequence. Let $s_x = <X_1, X_2, ...,$ Xn> and $s_y = < Y_1, Y_2, ..., Y_m>$ be two sequences, and $n \leq m$. If there exist integers $1 < i_1 \leq i_2 < ... <$ in $< m$ such that $X_1 \subseteq Y_{i1}$, $X_2 \subseteq Y_{i2}$, ..., $X_n \subseteq Y_{in}$, s_y is considered a **supersequence** of s_x or s_x is considered a **subsequence** of s_y (i.e. s_x **appears** in s_y). Additionally, $s_x \sqsubset s_y$ denotes subsequence relation. For example, $<$ {3}, {1, 3} $>$ is considered a subsequence of $<$ {1, 3, 4}, {1, 2}, {3, 4}, {1, 2, 3}> (which is the first sequence in **Table 1**).

A list of user transaction sequences makes up a **sequence database** SDB = {S_1, S_2,..., S_g}. An integer value, which is called a transaction sequence id (tSiD), is assigned to each sequence. For example, there are five user transaction sequences in **Table 1** in which they have TSID from one to five. If a sequence is a subsequence of (or appears in) one or more user transaction sequences in a database, that sequence is considered a **pattern** in that database. Let supp(s) denote the number of user transaction sequences in which a pattern s appears; supp(s) is also called **the support of** s. Let δ be an integer value that denotes a **minimum support threshold** given by users, **a frequent (sequential) pattern** is a pattern having its support $\geq \delta$.

There are two types of patterns in SPM, **s-extension**, and ***i*-extension**. A pattern has an s-extension if its last itemset (i.e. the right-most itemset in that pattern) only has one item. Otherwise, if there is more than one item in the last itemset, the pattern has an i-extension. For example, let $X = <$ {2, 4}, {3}> and $Y = <$ {5}, {1, 3} $>$ be two patterns, then X has an s-extension and Y has an i-extension.

Let $X = < X_1, X_2 \cdots, X_n >$ and $Y = < Y_1, Y_2 \cdots, Y_m >$ $(n < m)$ be two sequences, X is a **prefix** of Y if $X_1 = Y_1, X_2 = Y_2, \ldots, X_n = Y_n$. For example, a sequence $< \{1\} >$ is a prefix of $< \{1\}, \{1, 5\}, \{3, 4, 5\} >$ and $< \{1\}, \{1, 5\}, \{3, 4\} >$ is also a prefix of $< \{1\}, \{1, 5\}, \{3, 4, 5\} >$.

Problem statement. Given a sequential database SDB and a minimum support threshold S, the goal of SPM is to find out all frequent patterns with supports $\geq S$ in SDB.

The downward closure property states that if a sequential pattern is frequent then all of its subsequences must be frequent. The property implies that: (1) If a frequent pattern Y has supp(Y), $\forall X \subset Y$: $supp(X) \geq supp(Y) \geq \delta$; (2) Let X be a pattern, such that $X \subset Y$. Assuming that S_X is a set of user transaction sequences containing X and SY is a set of user transaction sequences containing Y, then S_y is a subset of S_X . In other words, if we know that a user transaction sequence contains Y, we can infer that the transaction sequence must also contain X. However, if we are sure that a transaction sequence contains X, we cannot be sure that the transaction sequence contains Y. For example, let $X = < \{1\}, \{3\} >$, Y $= < \{1,3\}, \{3\} >$, then we have $S_X = \{100, 200, 300, 500\}$, $S_Y = \{100, 500\}$, and $S_Y \subset S_X$. This property is a popular strategy for filtering candidates in a lot of existing algorithms.

Example 1. Given $\delta = 2$ and an SDB in **Table 1**, the first transaction sequence (with TSID = 500) is $< \{2, 3\}, \{1, 2, 3\}, \{1, 3\} >$. Patterns $< \{1, 2\}, \{1\} >$ and $< \{2\}, \{1\}, \{1\} >$ are both subsequences of transaction sequence 500 and are 3-patterns, whereas $< \{2, 1\}, \{1\}, \{3\} >$ is not. Pattern $< \{1, 2\}, \{1\} >$ has supp($< \{1, 2\}, \{1\} >$) $= 3$ because it appears in transaction sequences 100, 200 and 500, thus it is considered frequent. Pattern $< \{2\}, \{1\}, \{1\} >$ only appears in transaction sequence 500, thus its supp($< \{2\}, \{1\}, \{1\} >$) $= 1 < \delta$ and it is infrequent.

4. The SUI algorithm

In this section, we present SUI (Sequential pattern mining Using Indices) and its structure. As we mentioned in the introduction, duplicate data often occur as a result of multiple data replication during the mining process. While [**22**] proposed an idea of pseudo-IDLists to avoid this issue for clickstream pattern mining, it is only partly compatible with sequential pattern mining. The reason is that clickstream patterns only have one item per itemset in their patterns, while sequential patterns can have multiple items per itemset. By the definition of sequential patterns, clickstream patterns are made up of multiple 1-itemsets. Thus, clickstream patterns always have s-extension but a sequential pattern, on the other hand, can have either s-extension or i-extension. CUP [**22**] is therefore only compatible with s-extension patterns, but not i-extension patterns. To deal with this issue, we propose a method (**Section 4.4**) to adopt this approach to sequential pattern mining. There are two different data structures for storing pattern information in SUI. The first one is data-IDList, where the actual position data of a pattern is kept. The second is pseudo-IDList, where necessary indices are kept to obtain the actual data from corresponding data-IDLists. Each data structure used depends on whether a pattern has an s-extension or an i-extension.

4.1. Data-IDList

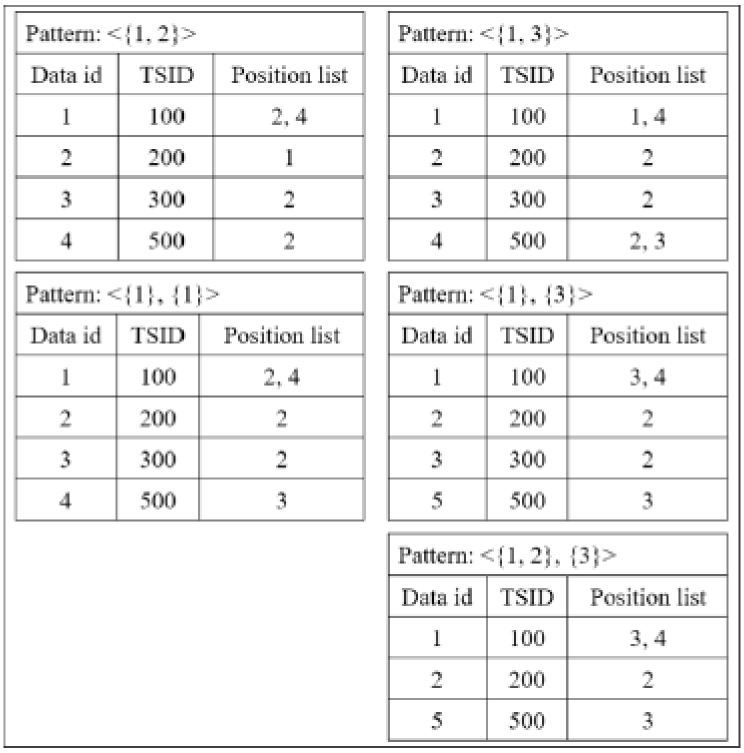
A data-IDList [**22**] stores the information of a pattern's occurrences in the horizontal database. The positions mainly consist of information about TSIDs (identifying which transaction sequences the pattern appears in) and position lists (at which positions in the transaction sequences). Based on [**17**], only the positions of the last item of the sequential pattern need storing. For example, let $X = < \{2, 4\}, \{1\} >$ be a sequential pattern, and $Y = < \{5\}, \{2, 4\}, \{3\}, \{1\}, \{1\} >$ be a transaction sequence, the position list of X in Y is <4, 5> A vertical database (**Fig. 1**) is basically a collection of data-IDLists. A data-IDList is made up of three elements P, M, and $supp$.

- P: the pattern.
- M: a collection of 3-tuples {Data id, TSID, Position list}. TSID is the transaction sequence id. The position list contains all the positions of the pattern's last item in the transaction sequence. A data id value is assigned for each transaction sequence, acting as a row index for pseudo-IDList. Different data-IDLists can assign a different value of data id to the same transaction sequence. For example, the data id of transaction sequence 500 is 4 in the data-IDList of $< \{1\} >$ but 5 in the data-IDList of $< \{2\} >$.
- $Supp$: the support count, which is the number of rows in M.

4.2. Pseudo-IDList

In most vertical SPM algorithms, duplicate data occurs mostly when creating data-IDLists for descending candidate patterns. For s-extension patterns, the children's data-IDList is a part of one of their parents' data-IDList (**Section 4.4** talks more about this). Instead of copying data to generate the new data-IDList, we can use indices to the original data-IDLists to retrieve the necessary information. Thus, pseudo-IDList [**22**] (e.g., Fig. 3) is proposed and instead of holding the real data, it holds all the necessary indices to another data-IDList to retrieve position data. It has four main elements as follows.

