IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 21 February 2023, accepted 26 March 2023, date of publication 29 March 2023, date of current version 3 April 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3262823

RESEARCH ARTICLE

An Effective Method for Mining Negative Sequential Patterns From Data Streams

NANNAN ZHANG, XIAOQIANG REN, AND XIANGJUN DONG
Department of Computer Science and Technology, Qilu University of Technology (Shandong Academy of Sciences), Jinan 250353, China
Corresponding author: Xiangjun Dong (d-xj@163.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 62076143, and in part by the Fundamental Research Promotion Plan of Qilu University of Technology (Shandong Academy of Sciences) under Grant 2021JC02009.

ABSTRACT Traditional negative sequential patterns(NSPs) mining algorithms are used to mine static dataset which are stored in equipment and can be scanned many times. Nowadays, with the development of technology, many applications produce a large amount of data at a very high speed, which is called as data stream. Unlike static data, data stream is transient and can usually be read only once. So, traditional NSP mining algorithm cannot be directly applied to data stream. Briefly, the key reasons are: (1) inefficient negative sequential candidates generation method, (2) one-time mining, (3) lack of real-time processing. To solve this problem, this paper proposed a new algorithm mining NSP from data stream, called nsp-DS. First, we present a method to generate positive and negative sequential candidates simultaneously, and a new negative containment definition. Second, we use a sliding window to store sample data in current time. The continuous mining of entire data stream is realized through the continuous replacement of old and new data. Finally, a prefix tree structure is introduced to store sequential patterns. Whenever the user requests, it traverses the prefix tree to output sequential patterns. The experimental results show that nsp-DS may discover NSPs from data streams.

INDEX TERMS Data stream, transient, sliding window, negative sequential patterns (NSPs).

I. INTRODUCTION

Sequential pattern mining aims to discover regular sequential patterns from a dataset of ordered events. Sequential pattern mining has been widely used in the field of group behavior analysis [1], [2], [3], optimization strategy [4], classification and clustering problem [5], [6], phenotypic structure learning [7], comparative behavior analysis [8], abnormal behavior detection [9], business intelligence [10], [11], education [10], recommendation system [12] and so on. In traditional sequential pattern mining research, sequence data are mostly in static form, but in reality, sequence data are mostly presented in a continuous dynamic ''stream''. Data stream [13], [14], [15] is a set of continuous sequence information, such as the user's network click stream, call data, sensor network data, scientific data, online retail transaction data, real-time stock transaction data, and network color data. Sequential pattern mining in data streams is an important research branch of sequential patterns mining.

However, the current research on sequential patterns mining in data streams only focuses on positive sequential patterns(PSPs), and no relevant research on NSPs mining has been found. Meanwhile, traditional NSP mining algorithms only target static datasets and cannot be applied directly to data streams. The data stream-oriented sequential pattern mining algorithm needs to take more into account the continuous, fast, real-time, massive, changing and orderly characteristics of the data stream. So, it is challenging to apply total algorithms in static datasets to data streams. The reasons are as follows:

1) *Inefficient Negative Sequential Candidates Method.* Current NSPs mining algorithms first mine PSPs, and then generate and obtain NSPs based on PSPs. This segmental method will produce a lot of intermediate results and affect the mining efficiency.

2) *One-time Mining.* Traditional NSPs mining is based on static datasets. All sequences are stored in memory

The associate editor coordinating the review of this manuscript and approving it for publication was Vlad Diaconita.

simultaneously. So, traditional algorithms process all data at one time. But, NSPs mining in data streams is based on continuous, rapid, and massive data. It is difficult to keep an entire data stream in memory, so it becomes impossible to process all data at one time.

3) *Lack of Real-time Processing.* Mining in data streams has higher requirements for speed. The real-time arrival and processing of the data stream are not available in traditional NSPs mining.

Based on the traditionl sequential pattern mining algorithms, the purpose of this paper is to propose a sequential pattern mining algorithm that can be adapted to data stream mining scenario. In this article, we propose an efficient algorithm, named nsp-DS, to mine NSPs from data streams. nsp-DS introduces ideas such as sliding windows, prefix trees, etc. The main idea is as follows.

First, we propose a method to generate positive and negative sequential candidates simultaneously, by improving the S-Step process in SPAM [16]. At the same time, we present a new negative containment definition suitable for mining PSPs and NSPs simultaneously. nsp-DS uses a bitmap to store data and uses bitwise operations to calculate support while generating candidates, which requires each item to be represented on the bitmap in a one-to-one correspondence.

Second, nsp-DS uses a sliding window to store sample data at the current time. When sliding window is first full, the algorithm mines the dataset in the window for the first time. When new data arrives, it replaces the oldest data by overwriting to update the sliding window. Then, we continue to mine the current window according to new data.

Third, we design a prefix tree structure to store sequential patterns. When the sliding window is first full, we create a prefix tree to store NSPs. With the arrival of new data, we mine the current window only by the new data and update the prefix tree using the mined sequential patterns. This operation can avoid repeated mining of old data in the sliding window and improve mining efficiency. Whenever the user requests, it traverses the prefix tree to output sequential patterns.

Finally, we propose the corresponding algorithm, called nsp-DS, for mining NSPs from data stream, and conducte some experiments in four real-world datasets. The experimental results show that nsp-DS can mine NSPs from data stream efficiently.

The rest of this paper is organized as follows. Section II discusses related work. In Section III, we introduce fundamental concepts about mining sequential patterns in data streams. Section IV implements the proposed method in detail. Section V describes the experiments and explains the experimental results. Section VI concludes this work.

II. RELATED WORK

Currently, we only find relevant studies on PSP mining in data streams, but not on NSP mining. The relevant research on NSP mining is focused on static datasets. So, in this section, we introduce works related to NSPs mining in static datasets and PSPs mining in data streams.

A. NSPS MINING IN STATIC DATASETS

NSPs refer to a frequent sequence that contains both occurring and unoccurring events. However, it is much more difficult to discover NSPs than PSPs due to the complexity of the problem caused by unoccurring behavior. But, it is obvious that mining NSPs will find more valuable information. According to different requirements, researchers have proposed corresponding definitions and algorithms [17], which are wildly used in medical education, behavior analysis,and other fields [7], [8], [9]. Hsueh. et al. proposed PNSP algorithm, which converts frequent positive elements to negative elements and then generates negative sequential candidates by concatenation [17]. Zheng et al. proposed NegGSP [18] algorithm based on GSP [19], which scans the dataset to get NSPs. Cao et al. proposed a very innovative and efficient theoretical framework: the set theory-based NSP mining (ST-NSP) and the corresponding algorithm e-NSP [20]. It identifies NSPs only by PSPs discovered without re-scanning the database, which makes e-NSP performs particularly well on datasets with a small number of elements in the sequence, a large number of itemsets, and low minimum support. However, when the dataset becomes dense, the key process of obtaining the support of negative sequence candidates in e-NSP becomes very time-consuming. To solve the problem, Dong et al. proposed f-NSP [11] algorithm, which uses a bitmap to store the information of PSP, and then obtains the support degree of negative candidate sequences only through bit operation, and is much faster than the hash method in e-NSP. To mine the expected number of NSP, Dong et al. proposed a TopK-NSP algorithm [12] to mine k common NSP. Gao et al. proposed sc-NSP [21]. sc-NSP improves PrefixSpan algorithm and increases negative candidate sequences by relaxing constraints conditions, which makes more interesting candidate sequences found.

B. PSPS MINING IN DATA STREAMS

The initial research work on data streams was first carried out by Alon et al. in 1996 [22]. Data stream model was proposed by Henzinger et al. in 1998. Later algorithms on data streams proliferate. These algorithms are generally based on traditional sequential pattern algorithms, combined with window models commonly used for data stream mining to extend and optimise the algorithms to obtain new algorithms. Chedi et al. proposed SPEED [23] which uses a novel treereg data structure to store useful information. On the basis of SPEED algorithm, Lei et al. proposed the Seq-Stream algorithm [24]. Shih-Yang Yang et al. designed an incremental mining algorithm IAspam [25] to mine sequential patterns in interaction streams. Meanwhile, Shih-Yang Yang et al. also proposed the ICspan [25] algorithm to mine closed sequences in data streams. Referring to the two algorithms proposed by Shih-Yang Yang, Guanling Lee et al. proposed PAlgorithm

and PSAlgorithm [26] to mine sequential patterns in data streams, and these two algorithms use the data structure Path-Tree to effectively integrate some of the mining results. CI Ezeife et al. proposed the SSM (Sequential Stream Mining) algorithm [27], which decomposes the data stream into blocks of variable size. Mendes, J.Han et al. proposed the SS-BE (Stream Sequence miner using Bounded Error) algorithm [28] and SS-MB (Stream Sequence miner using Memory Error) algorithm [28], both of which decompose the data stream into data blocks, and then mine sequential patterns in each data block. He Xingxing et al. proposed an efficient pruning-based sequential pattern mining algorithm SSPM(Stream Sequential Pattern Mining). Raissi and Poncelet [29] proposed a sampling-based algorithm for approximating global patterns. Tanbeer S K [30] et al. proposed an algorithm combining sliding window with CPS-tree for data stream sequential pattern mining. Shakeri O [31] et al. proposed a data stream sequential patterns mining algorithm with fuzzy constraints.

III. FUNDAMENTAL CONCEPTS

Data stream(DS) can be defined as continuously incoming sequences at a certain speed, $DS = \{S_1, S_2, S_3, \ldots, S_n, \ldots\}$. Let $I = \{i_1, i_2, i_3, \ldots, i_n\}$ represents a collection of items in DS. S_i is a sequence, which consists of some or all of items in I, $S_i = \{s_1, s_2, s_3, \ldots, s_k\}$ $(1 \le k \le n)$ where $s_j(1 \le j \le k)$ is an element and contains only one item in I, $\forall S_i, S_i \subseteq I$. The number of all elements in the sequence S is called the size of the sequence, denoted as $size\,(S)$. If the size of S is m, $size\,(S) = m$, it is the $m - size$ sequence. Assuming $S_1 = \langle d, c, f, c\rangle$, it is a 4-size sequence, and $size\,(S_1) = 4$.

If the sequence of $\alpha = \langle\alpha_1, \alpha_2, \ldots, \alpha_n\rangle$ for the sequence $\beta = \langle\beta_1, \beta_2, \ldots, \beta_n\rangle$ or β is a supersequence of α, it needs to satisfy the requirement of $1 \le j_1 \le j_2 \le \ldots \le j_n \le m$ and $\alpha_1 \subseteq \beta_{j_1}$, $\alpha_2 \subseteq \beta_{j_2}, \ldots$, $\alpha_n \subseteq \beta_{j_n}$, denoted as $\alpha \subseteq \beta$(β includes α). For example, a subsequence of $\langle a, b, c, e\rangle$ is $\langle ce\rangle$, and $\langle a, b, c, e\rangle$ is a supersequence of $\langle c, e\rangle$.

Sliding Window($SW = \{S_1, S_2, S_3, \ldots, S_w\}$), constitutes an instantaneous sample of DS. w, the width of SW, is the number of sequences contained in SW. w is set by the user. For all sequences in a given window SW of a data stream, the support of sequence s is defined as $sup(s) = f(s)/w$ where $f(s)$ is the number of sequences in which s occurs. min_sup is a user defined value to determine if sequence s is frequent or not. If $sup(s) \ge min$ sup, the sequence s is frequent.

Definition 1 (Positive Sequential Pattern, PSP): If the support degree of a positive sequence is bigger than or equal to min_sup, the positive sequence is a PSP.

Definition 2 (Negative Size): The number of negative elements in the sequence ns is negative size denoted as negsize(ns). If negative(ns)=n, ns is an n-negsize sequence. For example, ns=$\langle\neg a, b, \neg e, f\rangle$, ns is a 2-negsize sequence, denoted as negsize(ns)=2.

Definition 3 (Positive Matching): For elements, Positive Matching refers to transforming negative elements into positive elements. For example, Positive Matching of negative elements $\neg a$ is a, denoted as p($\neg a$)=a; For sequences, Positive Matching is the transformation of all negative elements in the sequence into their corresponding positive elements, denoted as $p\,(ns) = \{\langle s_1\prime, s_2\prime, \ldots, s_k\prime\rangle \,|s_i\prime = p(s_i), s_i \in ns\}$. For example, $p\,(\langle\neg a, b, \neg e, f\rangle) = \langle a, b, e, f\rangle$.

Definition 4 (Maximum Positive Subsequence, MPS): The maximum positive subsequence of a sequence is the subsequence that includes all positive elements in this sequence. S is a subsequence of $ns = \langle s_1, s_2, \ldots, s_m\rangle$. If S includes and only includes all positive elements in ns, then S is the maximum positive subsequence of ns. denoted as MPS(ns). For example MPS($\langle\neg a, b, \neg e, f\rangle$)=$\langle b, f\rangle$.

The definition of negative containment in e-NSP is $MPS\,(ns) \subseteq ds$; $\forall 1\text{-}negMS \in 1\text{-}negMSS_{ns}$, $p(1 - negMS) \not\subset ds$, that is, there is no need for a negative element in ns correspond with a certain element in ds in position. Assuming $ns = \langle a, \neg b, c\rangle$, $ds_1 = \langle a, c, e\rangle$, $ns \subseteq ds_1$ under the definition of e-NSP negative containment, that is, there is no need for an element corresponding to $\neg b$ between elements a and c in ds(the element can be one or more). However, nsp-DS uses a bitmap to store data, which requires each element/item (including negative element/item) to be represented on the bitmap in a one-to-one correspondence. For example, $ds_2 = \langle a, d, c, e\rangle$, element d between a and c can correspond to $\neg b$, then $ns \subseteq ds_2$. The definition of negative containment in e-NSP can not satisfy this requirement.

We try to extend the bitmap in SPAM [16] by adding a 0 between each item. Taking the sequence $\langle b, a, b\rangle$ as an example, the bitmap of item a is shown in Fig. 1. Among them, the left side of Table 1 represents the original bitmap of item a, and the right side represents the expanded bitmap. The bold font represents the 0 or 1 in the original bitmap, and the usual represents the added 0.

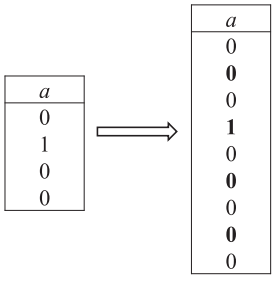


FIGURE 1. Bit extension.

However, this will make the bitmap sparse, which will reduce the efficiency of the algorithm. Therefore, in nsp-DS algorithm, we use a new definition of negative containment.

Definition 5 (Negative Containment): If the negative sequence $ns = \langle e_1, e_2, \ldots, e_k\rangle$ is contained in sequence $ds = \langle d_1, d_2, \ldots, d_m\rangle$$(m > k)$, for any negative element e_i, there are $p, q, r(p < q < r)$ that need to meet the following two conditions:

(1)MPS(ns) $\subseteq$ ds(2)$\exists e_{i-1} \subseteq d_p \bigwedge e_{i+1} \subseteq d_r$,and for $\forall d_q, e_i \not\subseteq d_q$.Especially, for $ns = \langle e_1, e_2\rangle$ that is 2-size, if e_1 is negative and e_2 is positive, $ds = \langle d_1, d_2, \ldots, d_m\rangle$ contains ns must need $e_2 \subseteq \langle d_2, \ldots, d_m\rangle \bigwedge e_2 \not\subseteq d_1$; Similarly, if e_1 is

positive and e_2 is negative, ds contains ns must need $e_1 \subseteq \langle d_1, d_2, \ldots, d_{m-1}\rangle \bigwedge e_1 \nsubseteq d_m$.