- P: the pattern.
- $DIP\ (data - IDList\ pointer)$: a pointer to a data-IDList of frequent 1-pattern which is the last item in P. For example, for a pattern $< \{1, 3\}, \{2\} >$ with a pseudo-IDList, the DIP points to the data-IDList of 1-pattern $< \{2\} >$.
- M: a collection of 3-tuples {Local id, Data id, Start index} that are indices. For each corresponding transaction sequence in the pseudo-IDList, it is assigned with an increment integer value called local id. The data id value matches a corresponding data id in the data-IDList (we can think of it as a row index to data-IDList). The start index is a start index location in a position list, from which we can retrieve the tail sublist (**Definition 1 Section 4.4**).
- $Supp$: the support count which is the number of rows in M.



Pattern: <{1, 2}>		
Data id	TSID	Position list
1	100	2, 4
2	200	1
3	300	2
4	500	2

Pattern: <{1, 3}>		
Data id	TSID	Position list
1	100	1, 4
2	200	2
3	300	2
4	500	2, 3

Pattern: <{1}, {1}>		
Data id	TSID	Position list
1	100	2, 4
2	200	2
3	300	2
4	500	3

Pattern: <{1}, {3}>		
Data id	TSID	Position list
1	100	3, 4
2	200	2
3	300	2
5	500	3

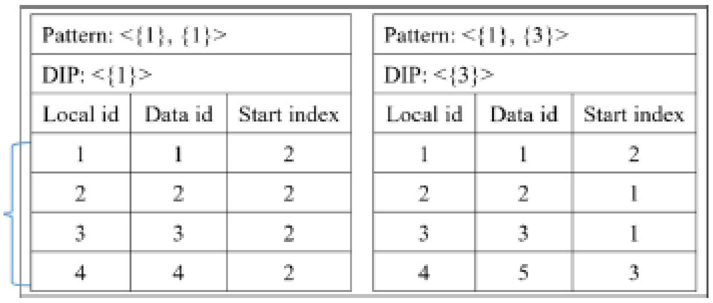
Pattern: <{1, 2}, {3}>		
Data id	TSID	Position list
1	100	3, 4
2	200	2
5	500	3

Fig. 2. Data-IDLists of some frequent patterns that share prefix <{1}>. The first two $<$ {1,2} $>$ and $<$ {1, 3} $>$ have i-extensions, and the other three, $<$ {1}, {1} $>$, $<$ {1}, {3} $>$ and $<$ {1, 2}, {3} $>$ have s-extensions.

4.3. Candidate generation

Most vertical algorithms employ a generate-and-test-candidate approach, and their candidates are formed by combining frequent patterns with lesser length. For example, let $X_1 = <$ {1, 3}, {2}, {4} $>$, $X_2 = <$ {1, 3}, {2}, {5} $>$ be two frequent patterns which share the same 3-prefix $<$ {1, 3}, {2} $>$. We can combine them and produce three pattern candidates, which are $Z_2 = <$ {1, 3}, {2}, {4}, {5} $>$, $Z_1 = <$ {1, 3}, {2}, {5}, {4} $>$, and $Z_3 = <$ {1, 3}, {2}, {4, 5} $>$. Z_1 and Z_2 have s-extensions and Z_3 has an i-extension. Z_1 and Z_3 have X_1 as its prefix and Z_2 has X_2 as its prefix. The candidate generation process can be either based on SPADE [**17**] or SPAM [**18**]. In this paper, we use the SPADE method. Briefly, candidates are generated according to the following rules:

- If both parent patterns X_1 and X_2 have s-extensions, the number of generated candidates is three and they are a mix of s and i-extensions. The previous paragraph shows an example of this case. There is one exception when both parents are the same pattern ($X_1 = X_2$), then there is only one s-extension candidate. For example, if $X_1 = X_2 = <$ {1, 3}, {2}, {4} $>$, there is only one s-extension candidate $<$ {1, 3}, {2}, {4}, {4} $>$.
- If both have i-extensions, there is only one i-extension candidate. For example, let $X_1 = <$ {1, 3}, {2, 3} $>$ and $X_2 = <$ {1, 3}, {2, 4} $>$, the generated candidate is $<$ {1, 3}, {2, 3, 4} $>$.
- If one of the parents X1 has i-extension and the other one X_2 has s-extension, there is only one s-extension candidate. For example, let $X_1 = <$ {1, 3}, {2, 4} $>$ and $X1 = <$ {1, 3}, {2}, {3} $>$, the generated candidate is $<$ {1, 3}, {2, 4}, {3} $>$.



Pattern: <{1}, {1}>		
DIP: <{1}>		
Local id	Data id	Start index
1	1	2
2	2	2
3	3	2
4	4	2

Pattern: <{1}, {3}>		
DIP: <{3}>		
Local id	Data id	Start index
1	1	2
2	2	1
3	3	1
4	5	3

Fig. 3. The respective pseudo-IDLists of s-extension patterns $<\{1\},\{1\}>$ and $<\{1\},\{3\}>$ in **Fig. 2**.

4.4. Adapting pseudo-IDList for sequential pattern mining

The idea of pseudo-IDLists is to eliminate the duplication of data for clickstream pattern mining. As mentioned above, a clickstream is a specific type of sequential pattern in which all itemsets contain only one item. In this section, we prove that pseudo-IDList works for s-extension patterns but not for i-extension patterns, thus making it not suitable for SPM without adjustments. Hence, we propose a method for adapting the idea of pseudo-IDList to work for sequential pattern mining.

Definition 1. Let $PL_x = < x_1, x_2 \cdots x_m >$, $PL_y = < y_n, y_{n+1} \cdots y_m >$ be two position lists and $1 \leq n \leq m$. PLy is a **tail sublist** of PL_X if $y_n = x_m$ $y_{n+1} = x_{n+1}, \ldots, y_m = x_m$.

For example, <4, 6> is a tail sublist of <1, 2, 4, 6> but <2, 6> is not a tail sublist of <1, 2, 4, 6>. Additionally, <4, 6> is also a tail sublist of itself.

Theorem 1. Let X and Y be two patterns having s-extensions and appearing in the same user transaction. If X is a sub-pattern of Y ($X \subset Y$), and the last item in X is equal to the last item in Y, then Y's position list is a tail sublist ofX's position list.

Proof. Let P_X be the $(k-1)$-prefix of X and Py be the $(k-1)$-prefix of Y, e is the last item of X (as well as Y), and S is a user transaction sequence in which X and Y both appear. Let prefix_pos(S, P_X, start_index) be a function that returns the first position of P_X (i.e. the first position of the itemset that contains the last item of P_X in S and satisfies the subsequence definition), and start_index is the position where the searching starts in S. Let include_item(i, e) be a function that returns true when item e is included at the ith itemset in S.

Because X (and Y) is an s-sequence, e's position is always > prefix_pos(S, P_X, 0). Thus, a position list can be formed by first finding the first occurrence of P_X in S. After this, for every position of e after P_X in S, they form the position list. In other words, the position list can be represented as an ordered set with an ascending ordering of integer numbers. The position list of X can be denoted as $PL_X = \{ i \mid \forall i \in Z^+$: include_item$(i, e)$ = true AND prefix_pos(S, P_X, 0) $< i <$ length of $S\}$. Similarly, $PL_Y = \{ i \mid \forall i \in Z^+$: include_item$(i, e)$ = true AND prefix_pos(S, P_Y, 0) $< i <$ length of $S\}$.

Because $X \sqsubset Y$ (X is a sub-pattern of Y), $P_X \sqsubset P_Y$, and length of $P_X <$ length of P_Y, the first occurrence of P_X always happens before or at the same location of P_Y. Or in other words, pattern_pos(S, P_X, 0) $\leq$ pattern_pos(S, P_Y, 0). We can rewrite the position list of X as:

PL_X = {i | ∀ i ∈ Z^+: $include\_item$(i, e) = true AND [prefix_pos(S, P_X, 0) $< i <$ $prefix\_pos$(S, P_Y, 0) OR $prefix\_pos$(S, P_Y, 0) $< i \leq$ length of S]}

= $\{i \mid \forall\, i \in Z^+$: $include\_item$(i, e) = true AND $prefix\_pos(S, P_X, 0) < i < prefix\_pos(S, P_Y, 0)\} \cup \{i \mid \forall\, i \in Z^+$: include_item(i, e) = true AND $prefix\_pos(S, P_Y, 0) < i \leq$ length of $S\}$

= $\{i \mid \forall$ i $\in Z^+$: include_item(i, e) = true AND $prefix\_pos(S, P_X, 0) < i < prefix\_pos(S, P_Y, 0)\} \cup$ PLY

Thus, PL_Y is a tail sublist of PL_X.

Example. Let $X = <\{3\}>$ and $Y = <\{1,2\}, \{3\}>$, considering the position list PL_X = <1,3, 4> for TSID = 100 in X's data-IDList and the position list PL_Y = <3, 4> for TSID = 100 in Y's data-IDList. We have PL_Y = <3, 4> $\subseteq PL_X$ = <1, $\underline{3, 4}$>, and L_Y is a tail sublist of L_x .