For example, sequence $ds = \langle b, a, c, d\rangle$:

1) For negative sequence $ns = \langle b, \neg b, d\rangle$, ns is contained in ds. That is because the element b does not appear between b and d in ds.
2) For negative sequence $ns = \langle b, \neg c, d\rangle$, ns is not contained in ds. That is because the element c appears between b and d.
3) For negative sequence $ns = \langle \neg a, b\rangle$, ns is not contained in ds. That is because there is no element before b in ds.
4))For negative sequence $ns = \langle d, \neg a\rangle$, ns is not contained in ds. That is because there is no element after d in ds.

NSP has no unified definition. Thus, if the form of NSPs is not constrained, the number of negative sequential candidates will explode. Most of these candidates are meaningless, which brings difficulties to the mining process. Different constraints are used in different NSPs mining algorithms. The negative sequential constraints adopted in this paper are as follows:

Constraint 1(Size Constraint): The maximum size of the negative sequence should not be greater than the data sequence size.

Constraint 2(Frequency Constraint): The positive element corresponding to each negative element in negative candidate sequences must be frequent.

Constraint 3(Format Constraint): No consecutive negative elements are allowed in a negative sequence.

Constraint 4(Negative Element Constraint): The smallest negative unit in a sequence is an element.

Definition 6 (Negative Sequential Pattern, NSP): If the support degree of a negative sequence is bigger than or equal to *min_sup*, the negative sequence is a NSP.

IV. NSP-DS ALGORITHM

In this section, we propose a novel and efficient algorithm, called nsp-DS, to obtain NSPs in a data stream. We introduce data structures used in nsp-DS in Section IV-A. The methods of generating candidate and support calculations are discussed in Section IV-B. The idea of nsp-DS algorithm is described by an example in Section IV-C.

A. DATA STRUCTURE

nsp-DS algorithm mainly includes three data structures, which are used to store timely datasets in sliding windows, sequence information, and NSP mining results.

1) BITMAP MATRIX

In SPAM algorithm, for the sequences whose size is between $2^k + 1$ and $2^{(k+1)}$, it needs to be complemented. The vacant position is filled with 0 in the bitmap, which makes the length of each bitmap equal. For example, Fig.2 is the bitmap of item a in the sequence $\langle b, a, b\rangle$. 0 in the fourth bit is a complement. We follow the method representing negative items in SPAM:

a	$\neg a$
0	1
1	0
0	1
0	**1**

FIGURE 2. The position of a and ¬a.

the occurrence position is represented by 1 and the absence of occurrence is represented by 0. Then, from Fig.2, we can see $\neg a$ appears in the first, third, and fourth elements of the sequence $\langle b, a, b\rangle$, but there are only three elements in the sequence $\langle b, a, b\rangle$, no fourth element. To sum up, according to the original filling method, an error will appear in the bitmap of the negative item.

Therefore, in nsp-DS, the complement operation is not performed. The length of the bitmap of each item is equal to the size of the current sequence. nsp-DS converts the dataset in a window into a bitmap matrix(BM), using 1 or 0 to represent if an item shows in a sequence. Specifically, nsp-DS creates a vertical bitmap for each item that appears in the window, and each bit in the bitmap corresponds to the position of each element in the window. If item i appears in a sequence, the bit corresponding to the position of item i in this sequence is set to 1; otherwise, the bit is set to 0. At the same time, we divide the bitmap, and the length of each part is the size of the corresponding sequence. Taking the data in Table 1 as an example, the vertical bitmap of each item is shown in Fig.3. Using this idea to negative single items, the position where a negative single item can appear is set to 1, and 0 otherwise. It can be found that the bitmap of the negative single item is complementary to the bitmap of the corresponding positive single item. Therefore, in nsp-DS, the bitmap of the negative single item is obtained by inverting the bitmap of the positive single item.

TABLE 1. Sequence dataset.

Sid	*Sequence*
1	$\langle b, b, b\rangle$
2	$\langle b, a, b\rangle$
3	$\langle b, d, b, c\rangle$
4	$\langle c\rangle$
5	$\langle b, d, a, c\rangle$

Sequences can also be represented by bitmaps according to the above idea. If the last element of a sequence is j, and all other elements or items of the sequence appear before j, then the bit corresponding to j will be set to 1; otherwise, it will be set to 0. Define $B(s)$ to represent a bitmap of the sequence s.

sid	*a*	*b*	*c*	*d*	$\neg a$	$\neg b$	$\neg c$	$\neg d$
1	0	1	0	0	1	0	1	1
	0	1	0	0	1	0	1	1
	0	1	0	0	1	0	1	1
2	0	1	0	0	1	0	1	1
	1	0	0	0	0	1	1	1
	0	1	0	0	1	0	1	1
3	0	1	0	0	1	0	1	1
	0	0	0	1	1	1	1	0
	0	1	0	0	1	0	1	1
	0	0	1	0	1	1	0	1
4	0	0	1	0	1	1	0	1
5	0	1	0	0	1	0	1	1
	0	0	0	1	1	1	1	0
	1	0	0	0	0	1	1	1
	0	0	1	0	1	1	0	1

FIGURE 3. Bitmap matrix.

2) TABLE OF SEQUENCE INFORMATION

The conversion method of the sequence s in SPAM is to avoid the situation that the subsequent element α appears before s and in s. Set to 1 is to ensure that statistics can be performed as long as the element α appears later. This method only considers whether the subsequent element α appears and does not consider the position of α. But a NSP needs to take into account the position of α and the position where s starts at the time of conversion. In nsp-DS, we use a new data structure to show sequence information, including a sequence bitmap, a negative element bitmap, the set of sequence start position pointers, and support. The negative element bitmap is the bitmap of the last negative element in the sequence. Hash table *sidHash* $\langle id, position\rangle$ is the set of sequence starting position pointers. Suppose e is the last positive element of the sequence s. *sid* is the sid number of the sequence containing e, and *position* is the first position where e shows in each bitmap. Taking the dataset in Table 1 as an example, the information of negative sequences $\langle b, \neg b\rangle$ and $\langle b, \neg b, c\rangle$ are shown in Fig. 4.

Sequence	Sequence Bitmap	Negative Element Bitmap	The Start Position Of Sequence				*sup*
<b¬b>	|0|0|0|0|1|0|0|1|0|1|0|0|1|1|1|	|0|0|0|0|1|0|0|1|0|1|1|0|1|1|1|	*sid*	2	3	5	1
			position	4	7	12	
<b¬bc>	|0|0|0|0|0|0|0|0|0|0|0|0|0|0|1|	|0|0|0|0|1|0|0|1|0|1|1|0|1|1|1|	*sid*	5			1
			position	12			

FIGURE 4. Sequence information.

3) SEQUENCE PATTERN TREE

A sequence pattern tree(SPT) is used to maintain sequential patterns in each sliding window. It is essentially a prefix tree. Attributes of nodes include item(element) name and *sup* of sequence which consists of items in the path from the root to the node. If node *Root* is layer 0, the path from *Root* to a node at layer k represents $k - size$ sequence. Due to NSPs' dissatisfaction with the downward closure property, there may be some infrequent negative sequences in SPT. We mark infrequent negative sequences which consists of items in the path from the root to a node by setting *sup* of the node to 0. The structure of SPT is shown in Fig.5.

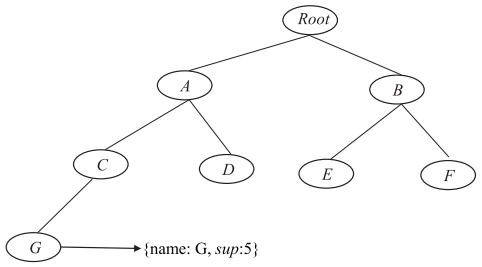


FIGURE 5. The structure of SPT.

B. CANDIDATE GENERATION AND SUPPORT CALCULATION METHOD

SPAM algorithm records the position of the first occurrence in each bitmap of the existing sequence s as k, stores the position of k in the table, and then converts it once (the bitmap before k and k is set to 0, and the element after k is set to 1), and finally does AND operation with the subsequent element α. If we use the same method as SPAM does, there will be some problems. Take the dataset in Table 1 as an example. the bitmap of $\langle b, \neg b\rangle$ is shown in Fig.6, where b_s represents the bitmap transformed by b.

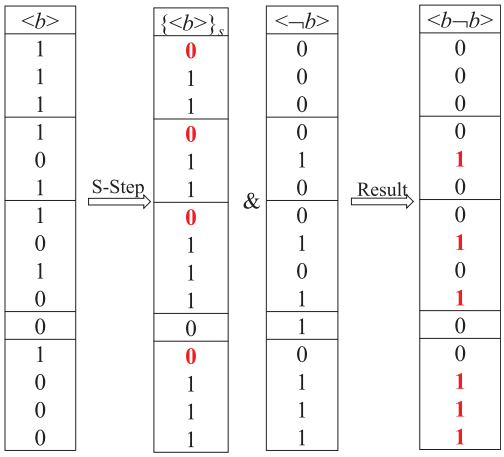


FIGURE 6. The bitmap of $\langle b\neg b\rangle$.

According to the support calculation method in SPAM, if a bitmap partition contains 1, add one to *sup*; otherwise, this partition has no contribution to *sup*. In Fig.6, the *sup* of $\langle b, \neg b\rangle$ is 3, that is, $\langle b, \neg b\rangle$ appears in the 2^{nd}, 3^{rd}, and 5^{th} sequences. We can find that in the 2^{nd} sequence $\langle b, a, b\rangle$ and 3^{rd} sequences $\langle b, d, b, c\rangle$, b appears after the first element b. Therefore, this method is not suitable for calculating the support of negative sequences in nsp-DS. Next, we will show how to complete the S-Step process with the structure mentioned above. The S-Step is to recursively expand the sequence of positive and negative elements, and the result is a logical sequence tree. In other words, S-Step is the process of creating a sequence tree by traversing. In S-Step, we extend sequence s by adding an extended element e after the last element of s. The S-Step includes three types: PP, PN, and NP. PP means that the last element of s and e are both positive,

PN is that the last element of s is positive and e is negative, and NP is that the last element of s is negative and e is positive.

Suppose there is a dictionary sequence in the dataset, if item i occurs before item j, we denote $i \leq j$. If s_a is a subsequence of s_b, this ordering can be extended to the sequence definition $s_a \leq s_b$. The root is marked as null in sequence tree T. Recursively down, if n is a node of the tree, then all children nodes of n are n', such that $n \leq n'$ and $\forall m \in T : n' \leq m \Rightarrow n \leq m$. Trees are infinite in this definition. Since NSP does not satisfy the property of downward closure, in practice, the sequence tree is also infinite. Therefore, we need to limit the size of the tree. According to the definition and negative constraint conditions, the size of PSPs and NSPs cannot exceed the size of the largest sequence in data streams. Assuming the size of the largest data sequence is k, for $\forall m \in T : n' \leq m \Rightarrow n \leq m, m \leq k$, the sequence tree is finite.

Fig.7 is the result of extending two elements (a and b). Assume that the maximum size of sequences after extending is 4. The top of the tree is $\emptyset$. $k - size$ sequence is in k layer. Sequences are arranged in lexicographic order in each layer.

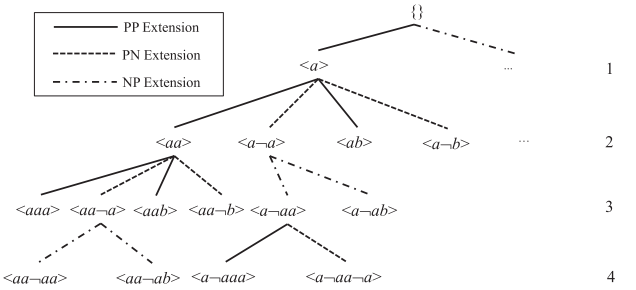


FIGURE 7. Sequence tree.

1) PP GENERATION

Assume that the bitmap of sequence s_a is $B(s_a)$, the bitmap of extended element e is $B(e)$, and the bitmap of the positive extended sequence s_b is $B(s_b)$. $B(s_b)$ has such an attribute. If the value is 1 in a position in $B(s_b)$, the corresponding element j in its data sequence must contain e, and all other elements in s_b must be contained in the elements before j. Suppose the first occurrence of 1 in $B(s_a)$ is at k. The element corresponding to position k should be the last element j of s_a, and all other elements of s_a must show before element j. When an extension element e is added to s_a, e must exist strictly after element j of s_b. Therefore, the value in position k in $B(s_b)$ is 0. What is more, if e is contained in any element after j, then the value in the corresponding position in $B(s_b)$ should be 1. Therefore, in $B(s_b)$, all 1 after k should correspond to $B(e)$.

When both the last element of s_a and e are positive, we first generate a bitmap according to $B(s_a)$. In this bitmap, values in all positions less than or equal to k are set to 0, and values in all positions after k are set to 1. This bitmap is called a conversion bitmap. The conversion bitmap does an ''AND'' operation with the bitmap of the extended element e to get a new bitmap. The bitmap obtained is the bitmap of the extended sequence s_b. The calculation method of sup of s_b is to judge whether there is a 1 in each partition of $B(s_b)$ and if there is a 1, sup is increased by one. If an element appears multiple times in a certain data sequence, that is, 1 appears more than once in a certain partition, sup of this partition is only recorded as 1.

Take the generation process of the extended sequence $\langle b, b\rangle$ as an example. First, find the position k where 1 first appears in each partition of $B(\langle b\rangle)$. Value in position k is set to 0, and values in positions after k are all 1. As shown in Fig.8, in the first partition of $B(b)$, the position where the first 1 appears is 1. So, the bit in position 1 is converted to 0, and bits in other positions are converted to 1. The converted bitmap $B(\{\langle b\rangle\}_s)$ is then ANDed with the bitmap of the extended element $\langle b\rangle$, and the result is $B(\langle b, b\rangle)$.

2) PN GENERATION

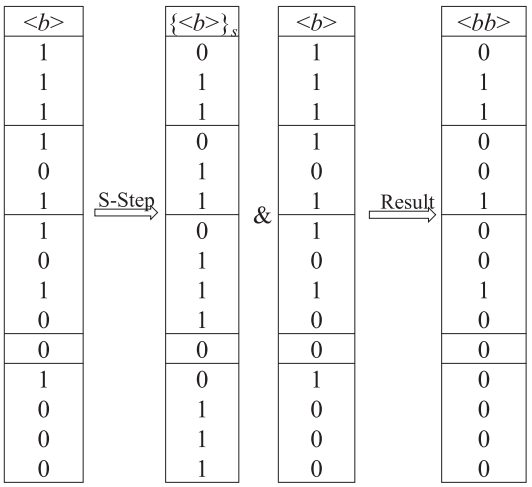
Assume that the last element of sequence s_a is positive, and the extended element e is negative. The generation method of the extended sequence s_b bitmap is consistent with that of PP. But the sup calculation method is different. Suppose that the position where 1 first appears in a partition of $B(s_a)$ is k, the element corresponding to position k should be the last element j of s_a, and j is positive. If 0 appears after k in the corresponding partition of $B(e)$, it means that the positive element corresponding to e appears after j. According to Definition 5, s_b is not included in this sequence.

Therefore, sup of s_b is to judge whether 0 appears in position after k in $B(e)$. If 0 does not appear, as long as 1 appears in the corresponding partition in $B(s_b)$, sup is increased by one; otherwise, sup is unchanged.

Take the example of generating the extended sequence $\langle b, \neg b\rangle$. As shown in Fig.9, the transpose position k of the sequence $\langle b\rangle$ is marked red, and the position where $B(\neg b)$ appears 0 in this partition is represented by the arrow. We can find that 1 appears in 2^{nd} and s^{nd} partition in $B(\langle b, \neg b\rangle)$. However, 1 in these two partitions does not influence sup because of the appearance of 0 in $B(\neg b)$ after k. sup of $\langle b, \neg b\rangle$ is only 1. At the same time, the sequence starting position in $sidHash\ \langle id, position\rangle$ of $\langle b, \neg b\rangle$ is modified, and the transpose position k of each partition containing 1 is stored.