This theorem is important for our proposed method. Because when pseudo-IDLists retrieve position information from data-IDList, the position information must always be tail sublists (i.e. the continuous array of elements from a start index to the end of a position list) as in the example above. So if s-extension patterns do not satisfy this condition (i.e. the theorem does not hold), the idea of pseudo-IDList does not work for our proposed method.

Lemma 1. For every $sub-pattern\ X$ of a pattern Y ($X \subset Y$), if X and Y are both s-extensions, and the last item in X is equal to the last item in Y, then every position list in Y's data-IDList is a tail sublist of another position list in X's data-IDList with the same TSID.

Example. Let $Y = <\{1, 2\}, \{3\}>$, then all sub-patterns of Y that have s-extensions are $<\{1\}>$, $<\{2\}>$, $<\{3\}>$, $<\{1\}, \{3\}>$, and $<\{1\}, \{2\}>$. However, only $<\{1\}, \{3\}>$ and $<\{3\}>$ have the same last item $\{3\}$ as Y. The data-IDList of Y is shown in **Fig. 2** and every position list of it is a tail sublist of either $<\{1\}, \{3\}>$, or $<\{3\}>$'s data-IDLists (**Fig. 1** and **Fig. 2**).

Lemma 2. For every k-pattern Y ($k \geq 2$) that has an s-extension and exists in a database, there always exists a frequent 1-pattern X such that $X \subset$ Y and every position list in Y's data-IDList is a tail sublist of another position list in X's data-IDList with the same TSID.

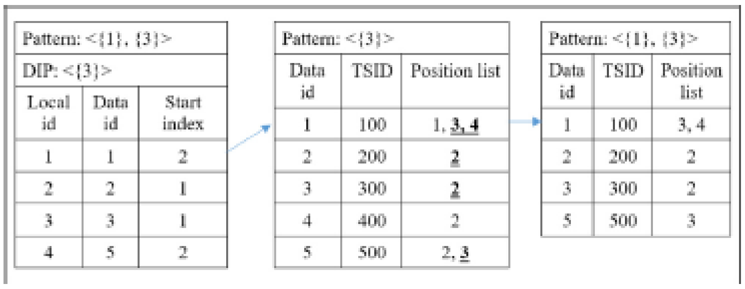
Lemma 2 is deduced directly from Lemma 1. This lemma means that data-IDLists of frequent 1-patterns are sufficient to provide position information for all k-patterns ($k \geq 2$) that have s-extensions. We can thus avoid generating unnecessary data-IDLists.

Based on the above theorem and lemmas, pseudo-IDList is proven to work on s-extension patterns. However, it does not work on patterns with i-extensions. This can be proved via the following counter example. Let $X = <\{3\}>$ and $Y = <\{1, 3\}>$ be two i-extension patterns, the position list PL_Y = <1, 4> for TSID = 100 in Y's data-IDList and the position list PLX = <1, 3, 4> for TSID = 100 in X's data-IDList. We can see that PL_Y = <1, 4> $\subseteq$ PLX = <1, 3, 4> but PL_Y is not a tail sublist of PL_X. Hence, the idea of pseudo-IDList fails in this case.

We propose the following changes to make the idea work for sequential pattern mining:

- We use data-IDLists to directly hold the position data for i-extension patterns as pseudo-IDLists do not work for i-extension patterns.
- We continue to use pseudo-IDLists for s-extensions patterns as they are proved to still work on them.

IDList Generation In the original idea in [**22**], pseudo-IDLists are always produced in the IDList generation process because all clickstreams are s-extension patterns.



Pattern: <{1}, {3}>		
DIP: <{3}>		
Local id	Data id	Start index
1	1	2
2	2	1
3	3	1
4	5	2

Pattern: <{3}>		
Data id	TSID	Position list
1	100	1, 3, 4
2	200	2
3	300	2
4	400	2
5	500	2, 3

Pattern: <{1}, {3}>		
Data id	TSID	Position list
1	100	3, 4
2	200	2
3	300	2
5	500	3

Fig. 4. The data retrieval of a pseudo-IDList for pattern $< \{1\}, \{3\} >$. The first row in the pseudo-IDList of $< \{1\}, \{3\} >$ points to the first row in data-IDList of pattern $< \{3\} >$ because the data id values are both equal to 1. The start index value in the first row is 2, so it retrieves the continuous $sub - list$ <**3, 4**> of <1, 3, 4> (then the first row in data-IDList of <{1}, {3}> can be simulated based on retrieved data). Similarly, it retrieves <2>, <2> and <3> in the second, third and fifth rows to stimulate the other remaining rows in data-IDList of $< \{1\}, \{3\} >$.

However, our candidate generation can produce either i-extension or s-extension candidate patterns. Thus, our IDList generation process needs to produce a pseudo-IDList if the candidate pattern has an s-extension or a data-IDList if otherwise. The process of generating pseudo-IDLists for s-extensions is illustrated in Algorithm 1 and the process of generating data-IDLists for i-extensions is shown in Algorithm 2. Note that the length of those s- and i-extension patterns must be longer than one.

Two cases can happen in Algorithm 1 for parent patterns X and Y. If X and Y both have s-extensions, both X and Y's IDLists are pseudo-IDLists. The other case is that one of them has an i-extension and the other one has an s-extension. In such a case, we assume that X always has the i-extension, and Y always has the s-extension. There are also two cases in Algorithm 2, which are X and Y are either both s-extensions or both i-extensions (see **Fig. 4**).

Algorithm 1. Pseudo-IDList generation for an *s*-extension candidate pattern

```
    Input: IDLists of $X$ and $Y$
    Output: pseudo-IDList of $Z$
1:  Let $M_X$ be $X$'s IDList // $M_X$ can be either a data-IDList or a pseudo-IDList
2:  Let $M_Y$ be $Y$'s pseudo-IDList, let $M_Z$ be $Z$'s pseudo-IDList and initialize $M_Z$ to initial values
3:  $local\_id_Z \leftarrow 1$; $r_X \leftarrow$ first row in $M_X$; $r_Y \leftarrow$ first row in $M_Y$;
 /* We visit each row in $M_X$ and $M_Y$ using $r_X$ and $r_Y$ */
4:  while either $M_X$ or $M_Y$ still has unvisited row do
5:      if $X$ has an s-extension //$r_X$ contains {$Local\_id$, $Data\_id$, $Start\_index$}
6:          Use $r_X$ to retrieve the corresponding $r'_X$ = {$Data\_id$, $TSID$, $Position\_list$} //Following the example in Fig. 4
7:      else $r'_X \leftarrow r_X$ // $r_X$ contains {$Data\_id$, $TSID$, $Position\_list$}
8:      if $r'_X.TSID < r_Y.TSID$ then // Elements of a row is accessed by a dot (.)
9:          Move $r_X$ to the next row in $M_X$ and loop back to step 4 // Visit the next row in $M_X$
10:     else if $r'_X.TSID > r_Y.TSID$ then
11:         Move $r_Y$ to the next row in $M_Y$ and loop back to step 4 // Visit the next row in $M_Y$
12:     else if $r'_X.TSID = r_Y.TSID$ then
13:         $e_X \leftarrow$ the first element in $r_X.Position\_list$
14:         Use $r_Y$ to retrieve the corresponding $r'_Y$ = {$Data\_id$, $TSID$, $Position\_list$}
    /* After we found the rows containing duplicate information, we search for where the duplicate information occurs in the rows*/
15:         $offset\_index \leftarrow r'_Y.Start\_index$
16:         for each element $e_Y$ in $r'_Y.Position\_list$ do
17:             if $e_Y > e_X$ then // Found the start of duplicate data
18:                 Add a row containing {$local\_id_Z$, $r'_Y.Data\_id$, $offset\_index$} to $M_Z$
19:                 break; // Stop searching for duplicate information in $r'_Y.Position\_list$
20:             else $offset\_index \leftarrow offset\_index + 1$
21:         $local\_id_Z \leftarrow local\_id_Z + 1$
22:         Move $r_X$ to the next row in $M_X$ and $r_Y$ to the next row in $M_Y$
23: $Z$'s $DIP \leftarrow Y$'s $DIP$ // DIP of Z points to the same data-IDList that Y's points to
24: return $M_Z$
```