3) NP GENERATION

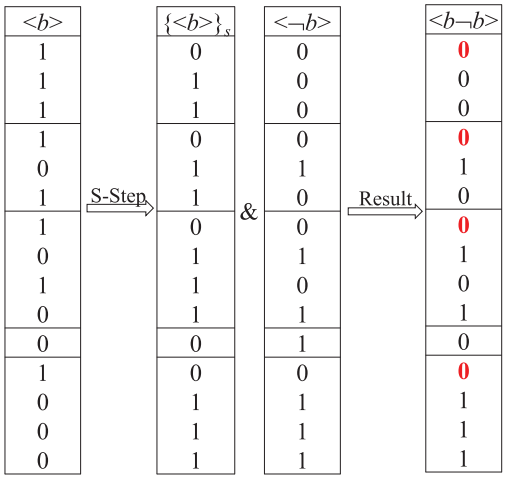
Assuming that the last element of sequence s_a is negative, and the extended element e is positive, the generation method of the extended sequence s_b bitmap is more complicated than the first two. First, AND the transposed bitmap of s_a with $B(e)$ to get a new bitmap. Secondly, we denote the starting position recorded in the table of s_a information as p and denote the position where 1 first appears in each partition in the new bitmap as j. Judge whether 0 appears between p and j in the negative element bitmap. If not appear, bit 1 in position j is reserved. If appear, the bit in position j is set to 0, and bit 1 in position after j in this partition is also set



Sequence	Sequence Bitmap	Negative Element Bitmap	The Start Position Of Sequence				Sup
<b>	B(<b>)	null	*sid*	1	2	3	5	4
			position	1	4	7	12	

Sequence	Sequence Bitmap	Negative Element Bitmap	The Start Position Of Sequence				sup
<bb>	B(<bb>)	null	*sid*	1	2	3	3
			position	2	6	9	

FIGURE 8. PP generation.



Sequence	Sequence Bitmap	Negative Element Bitmap	The Start Position Of Sequence				sup
<b>	B(<b>)	null	*sid*	1	2	3	5	4
			position	1	4	7	12	

Sequence	Sequence Bitmap	Negative Element Bitmap	The Start Position Of Sequence	sup
<¬b>	B(<¬b>)	B(<¬b>)	null	1

Sequence	Sequence Bitmap	Negative Element Bitmap	The Start Position Of Sequence				sup
<b¬b>	B(<b¬b>)	B(<¬b>)	*Sid*	2	3	5	1
			Position	4	7	12	

FIGURE 9. PN generation.

to 0. At the same time, if all 1 are reserved, *sup* is increased by one. This operation is to ensure that a negative element that occurs between two positive elements of the extended sequence s_b does not appear at the corresponding position in the data sequence.

If we do not make the above operation, subsequent sequence extensions may have some misjudgment. For a sequence, $ds = \langle a, b, d, a, b, d, a, c, d\rangle$, $B(\langle a, \neg b, d\rangle)$ is shown as Fig.10. $\langle a, \neg b, d\rangle$ is the extension sequence of sequence $\langle a, \neg b\rangle$. According to $B(\langle a, \neg b, d\rangle)$, we can infer that element d may appear in 6^{th} and 9^{th} position. But, in ds, there is an element b between a in 1^{st} position and d in 6^{th} position. In the same way, this b is also between a in 1^{st} position and d in 9^{th} position. According to Definition 5, $\langle a, \neg b, d\rangle$ is not contained in ds. If we extend $\langle a, \neg b, d\rangle$ to $\langle a, \neg b, d, \neg b\rangle$, it is obviously wrong that $\langle a, \neg b, d, \neg b\rangle$ is contained in ds according to the *sup* calculation method used in PN Generation.

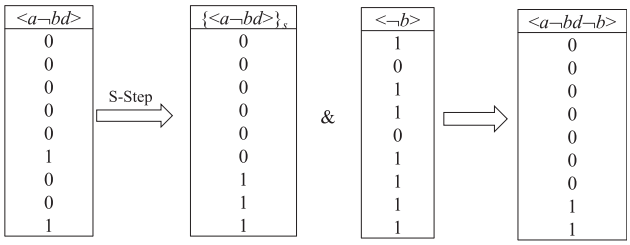


FIGURE 10. The error example of NP support calculation.

Take the generation of the extended bitmap $B(\langle b, \neg b, c\rangle)$ as an example, as shown in Fig.11. The red indicates the starting position k recorded in the table of the sequence $\langle b\neg b\rangle$ information. j is the position where bit 1 first shows in each partition of $B(\langle b, \neg b, c\rangle)$. The arrow points to the position where bit 0 occurs between k and j. Because bit 0 appears on position 3 in 3^{rd} partition of $B(\langle \neg b\rangle)$, all bit 1 after position 3 in 3^{rd} partition of are set to 0.

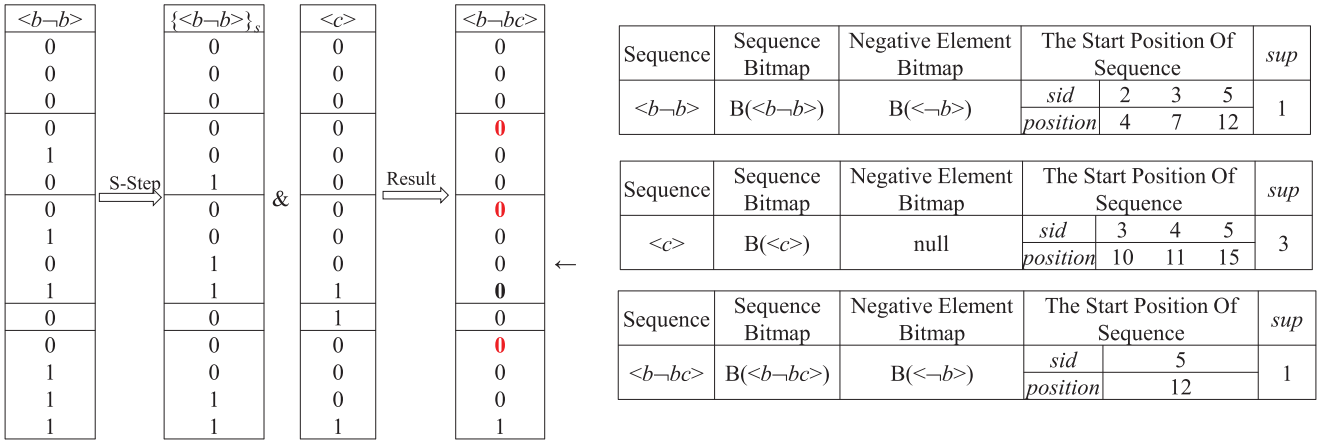


FIGURE 11. NP generation.

4) PRUNING STRATEGY

nsp-DS algorithm generates $k-size$ candidate sequences by extending $(k-1)-size$ sequences, so the number of candidate sequences will be very large. It makes it difficult to search for meaningful sequential patterns. In order to improve the performance of the algorithm, we need to remove some candidates during S-Step. This is called Pruning Strategies which are divided into the pruning strategy of positive candidate sequence and the pruning strategy of negative candidate sequence.

The pruning strategy of positive candidate sequences is removing infrequent positive sequences generated during extension according to the downward closure property. For example, the sequence $\langle c, d\rangle$ extends to $\langle c, d, e\rangle$ during S-Step. If $\langle c, d, e\rangle$ is infrequent, its extension sequences are also infrequent. Therefore, delete $\langle c, d, e\rangle$ directly and no longer extend it. The pruning strategy of negative candidate sequences is that two adjacent negative elements are not allowed in a NSP according to Constraint 3. For example, for the negative sequence $ns = \langle e_1, e_2 \ldots e_n\rangle$, when e_n is a negative element, and the following element α is a negative element, we prune it and do not perform the following steps.

The algorithm is explained in detail as follows:

1) Traverse each element e in Sn, and combine the sequence s with e to generate a new extended sequence s', as shown in Line 3.
2) If element e is negative and the last element of the sequence s is also negative, stop the current loop (Lines 3-5).
3) If the extended sequence s' is positive and frequent, s' is a PSP, and the element e is stored in S_{temp} (Lines 8-11).
4) If the extended sequence s' is negative, judge whether s' and s are frequent. If neither is frequent, stop the current loop, otherwise store element e in S_{temp} (Lines 13-17).
5) Merge elements in S_{temp} with sequence s to generate a new sequence and perform the next recursion with the new sequence and S_{temp}.

Algorithm 1 DFS of Sequence Tree(sequence s,Sn)

Input: sequence s,Sn;
Output: $k-size$ PSPs and NSPs(k>1);

```
1:  S_temp = ∅;
2:  for each element e in Sn do
3:      sequence s' ← s = ⟨e1, e2, ..., en⟩ ∪ e
4:      if e is negative and the last element of s is negative
        then
5:          Continue;
6:      end if
7:      if s' is positive sequence then
8:          if s' is frequent then
9:              s' is stored in PSPs;
10:             S_temp = temp ∪ e;
11:         end if
12:     else
13:         if s and s' are infrequent then
14:             Continue;
15:         else
16:             S_temp = S_temp ∪ e;
17:         end if
18:     end if
19: end for
20: for each element e in S_temp do
21:     sequence s' ← s = ⟨e1, e2, ..., en⟩ ∪ e
22:     DFS of Sequence Tree(s',Sn);
23: end for
```

C. IDEA OF NSP-DS ALGORITHM

When the window is full(the number of sequences in a window is equal w) for the first time, mine PSPs and NSPs, and store these patterns to SPT. After the window slides, the new sequence S_{new} replaces the oldest sequence S_{old}. SPT is updated according to elements in S_{old}. And then, sequential pattern mining is performed for S_{new}. Finally, update the prefix tree again. When the user makes a request, scan SPT to get PSPs and NSPs in the current period.

Take the data stream shown in Table 2 as an example to introduce the nsp-DS algorithm, and set *min_sup*=0.5 and *w*=6.

TABLE 2. Data stream.

Sid	Sequence
1	⟨a, c, d, f, g⟩
2	⟨a, b, d, e, g⟩
3	⟨a, d, f, g⟩
4	⟨b, d, f⟩
5	⟨e, f, g⟩
6	⟨a, b, c, d, g⟩
7	⟨a, b, e, g⟩
...	...

1) WHEN WINDOW IS FIRST FULL

At first, the window W is empty($w = 0$). When the number of sequences in W is equal to w, W is full. At this time, we convert the dataset in W into a bitmap matrix and mine PSPs and NSPs in the dataset of W.

Take the data stream in Table 2 as an example. When W is filled for the first time, there are six sequences in W. So, the dataset in W will be converted into a bitmap matrix(BM), as shown in Fig.12.

sid	*a*	*b*	*c*	*d*	*e*	*f*	*g*	¬*a*	¬*b*	¬*c*	¬*d*	¬*e*	¬*f*	¬*g*
1	1	0	0	0	0	0	0	0	1	1	1	1	1	1
	0	0	1	0	0	0	0	1	1	0	1	1	1	1
	0	0	0	1	0	0	0	1	1	1	0	1	1	1
	0	0	0	0	0	1	0	1	1	1	1	1	0	1
	0	0	0	0	0	0	1	1	1	1	1	1	1	0
2	1	0	0	0	0	0	0	0	1	1	1	1	1	1
	0	1	0	0	0	0	0	1	0	1	1	1	1	1
	0	0	0	1	0	0	0	1	1	1	0	1	1	1
	0	0	0	0	1	0	0	1	1	1	1	0	1	1
	0	0	0	0	0	0	1	1	1	1	1	1	1	0
3	1	0	0	0	0	0	0	0	1	1	1	1	1	1
	0	0	0	1	0	0	0	1	1	1	0	1	1	1
	0	0	0	0	0	1	0	1	1	1	1	1	0	1
	0	0	0	0	0	0	1	1	1	1	1	1	1	0
4	0	1	0	0	0	0	0	1	0	1	1	1	1	1
	0	0	0	1	0	0	0	1	1	1	0	1	1	1
	0	0	0	0	0	1	0	1	1	1	1	1	0	1
5	0	0	0	0	1	0	0	1	1	1	1	0	1	1
	0	0	0	0	0	1	0	1	1	1	1	1	0	1
	0	0	0	0	0	0	1	1	1	1	1	1	1	0
6	1	0	0	0	0	0	0	0	1	1	1	1	1	1
	0	1	0	0	0	0	0	1	0	1	1	1	1	1
	0	0	1	0	0	0	0	1	1	0	1	1	1	1
	0	0	0	1	0	0	0	1	1	1	0	1	1	1
	0	0	0	0	0	0	1	1	1	1	1	1	1	0

FIGURE 12. BM.

Then, mine dataset in W to get PSPs and NSPs. The sequential patterns $\{\langle a\rangle, \langle a, d\rangle, \langle a, d, \neg b\rangle, \langle a, d, \neg b, g\rangle, \langle a, d, g\rangle, \langle a, g\rangle, \langle b\rangle, \langle b, d\rangle, \langle b, d, \neg b\rangle, \langle d\rangle, \langle d, \neg b\rangle, \langle d, \neg b, g\rangle, \langle d, f\rangle, \langle d, g\rangle, \langle \neg b\rangle, \langle \neg b, f\rangle, \langle \neg b, f, g\rangle, \langle \neg b, g\rangle, \langle f\rangle, \langle f, \neg b\rangle, \langle f, g\rangle, \langle g\rangle\}$ is obtained. These patterns are stored in SPT, as shown in Fig.13.

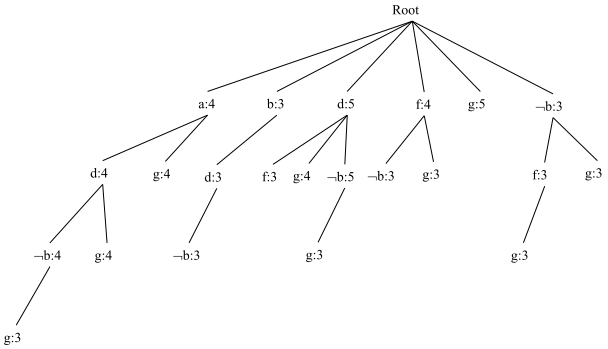
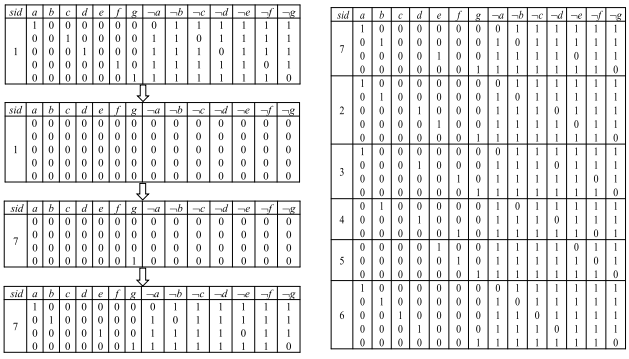


FIGURE 13. Sequence pattern tree.

2) WHEN WINDOW STARTS SLIDING

When W is filled, the newly arrived sequence overrides the oldest sequence in the window, completing a window sliding. $Sid_{old} = Sid_{new} \cdots\cdots w$. Sid_{old} is the *Sid* of replaced sequence, and Sid_{new} is the *Sid* of the newly arrived sequence.

Step 1: updating BM. The new sequence $Sid = 7$ replaces the old sequence $Sid = 1$. Set the bit of each column of the row *Sid*=1 in BM to 0, and re-assign the bit to the row with the sequence *Sid*=7. The process and result are as shown in Fig.14.



(a) Process (b) Result

FIGURE 14. Update BM.