Algorithm 2. Data–IDList generation for an *i*–extension candidate pattern

```
    Input: IDList of $X$ and $Y$
    Output: $Z$'s Data–IDList
1:  if $X$ and $Y$ have s–extensions
2:      Let $M_X$ and $M_X$ be the data–IDLists that $X$'s DIP and $Y$'s DIP point to, respectively
3:  else
4:      Let $M_X$ and $M_X$ be $X$'s data–IDList and $Y$'s data–IDList, respectively
5:  $r_X \leftarrow$ first row in $M_X$; $r_Y \leftarrow$ first row in $M_Y$; // Both rows contain {Data_id, TSID, Position_list}
    /* We visit each row in $M_X$ and $M_Y$ using $r_X$ and $r_Y$ */
6:  while either $M_X$ or $M_Y$ still has an unvisited row do
7:      if $r_X$.TSID < $r_Y$.TSID then
8:          Move $r_X$ to the next row in $M_X$ and loop back to step 6 // Visit the next row in $M_X$
9:      else if $r_X$.TSID > $r_Y$.TSID then
10:         Move $r_Y$ to the next row in $M_Y$ and loop back to step 6 // Visit the next row in $M_Y$
11:     else if $r_X$.TSID = $r_Y$.TSID then
12:         $plist_Z \leftarrow \emptyset$ // An empty position list
13:         for each element $e_X$ starting in an ascending order in $r_X$.Position_list do
14:             for each element $e_Y$ starting in an ascending order in $r_Y$.Position_list do
                if $e_X = e_Y$ then   /* Only obtain the same positions in both lists */
15:                     Add $e_X$ to $plist_Z$
16:         if $plist_Z$ is not empty then
17:             Add a row containing {$r_X$.Data_id, $r_X$.TSID, $plist_Z$} to $M_Z$
18: return data–IDList of $Z$
```

To estimate SUI complexity is very complicated, however, most of the resource-consuming steps for vertical algorithms are in the IDList creation step. We can thus estimate how well the algorithms perform via the complexity of the IDList creation step. Let X and Y be two patterns with the respective IDLists M_X and M_Y, the numbers of elements in M_X and M_Y are n and m, the fastest IDList creation can happen when the new candidate P_3 is an s-extension (Algorithm 1), which is Ω (supp(X) + supp(Y)). This happens when the duplication indices are all first elements in s-extension IDLists. The worst time for Algorithm 1 is O(supp(X) + m) when the creation process has to navigate all the elements in the IDList of Y. For Algorithm 2, the worst runtime has the complexity of O($n + m$). Therefore, the worst case of the ID generation process happens when the pattern candidate is i-extension. The whole mining process is slowest when the database is a pure itemset database (i.e. no s-extension pattern exists in the database). In reality, a sequential database should contain both s-extension patterns and i-extension patterns. Therefore, the complexity of the IDList creation of SUI would be between Ω(supp(X) + supp(Y)) and O(n + m). This contrasts with the general IDList creation used in other methods that would always take up the runtime in between Ω (supp(X) + m) and O(n + m).

4.5. *The SUI algorithm*

The main steps of SUI's mining process are:

- **Step 1**. Scanning the whole horizontal database and gathering all frequent 1-patterns (which are considered frequent i-extension 1-patterns) and their data-IDLists (**Section 4.1**).
- **Step 2**. Generating candidate patterns (which have either i-extensions, s-extensions, or both) by combining two k-patterns that have the same (k-1)-prefix (**Section 4.3**). Frequent 1-patterns are considered to share an empty prefix.
- **Step 3**. Generating pseudo-IDLists and data-IDLists for the candidates in the previous steps, and checking the candidates' supports for minimum support requirement. If any candidate's

support $<\ \delta$, they are discarded. The candidate's support can be obtained via their IDLists instead of scanning the whole database like in Step 1. Producing pseudo-IDLists and data-IDLists is considered taking a large portion of the algorithm runtime. After this, we return to Step 2 and repeat until no candidates can be found.

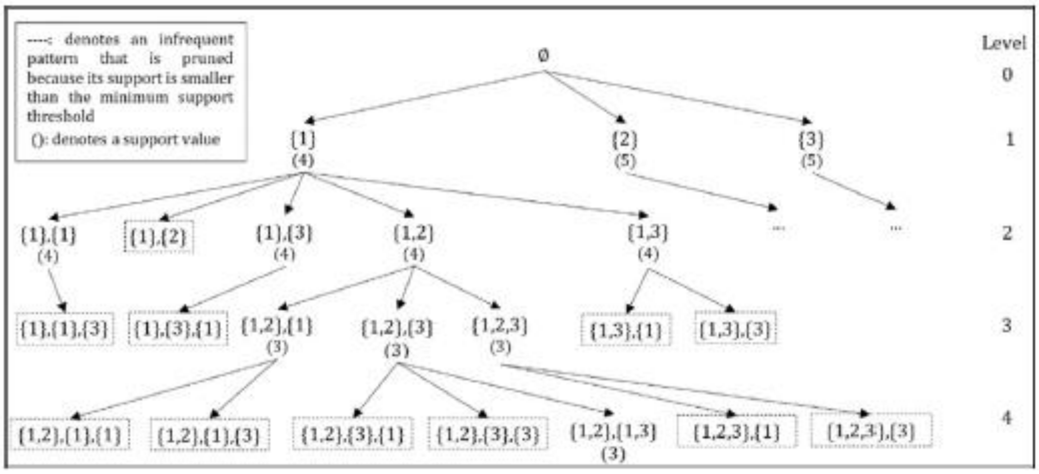


Fig. 5. A part of the pattern tree based on the example database.

For example, assuming that we use the database SDB in **Table 1** with 8 = 3. A part of the pattern tree is shown in **Fig. 5** and the mining process executes as follows.

Step 1. We detect the set of frequent 1-patterns {<{1} $>$, $<$ {2}>, $<$ {3}>} and their supports {4, 5, 5} respectively via scanning the database SDB. In this step, all pseudo-IDLists and data-IDLists for those frequent 1-patterns are also generated.

Step 2. By combining $<$ {1} $>$ with $<$ {1} $>$, $<$ {1} $>$ with $<$ {2} $>$ and $<$ {1} $>$ with $<$ {3} $>$, we create a set of 2-candidates {<{1}, {1} $>$, $<$ {1}, {2} $>$, $<$ {1}, {3} $>$, $<$ {1, 2} $>$, $<$ {1, 3}>} that use the same 1-prefix $<$ {1} $>$.

Step 3. In this step, as $<$ {1}, {1} $>$, $<$ {1}, {2} $>$ and $<$ {1}, {3} $>$ are s-extension candidates, we generated pseudo-IDLists for all of them. Meanwhile, $<$ {1, 2} $>$ and $<$ {1, 3} $>$ are i-extension candidates, so data-IDLists are generated instead. After this their supports are computed as {4, 2, 4, 4, 4}, respectively, although candidate $<$ {1}, {2} $>$ is removed because its support is 2 $< \delta$ = 3 and the remaining candidates are frequent patterns because their supports $> \delta$ = 3. SUI then repeats step 2, but this time it works on the frequent 2-pattern set {<{1}, {1} $>$, $<$ {1}, {3} $>$, $<$ {1, 2} $>$, $<$ {1, 3} $>$ }. The algorithm continues to repeat those steps and stops after no more candidates can be generated in the $<$ {3} $>$ branch.

5. Experimental results

We evaluate our SUI algorithm by comparing its performance and scalability to PrefixSpan [**21**], and CM-SPADE [**19**], which are two state-of-the-art SPM algorithms. The PrefixSpan algorithm belongs to the horizontal group while the CM-SPADE algorithm is in the vertical group. According to [**19**], both are very effective algorithms. All algorithms are implemented in Java, and PrefixSpan (version 2016) and CM-SPADE come from the SPMF package [50]. The computer used for the experiments was equipped with an Intel Core I7 8750H 2.2 GHz, 16 GB RAM, Windows 10 64-bit, and JDK 8.

Three real-life databases are used, in which Kosarak and MSNBC are considered big databases and FIFA a medium database. MSNBC is also considered a dense database, while FIFA and Kosarak are sparser. They can be obtained via the following link **https://www. philippe-fournier-viger.com/spmf/index.php?link=datasets.php**. Three synthetic test databases are also used for the experiments, C50S15T3, C150S40T2, and C200S12T5. The database C50S15T3 is considered small, while the other two are medium. Those synthetic databases are generated by following the standard generator in [**1**].

Table 2 Test database summary.

Database	Database size	Distinct items	Average sequence length
C50S15T3	49,060	2,611	39.97
C150S40T2	150,000	954	76.64
C200S12T5	183,950	1,922	51.57
FIFA	20,450	2,990	36.24
Kosarak	990,002	41,270	8.1
MSNBC	989,818	17	4.75

The databases' characteristics are summarized in **Table 2**. To evaluate the runtime (**Fig. 6**) and memory footprint (**Fig. 7**), all the mentioned algorithms are executed on six databases while the value of 8 is decreased. Note that we also integrate the CMAP data structure [**19**] and DUB heuristic [**22**] to prune candidates in SUI for optimization reasons.

Performance comparison. Via the results in **Fig. 6**, SUI is seen to generally perform better than both PrefixSpan and CM-SPADE in terms of runtime, and the gap in runtime between those algorithms is larger when 8 becomes smaller. SUI's runtime increases in a more gradual manner than the other two algorithms towards the smaller value of δ. For example, PrefixSpan has a lower runtime than the other two on the C200S12T5 dataset at δ = 0.7% and 0.6%. However, because PrefixSpan has a steeper increase in runtime, SUI and CM-SPADE have lower runtime in the end when $\delta \leq$ 0.5%. At δ = 0.3%, and SUI has the fastest runtime among the three. Even though SUI and CM-SPADE have several similarities in their mining process, SUI still has less runtime and a lower runtime increment rate than CM-SPADE. The reason can be explained via the data structure used. SUI does not have to replicate duplicate data multiple times in the mining process with its pseudo-IDList approach. Additionally, instead of storing real data, storing indices results in less memory footprint alongside performance advantages. **Fig. 7** shows the memory consumption difference between CM-SPADE and PrefixSpan and SUI. CM-SPADE generally has the highest memory consumption, while SUI and PrefixSpan use lower amounts of memory. This means SUI is efficient in terms of runtime, while still can manage to achieve a low memory footprint.

Scalability. To see how SUI scales with bigger databases, we scale up the databases by following the standard method in [**1**]. There are three parameters that we consider in the experiments, which are the database size (i.e. the number of transaction sequences in a database), the average sequence size (i.e. the average number of itemsets in transaction sequences), and the average itemset size (i.e. the average number of items per item-set in the database).

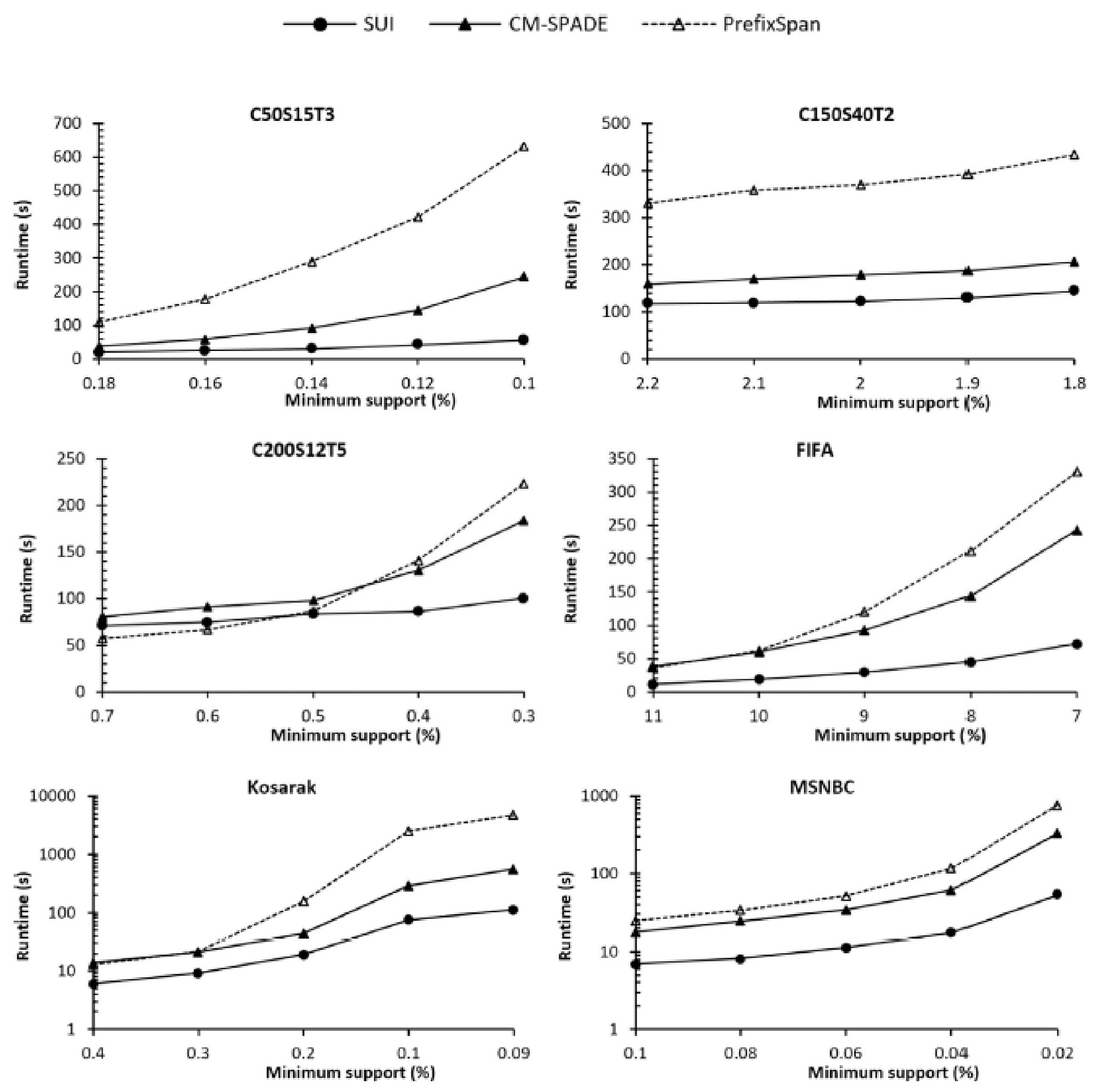


Fig. 6. Runtime on six databases.

The database name denotes parameters and their values as follows: C<Number of transactions in thou-sands>S<Average number of itemsets per transaction sequence >T<Average number of items per itemset>. For example, C200S20T2 means it is a database generated with parameters of 200,000 transaction sequences, 20 itemsets on average for a transaction sequence, and two items on average for each itemset.

The way we carry out the scalability tests in these experiments is to increase the value of one parameter at a time while keeping the values of two other parameters fixed. Using C200S20T2 (**Figs. 8** and **9**) and C200S10T2 (Fig. 10) as bases, we can see the growth rate of the algorithms when the databases scale up. When either the average sequence size or the average itemset size in a database increases, the database is denser. We only tested the scaling of two algorithms, SUI and PrefixSpan, because CM-SPADE could not run on most of the scalability databases. 8 is kept as 0.1% in all scalability tests.

Generally, the runtime of SUI is lower than that of PrefixSpan. When the database sizes increase, but the average sequential size and average itemset size are fixed, both algorithms appear to have similar runtime growth rates, as shown in **Fig. 8**. They both grow linearly. When only the sequence size increases (Fig. 9), both algorithms' growth rates rise a little faster towards bigger sequence size, but SUI still has a smaller growth rate than PrefixS-pan. Similarly, both algorithms are also sensitive to the change in itemset size (**Fig. 10**) as the growth rate doubles when the itemset size increases from 5 to 6. However, SUI’s growth rate is 1/4 that of PrefixSpan as SUI jumps from 5.6 to 11.3 compared with PrefixSpan, which jumps from 23.5 to 45.8.

Regarding memory consumption, there are some interesting results as SUI uses more memory at first, but as the databases grow (i.e. when one of the parameters increases) SUI starts to use less memory. The growth rate for the maximum memory of SUI is a smoother and more linear, while PrefixSpan is steeper. We observe some sudden jumps in memory for PrefixSpan when the average sequence size and average itemset size change.

The larger the databases, the more duplicate information they contain, and pseudo-IDLists eliminate some of this duplicate information which saves memory and reduces copying operations. Theoretically, the effectiveness of SUI scales with the number of s-extension patterns in the databases and the size of the position lists. These two factors depend on the average sequence size. The longer the sequence size, the longer the position lists can be for the s-extension pattern. And the more elements the position lists have, the more effective pseudo-IDList. This is because each position list of s-extension patterns with length k ($k > 2$) can be produced with an index value to a position list in a data-IDList (as in **Lemma 2**). This can explain why in Fig. 9 SUI has a smoother memory growth rate than PrefixSpan when the average sequence size increases. The same can be seen in **Fig. 10**. Because when the average itemset size increases, it also increases the number of s-extension patterns and duplicate information (i.e. there are more elements) in the position lists. Thus, we can see that SUI deals better with denser databases.