Update *sup* of the corresponding node of SPT. $1-size$ patterns associated with $Sid = 1$ are $\{\langle a\rangle, \langle b\rangle, \langle d\rangle, \langle f\rangle, \langle g\rangle, \langle \neg b\rangle\}$. So, *sup* of nodes $\langle a, b, d, f, g, \neg b\rangle$ in SPT are decremented by 1. This is shown in Fig.15.

Step 2: mining sequential patterns for S_{new}. $S_{new} = S_{Sid=7} = \langle a, b, e, g\rangle$. In this step, sequential pattern mining is conducted only for the elements $\{a, b, e, g, \neg b, \neg c, \neg d, \neg e, \neg f\}$. These negative elements are the ones that change during the alternation of the old and new sequences. Because the number of items contained in each sequence is far less than the total number of items in the whole data stream, the efficiency of nsp-DS is improved.

Scan the BM after updating, and count the number of 1 in columns $a, b, e, g, \neg b, \neg c, \neg d, \neg e$, and $\neg f$. $\{\langle a\rangle, \langle b\rangle, \langle e\rangle, \langle g\rangle, \langle \neg e\rangle, \langle \neg f\rangle\}$ are $1-size$ sequential

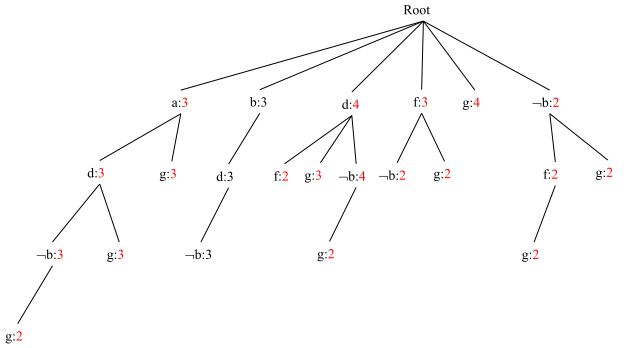


FIGURE 15. SPT after removing *Sid*=1.

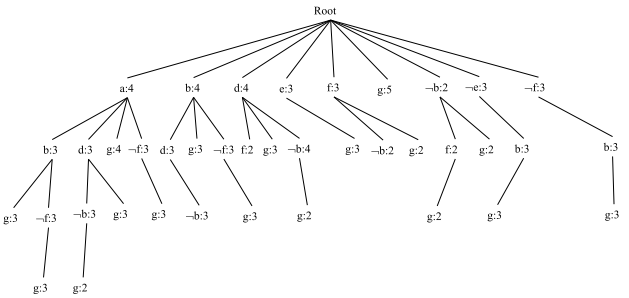


FIGURE 16. SPT after adding *Sid*=7.

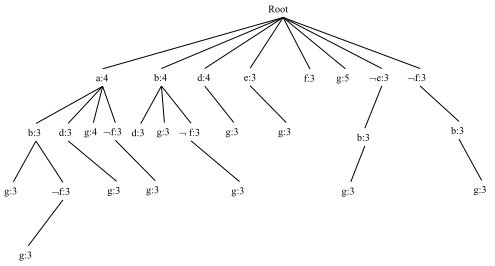


FIGURE 17. SPT after pruning.

patterns. These $1-size$ sequential pattern execute $S-Step$ to get $k-size\,(k>1)$ sequential patterns $\{\langle a,b\rangle, \langle a,b,g\rangle, \langle a,b,\neg f\rangle, \langle a,b,\neg f,g\rangle, \langle a,g\rangle, \langle a,\neg f\rangle, \langle a,\neg f,g\rangle, \langle b,g\rangle, \langle b,\neg f\rangle, \langle b,\neg f,g\rangle, \langle e,g\rangle, \langle \neg e,b\rangle, \langle \neg e,b,g\rangle, \langle \neg f,b\rangle, \langle \neg f,b,g\rangle\}$

Update SPT with the above sequential patterns. For existing sequential patterns in SPT, update the *sup* of corresponding nodes; for newly generated sequential patterns, add them to corresponding positions in SPT. At this time, we get a new SPT, as shown in Fig.16.

Step 3: responding to the request of the user. When the user requests, scan SPT to get sequential patterns in the current window. Traverse SPT. If *sup* of for all nodes on the path from a node to its leaf node are less than *min_sup*, delete those nodes from SPT. After pruning, SPT is shown as Fig.16.

Get sequential patterns $\{\langle a\rangle, \langle a,b\rangle, \langle a,b,g\rangle, \langle a,b,\neg f\rangle, \langle a,b,\neg f,g\rangle, \langle a,d\rangle, \langle a,d,g\rangle, \langle a,g\rangle, \langle a,\neg f\rangle, \langle a,\neg f,g\rangle, \langle b\rangle, \langle b,d\rangle, \langle b,g\rangle, \langle b,\neg f\rangle, \langle b,\neg f,g\rangle, \langle d\rangle, \langle d,g\rangle, \langle e\rangle, \langle e,g\rangle, \langle f\rangle, \langle g\rangle, \langle \neg e\rangle, \langle \neg e,b\rangle, \langle \neg e,b,g\rangle, \langle \neg f\rangle, \langle \neg f,b\rangle, \langle \neg f,b,g\rangle\}$.

The pseudocode of nsp-DS algorithm is as follows:

Algorithm 2 Nsp-DS Algorithm

Input: Data Stream *DS*, *min_sup*, Window Size *w*;
Output: Sequence Pattern Tree(SPT);

```
if w is first full then
    Scan w for all 1 − size PSPs
    1 − size NSPs is generated by 1 − size PSPs;
    insert 1 − size PSPs and NSPs to SPT;
    for each sequence s in 1 − size PSPs and NSPs do
        1 − size PSPs and NSPs←DFS of sequence Tree(e,1 − size PSPs and NSPs);
        insert k − size PSPs and NSPs to SPT;
    end for
else
    while sequence s is not the last sequence of DS do
        get 1 − size PSPs and NSPs associated with the oldest sequence in the current window;
        update SPT(the frequency of all nodes containing 1 − size PSPs and NSPs in SPT are reduced 1);
        replace the oldest sequence with sequence s;
        get 1−size PSPs and NSPs associated with sequence s;
        update SPT(1 − size PSPs and NSPs associated with sequence s);
        for each sequence s in 1 − size PSPs and NSPs do
            k − size PSPs and NSPs←DFS of sequence Tree(e,1 − size PSPs and NSPs);
            update SPT(k − size PSPs and NSPs);
        end for
    end while
end if
return SPT;
```

V. EXPERIMENTS ANALYSIS

We conduct experiments on four real-world datasets downloaded from https://www.philippe-fournier-viger.com/spmf/. Table 3 shows the features of four datasets. nsp-DS are implemented in Eclipse, running on a Windows 11 PC with 16 GB memory, and an Intel Core i5 2.4 GHz CPU. All the programs are written in Java.

Unfortunately, we find no algorithm suitable for a comparison algorithm. The reasons are as follows:

TABLE 3. Features of four datasets.

Dataset Name	Sequence Count	Item Count
BMSWebView1	59061	497
BMSWebView2	77512	3340
FIFA	20450	2990
MSNBC	31790	17

1) A traditional sequential patterns mining algorithm is not suitable for a comparison algorithm. The dataset used in traditional sequential pattern mining algorithms is completely different in nature from the dataset used in nsp-DS. In traditional sequential pattern mining algorithms, all data are stored in memory simultaneously. But data streams are generated continuously over time. In other words, datasets in traditional algorithms are static, but in nsp-DS they are dynamic. If comparing nsp-DS with a traditional sequential patterns mining algorithm, we cannot find a dataset that is suitable for both algorithms. Thus, nsp-DS cannot be compared with a traditional sequential patterns mining algorithm.
2) A PSPs mining algorithm in data streams is not suitable for a comparison algorithm. In order to improve the efficiency of the algorithm and to conform to the transient characteristics of the data stream, nsp-DS generates positive and negative candidate sequences simultaneously. If compared with PSPs mining algorithms in data streams, the new method of producing candidates in nsp-DS will become meaningless. So, no PSPs mining algorithm in data streams can be a comparison algorithm.