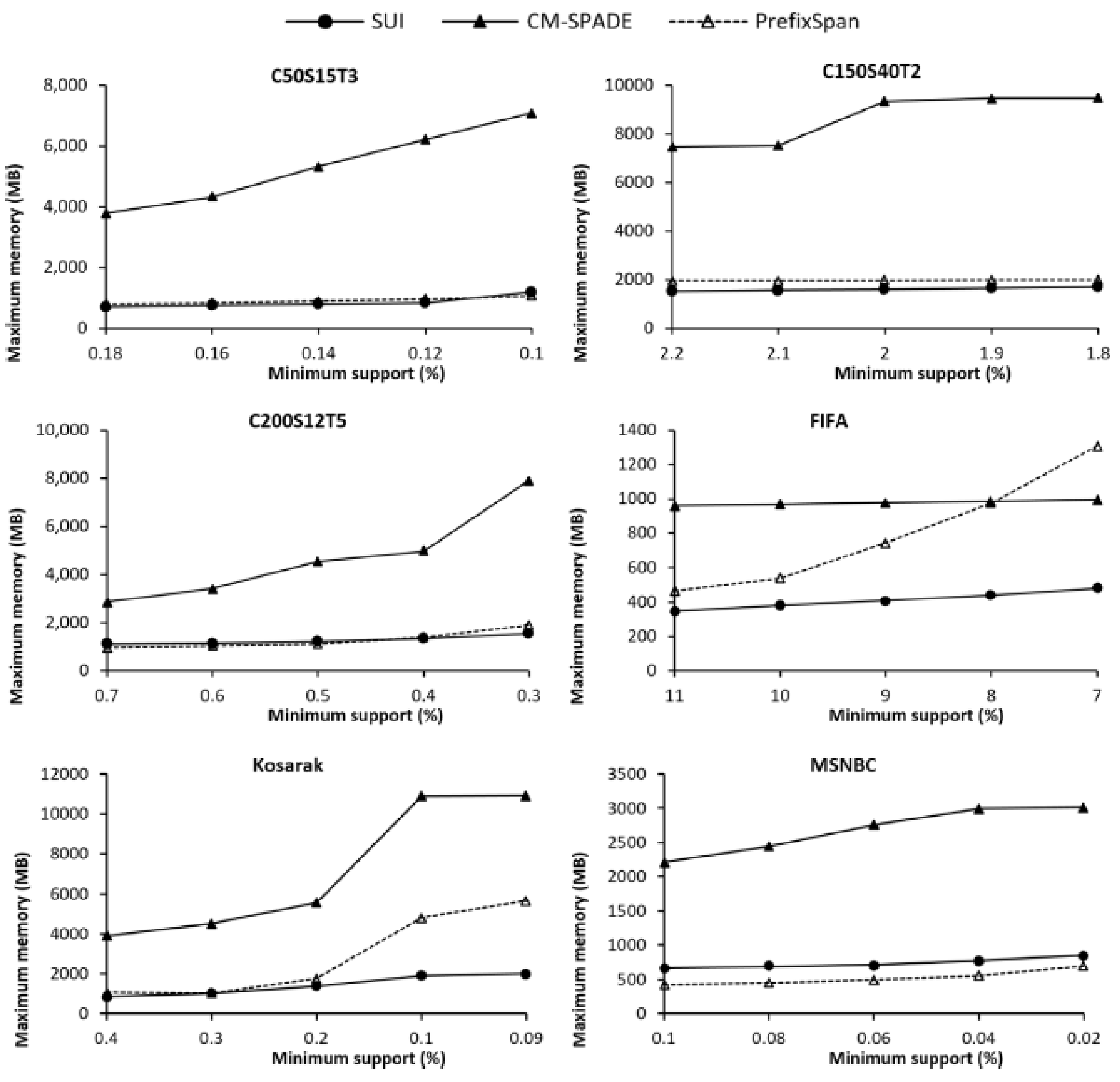


Fig. 7. Memory footprint on six databases.

6. Conclusion

SPM applications can potentially cover many domains, and many vertical algorithms have been proposed for the problem as well as its branches. An issue with vertical algorithms is that they can often produce duplicate data in their processes. Thus, the idea of pseudo-IDLists, which use indices instead of storing real data, is proposed in [23] to lessen the duplicate data problem. However, the original algorithm only works on clickstream patterns. In this paper, we proposed adaptations and changes in the original algorithm CUP to present SUI, an SPM algorithm that inherits the advantages of pseudo-IDLists. Via six test databases, SUI is shown to have better performance than both CM-SPADE and PrefixSpan, two efficient algorithms for SPM.

One issue of the current pseudo-IDLists is that they cannot help with removing duplicate data for i-extension patterns. In future work, we plan to further study this issue and propose an upgraded version of pseudo-IDLists so that they will fully work for both s- and i-extension patterns. Additionally, we would like to increase the performance of SUI via parallelism or grid computing to work on even bigger databases.

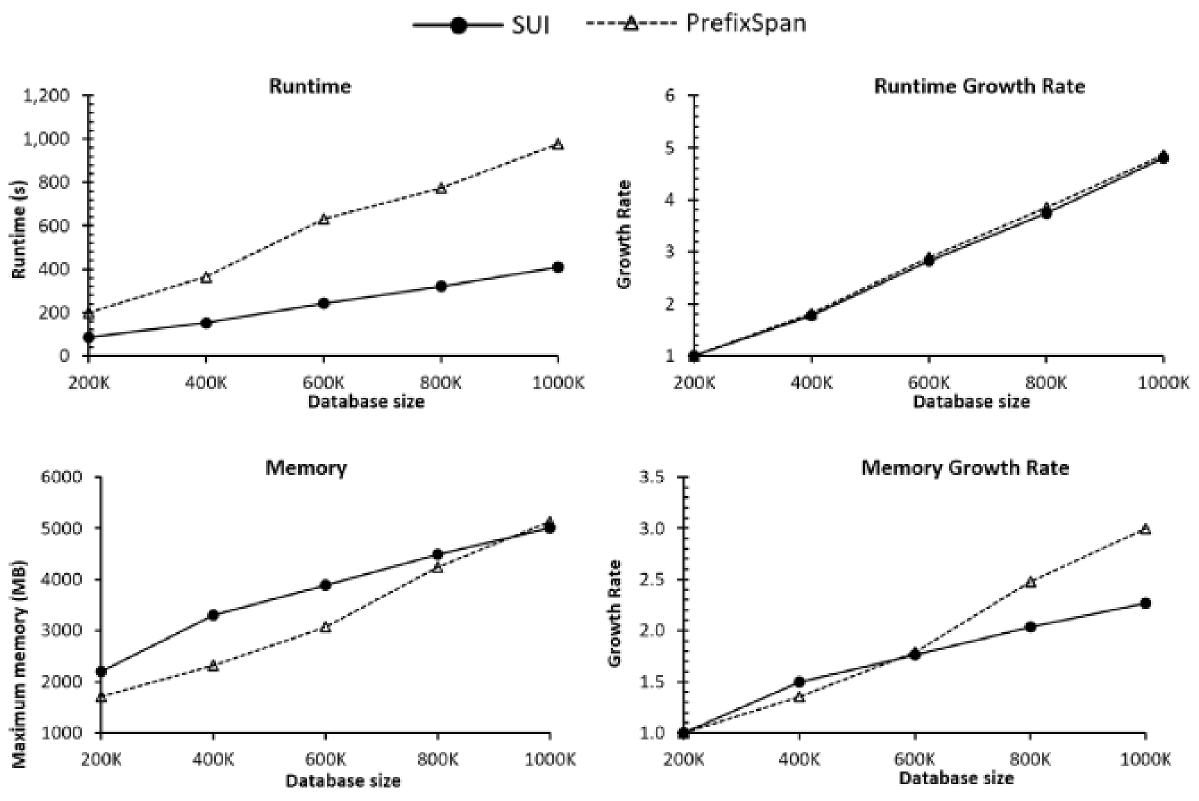


Fig. 8. Increasing the number of sequences.

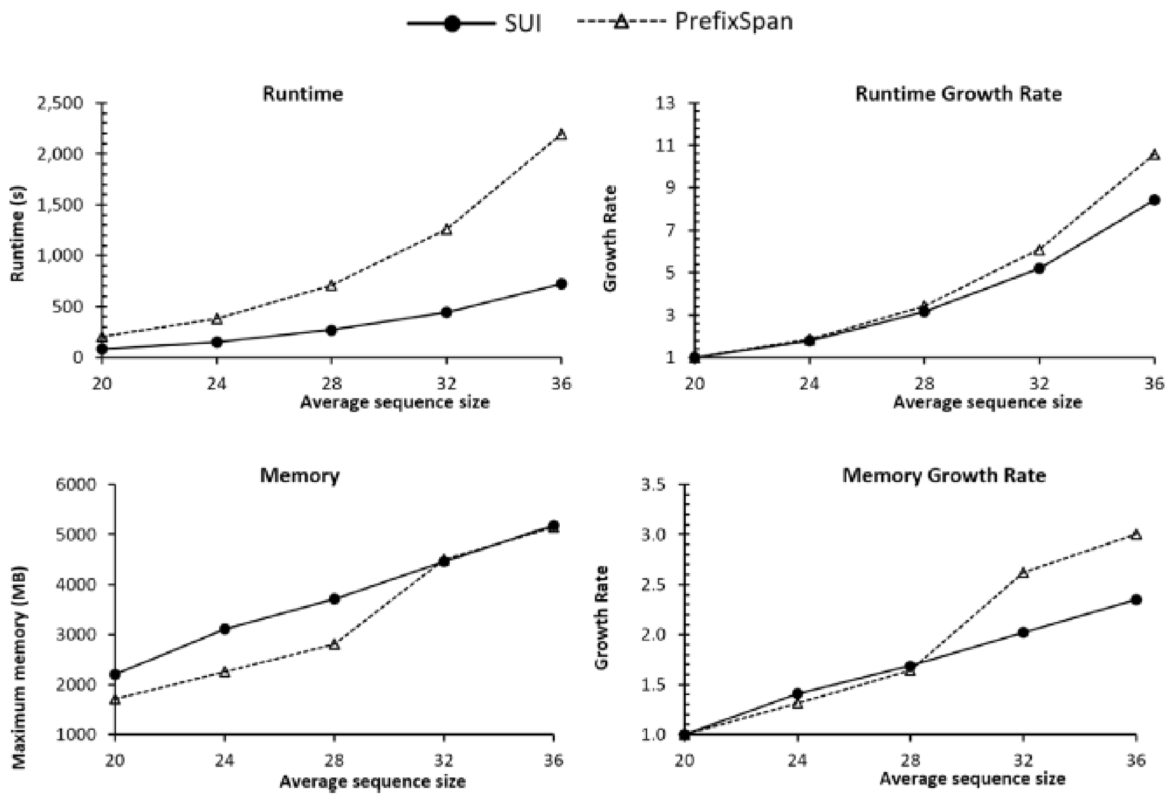


Fig. 9. Increasing the number of average itemsets per sequence.

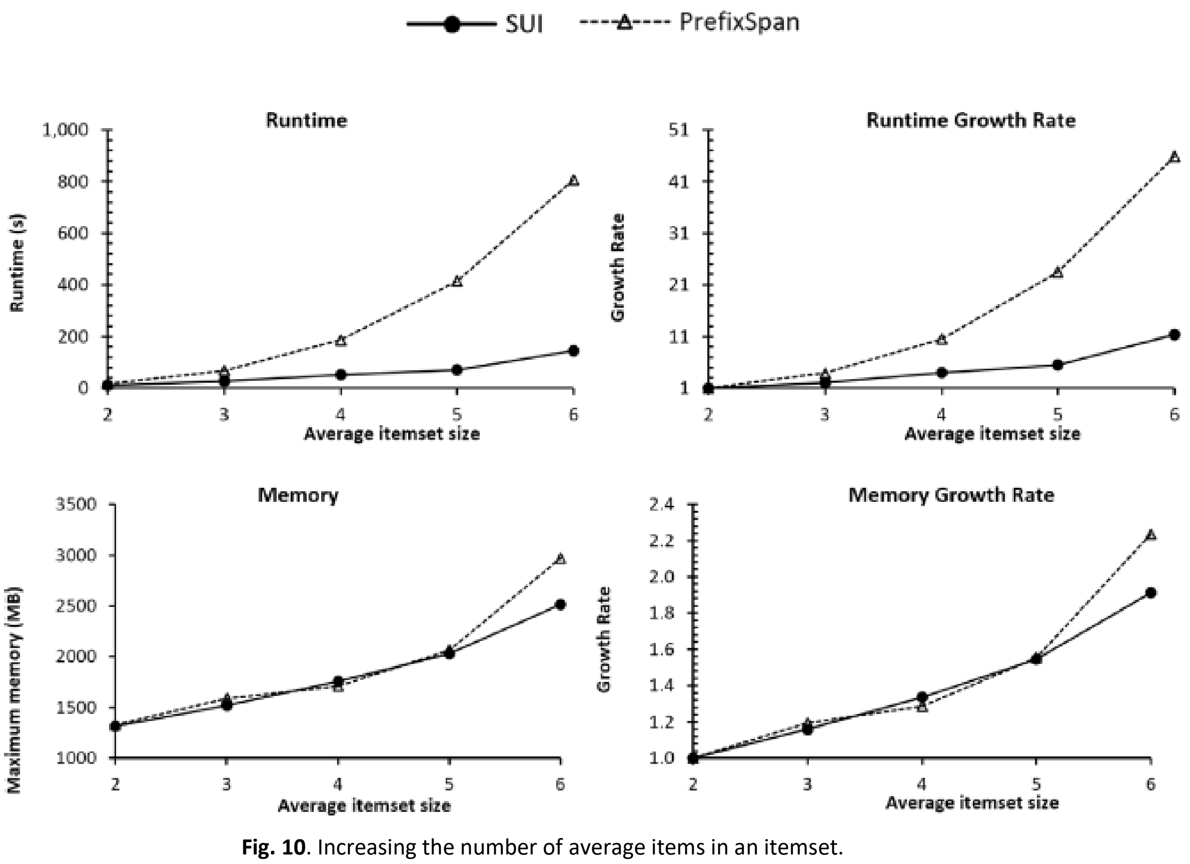


Fig. 10. Increasing the number of average items in an itemset.

References

[1] R. Agrawal, R. Srikant, Mining sequential patterns, in: Proceedings of the International Conference on Data Engineering, ICDE, 1995, pp. 3-14, **http://dx.doi.org/10.1109/ICDE.1995.380415**.

[2] B. Huynh, C. Trinh, H. Huynh, T.T. Van, B. Vo, V. Snasel, An efficient approach for mining sequential patterns using multiple threads on very large databases, Eng. Appl. Artif. Intell. 74 (2018) 242-251, **http://dx.doi. org/10.1016/j.engappai.2018.06.009**.

[3] W. Gan, J.C.-W. Lin, P. Fournier-Viger, H.-C. Chao, P.S. Yu, A survey of parallel sequential pattern mining, ACM Trans. Knowl. Discov. Data. 13 (2019) 1-34, **http://dx.doi.org/10.1145/3314107**.

[4] P. Fournier-Viger, J.C.W. Lin, R.U. Kiran, Y.S. Koh, R. Thomas, A survey of sequential pattern mining, Data Sci. Pattern Recognit. 1 (2017) 54-77.

[5] T. Van, B. Vo, B. Le, Mining sequential patterns with itemset constraints, Knowl. Inf. Syst. 57 (2018) 311-330, **http://dx.doi.org/10.1007/s10115-018-1161-6**.

[6] T. Le, A. Nguyen, B. Huynh, B. Vo, W. Pedrycz, Mining constrained intersequence patterns : a novel approach to cope with item constraints, Appl. Intell. 48 (2018) 1327-1343, **http://dx.doi.org/10.1007/s10489-017-1123-9**.

[7] W. Gan, J.C.-W. Lin, P. Fournier-Viger, H.-C. Chao, P.S. Yu, HUOPM: High-utility occupancy pattern mining, IEEE Trans. Cybern. 50 (2020) 1195-1208, **http://dx.doi.org/10.1109/TCYB.2019.2896267**.

[8] J.C.-W. Lin, Y. Li, P. Fournier-Viger, Y. Djenouri, J. Zhang, Efficient chain structure for high-utility sequential pattern mining, IEEE Access. 8 (2020) 40714-40722, **http://dx.doi.org/10.1109/ACCESS.2020.2976662**.

[9] W. Gan, J.C.-W. Lin, J. Zhang, H.-C. Chao, H. Fujita, P.S. Yu, ProUM: Projection-based utility mining on sequence data, Inf. Sci. (Ny). 513 (2020) 222-240, **http://dx.doi.org/10.1016/j.ins.2019.10.033**.

[10] M.G.H. Al Zamil, S.M.J. Samarah, M. Rawashdeh, M.A. Hossain, An ODT-based abstraction for mining closed sequential temporal patterns in IoT-cloud smart homes, Cluster Comput. 20 (2017) 1815-1829, **http://dx. doi.org/10.1007/s10586-017-0837-0**.

[11] B. Le, H. Duong, T. Truong, Fournier-Viger, FGenSM: Two efficient algorithms for mining frequent closed and generator sequences using the local pruning strategy, Knowl. Inf. Syst. 53 (2017) 71-107, **http://dx.doi.org/10. 1007/s10115-017-1032-6**.

[12] P. Fournier-Viger, P. Yang, Z. Li, J.C.-W. Lin, R.U. Kiran, Discovering rare correlated periodic patterns in multiple sequences, Data Knowl. Eng. 126 (2020) 101733, **http://dx.doi.org/10.1016/j.datak.2019.101733**.

[13] B. Dalmas, P. Fournier-Viger, S. Norre, TWINCLE: A constrained sequential rule mining algorithm for event logs, Procedia Comput. Sci. 112 (2017) 205-214, **http://dx.doi.org/10.1016/j.procs.2017.08.069**.

[14] H.M. Huynh, L.T.T. Nguyen, B. Vo, Z.K. Oplatkova, T.-P. Hong, Mining clickstream patterns using idlists, in: Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics, 2019, **http: //dx.doi.org/10.1109/SMC.2019.8914086**.

[15] S.C. Hsueh, M.Y. Lin, C.L. Chen, Mining negative sequential patterns for e-commerce recommendations, in: Proceedings of the IEEE Asia-Pacific Services Computing Conference, APSCC, 2008, pp. 1213-1218, **http://dx. doi.org/10.1109/APSCC.2008.183**.

[16] J.K. Tarus, Z. Niu, A. Yousif, A hybrid knowledge-based recommender system for e-learning based on ontology and sequential pattern mining, Futur. Gener. Comput. Syst. 72 (2017) 37-48, **http://dx.doi.org/10.1016/j. future.2017.02.049**.

[17] M.J. Zaki, SPADE: An efficient algorithm for mining frequent sequences, Mach. Learn. 42 (2001) 31-60, **http://dx.doi.org/10.1023/A: 1007652502315**.

[18] J. Ayres, J. Flannick, J. Gehrke, T. Yiu, Sequential pattern mining using a bitmap representation, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM Pres, New Yorks, New York, USA, 2002, pp. 429-435, **http://dx.doi.org/10.1145/ 775047.775109**.

[19] P. Fournier-Viger, A. Gomariz, M. Campos, R. Thomas, Fast vertical mining of sequential patterns using co-occurrence information, in: Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2014, pp. 40-52, **http://dx.doi.org/10.1007/978-3-319-06608-0_4**.

[20] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, Freespan: Frequent pattern-projected sequential pattern mining, in: In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining., 2000, pp. 355-359, **http://dx.doi.org/10.1145/347090.347167**.

[21] J. Pei, J. Han, Q. Chen, M.-C. Hsu, B. Mortazavi-Asl, H. Pinto, U. Dayal, Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth, in: In: Proceedings of the International Conference on Data Engineering, (ICDE), 2001, pp. 215-224, **http://dx.doi.org/10.1109/ICDE**.

22] H.M. Huynh, L.T.T. Nguyen, B. Vo, U. Yun, .Z.K. Oplatková, T.-P. Hong, Efficient algorithms for mining clickstream patterns using pseudo-idlists, Futur. Gener. Comput. Syst. 107 (2020) 18-30, **http://dx.doi.org/10.1016/j. future.2020.01.034**.

[23] H.M. Huynh, N.N. Pham, .Z.K. Oplatková, L.T.T. Nguyen, B. Vo, Sequential pattern mining using idlists, in: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2020, pp. 341-353, **http://dx.doi.org/10.1007/978-3-030-63007-2_27**.

[24] R. Agrawal, T. Imieliński, A. Swami, Mining association rules between sets of items in large databases, in: : Proceedings of the ACM SIGMOD International Conference on Management of Data., 1993, pp. 207-216, **http://dx.doi.org/10.1145/170036.170072**.

[25] B. Vo, L.V. Nguyen, V.V. Vu, M.T.H. Lam, T.T.M. Duong, L.T. Manh, T.T.T. Nguyen, L.T.T. Nguyen, T.-P. Hong, Mining correlated high utility itemsets in one phase, IEEE Access. 00 (2020) 1, **http://dx.doi.org/10.1109/ACCESS. 2020.2994059**.

[26] U. Yun, H. Nam, J. Kim, H. Kim, Y. Baek, J. Lee, E. Yoon, T. Truong, B. Vo, W. Pedrycz, Efficient transaction deleting approach of pre-large based high utility pattern mining in dynamic databases, Futur. Gener. Comput. Syst. 103 (2020) 58-78, **http://dx.doi.org/10.1016/j.future.2019.09.024**.

[27] L.T.T. Nguyen, V.V. Vu, M.T.H. Lam, T.T.M. Duong, L.T. Manh, T.T.T. Nguyen, B. Vo, H. Fujita, An efficient method for mining high utility closed itemsets, Inf. Sci. (Ny). 495 (2019) 78-99, **http://dx.doi.org/10.1016/j.ins.2019.05.006**.

[28] P. Fournier-Viger, Y. Yang, J.C.-W. Lin, J. Frnda, Mining locally trending high utility itemsets, in: Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2020, pp. 99-111, **http://dx.doi.org/10.1007/ 978-3-030-47436-2_8**.

[29] C. Antunes, A.L. Oliveira, Generalization of pattern-growth methods for sequential pattern mining with gap constraints, in: Machine Learning and Data Mining in Pattern Recognition, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 239-251, **http://dx.doi.org/10.1007/3-540-45065-3_ 21**.

[30] M.J. Zaki, Sequence mining in categorical domains, in: Proceedings of the Ninth International Conference on Information and Knowledge Management, CIKM ’00, ACM Press, New York, 2000, pp. 422-429, **http://dx.doi. org/10.1145/354756.354849**.

[31] Y.-H. Ke, J.-W. Huang, W.-C. Lin, B.P. Jaysawal, Finding possible promoter binding sites in DNA sequences by sequential patterns mining with specific numbers of gaps, IEEE/ACM Trans. Comput. Biol. Bioinforma. (2020) **http: //dx.doi.org/10.1109/tcbb.2020.2980234**.

[32] V. Liao, M.-S. Chen, Efficient mining gapped sequential patterns for motifs in biological sequences, BMC Syst. Biol. 7 (2013) S7, **http://dx.doi.org/10. 1186/1752-0509-7-S4-S7**.

[33] M. D’Andreagiovanni, F. Baiardi, J. Lipilini, S. Ruggieri, F. Tonelli, Sequential pattern mining for ICT risk assessment and management, J. Log. Algebr. Methods Program. 102 (2019) 1-16, **http://dx.doi.org/10.1016/j.jlamp.2018. 09.007**.

[34] L.K.M. Poon, S.-C. Kong, M.Y.W. Wong, T.S.H. Yau, Mining sequential patterns of students’ access on learning management system, 2017, **http: //dx.doi.org/10.1007/978-3-319-61845-6_20**, Presented at the.

[35] F. Setiawan, B.N. Yahya, Improved behavior model based on sequential rule mining, Appl. Soft Comput. J. 68 (2018) 944-960, **http://dx.doi.org/10. 1016/j.asoc.2018.01.035**.

[36] T. Ledieu, G. Bouzillé, E. Polard, C. Plaisant, F. Thiessard, M. Cuggia, Clinical data analytics with time-related graphical user interfaces: Application to pharmacovigilance, Front. Pharmacol. 9 (2018) **http://dx.doi.org/10.3389/ fphar.2018.00717**.

[37] G. Srivastava, J.C.-W. Lin, X. Zhang, Y. Li, Large-scale high-utility sequential pattern analytics in internet of things, IEEE Internet Things J. 8 (2021) 12669-12678, **http://dx.doi.org/10.1109/JIOT.2020.3026826**.

[38] G. Srivastava, J.C.-W. Lin, A. Jolfaei, Y. Li, Y. Djenouri, Uncertain-driven analytics of sequence data in IoCV environments, IEEE Trans. Intell. Transp. Syst. 22 (2021) 5403-5414, **http://dx.doi.org/10.1109/TITS.2020.3012387**.

[39] P. Fournier-Viger, A. Gomariz, M. Šebek, M. Hlosta, VGEN: Fast vertical mining of sequential generator patterns, in: Proceedings of the International Conference on Data Warehousing and Knowledge Discovery, 2014, pp. 476-488, **http://dx.doi.org/10.1007/978-3-319-10160-6_42**.

[40] U. Yun, G. Lee, K.M. Lee, Efficient representative pattern mining based on weight and maximality conditions, Expert Syst. 33 (2016) 439-462, **http://dx.doi.org/10.1111/exsy.12158**.

[41] X. Ao, P. Luo, J. Wang, F. Zhuang, Q. He, Mining precise-positioning episode rules from event sequences, IEEE Trans. Knowl. Data Eng. 30 (2018) 530-543, **http://dx.doi.org/10.1109/TKDE.2017.2773493**.

[42] B. Zhang, J.C.W. Lin, P. Fournier-Viger, T. Li, Mining of high utility-probability sequential patterns from uncertain databases, PLoS One (2017) **http://dx.doi.org/10.1371/journal.pone.0180931**.

[43] J.C.-W. Lin, Y. Li, P. Fournier-Viger, Y. Djenouri, L.S.-L. Wang, Mining high-utility sequential patterns from big datasets, in: 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 2674-2680, **http://dx.doi.org/ 10.1109/BigData47090.2019.9005996**.

[44] W. Gan, J.C.W. Lin, J. Zhang, P. Fournier-Viger, H.C. Chao, P.S. Yu, Fast utility mining on sequence data, IEEE Trans. Cybern. 51 (2021) **http: //dx.doi.org/10.1109/TCYB.2020.2970176**.

[45] W. Gan, J.C.W. Lin, J. Zhang, H. Yin, P. Fournier-Viger, H.C. Chao, P.S. Yu, Utility mining across multi-dimensional sequences, ACM Trans. Knowl. Discov. Data. 15 (2021) **http://dx.doi.org/10.1145/3446938**.

[46] M. Patel, N. Modi, K. Passi, An effective approach for mining weighted sequential patterns, in: Proceedings of the International Conference on Smart Trends for Information Technology and Computer Communications, 2016, pp. 904-915, **http://dx.doi.org/10.1007/978-981-10-3433-6_108**.

[47] H.M. Huynh, L.T.T. Nguyen, B. Vo, A. Nguyen, V.S. Tseng, Efficient methods for mining weighted clickstream patterns, Expert Syst. Appl. 142 (2019) 112993, **http://dx.doi.org/10.1016/j.eswa.2019.112993**.

[48] H.M. Huynh, L.T.T. Nguyen, B. Vo, .Z.K. Oplatková, P. Fournier-Viger, U. Yun, An efficient parallel algorithm for mining weighted clickstream patterns, Inf. Sci. (Ny) 582 (2022) **http://dx.doi.org/10.1016/j.ins.2021.08.070**.

[49] P. Fournier-Viger, C.W. Wu, V.S. Tseng, L. Cao, R. Nkambou, Mining partially-ordered sequential rules common to multiple sequences, IEEE Trans. Knowl. Data Eng. 27 (2015) 2203-2216, **http://dx.doi.org/10.1109/ TKDE.2015.2405509**.

[50] P. Fournier-Viger, J.C.W. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, H.T. Lam, The SPMF open-source data mining library version 2, in: Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases, 2016, pp. 36-40, **http://dx.doi.org/10.1007/978-3-319-46131-1_8**.