A. NUMBER OF NSPs AND PSPs GENERATED BY NSP-DS

In this section, we analyze the number of NSPs and PSPs generated by nsp-DS with various *min_sup*. From Fig.18, we can find that the number of NSPs and PSPs decreases as *min_sup* increases when *w* is fixed. For the same window size, as *min_sup* gets larger, the number of candidates that can meet the support gets smaller so fewer NSPs and PSPs are produced.

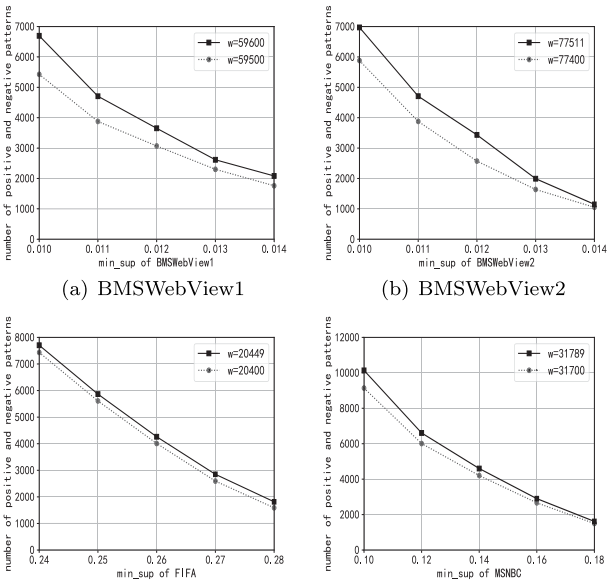


FIGURE 18. Number of NSPs and PSPs in different *min_sup* when *w* is fixed on four data streams.

B. RUNTIMES ON NSP-DS

In this section, we analyze runtimes on nsp-DS. The results are shown in Fig.19. It is clear that the runtime increases as the support decreases. When *min_sup* is decreased, a large number of NSPs are discovered so that the runtime is increased.

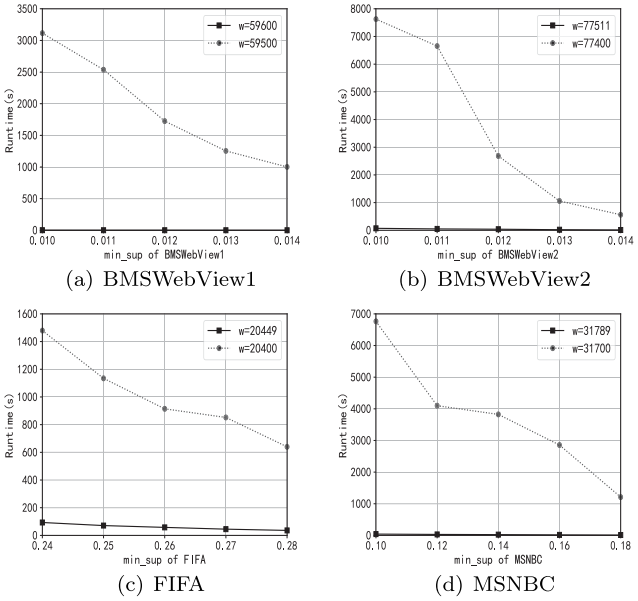


FIGURE 19. Runtime in different *min_sup* when *w* is fixed on four data streams.

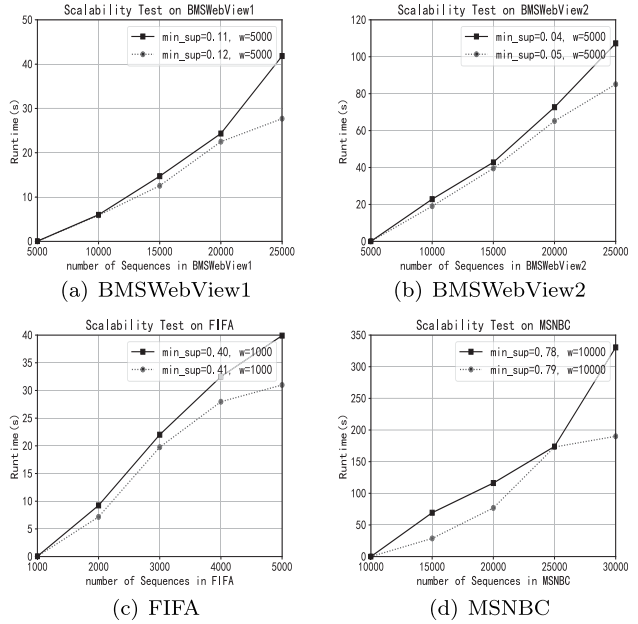


FIGURE 20. Scalability test of nsp-DS on four data streams.

C. SCALABILITY TEST ON NSP-DS

In this section, we perform scalability experiments on each four datasets with increasing number of sequences. Specially, the selected window sizes are 5,000 in BMSWebView1, 5,000

in BMSWebView2, 1,000 in FIFA, and 10,000 in MSNBC, with various low minimum supports *min_sup* 0.11 and 0.12 on BMSWebView1, 0.04 and 0.05 on BMSWebView2, 0.40 and 0.41 on FIFA, and 0.78 and 0.79 in MSNBC, respectively. Fig.20 shows the results of nsp-DS on each dataset in terms of different number of sequences: from 5,000 to 25,000 sequences of BMSWebView1, from 5,000 to 25,000 sequences of BMSWebView2, from 1,000 to 5,000 sequences of FIFA, from 1,0000 to 3,0000 sequences of MSNBC. Abscissa represents number of sequences, the ordinate represents the running time of the algorithm, and each point represents the running time of the algorithm in different support degrees when window size is fixed under the current dataset.

The experimental results show that the runtime has a roughly linear relationship with number of sequences increase under various minimum supports and window size. Therefore, nsp-DS has good scalability in terms of runtime with respect to number of sequences.

VI. CONCLUSION AND FUTURE WORK

Because of the characteristics of data streams, traditional NSPs mining algorithms cannot be directly applied to data streams. At the same time, the existing algorithms in the data stream only mine PSPs, and no algorithm mining NSPs is found. Mining NSPs in data streams is a challenging task. In this paper, we propose nsp-DS algorithm, which introduces a sliding window and a prefix tree structure to mine NSPs in data streams. Firstly, different from the method of two-step negative candidate generation, nsp-DS generates positive and negative candidates simultaneously. In particularly, it uses bit operations to generate negative candidates and calculate *sup* of negative candidates. Secondly, a prefix tree structure that can store NSPs is designed. During data replacement, we only need to update the prefix tree, not create another tree. Finally, experiments show that nsp-DS algorithm can mine NSPs in the data stream.

Although nsp-DS can be applied to mining NSPs in data streams, it is only applicable to sequences whose element is $1-size$ in data streams. Work still remains to accurately mine sequences with $k-size$ element$(k > 1)$ in data streams. Then, since NSPs do not have the downward inclusion property, there may be infrequent nodes in the prefix tree. We need to find a better structure to store NSPs.

ACKNOWLEDGMENT

(Nannan Zhang and Xiaoqiang Ren are co-first authors.)

REFERENCES

[1] L. Cao, Y. Ou, and P. S. Yu, "Coupled behavior analysis with applications," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 8, pp. 1378–1392, Aug. 2012.

[2] N. V. Mudrick, R. Azevedo, and M. Taub, "Integrating metacognitive judgments and eye movements using sequential pattern mining to understand processes underlying multimedia learning," *Comput. Hum. Behav.*, vol. 96, pp. 223–234, Jul. 2019.

[3] C. H. Yun and M. S. Chen, "Mining mobile sequential patterns in a mobile commerce environment," *IEEE Trans. Syst., Man, Cybern., C* , vol. 37, no. 2, pp. 278–295, Mar. 2007.

[4] C. Bi, "Comparison of optimization techniques for sequence pattern discovery by maximum-likelihood," *Pattern Recognit. Lett.*, vol. 31, no. 14, pp. 2147–2160, Oct. 2010.

[5] D. Fradkin and F. Mörchen, "Mining sequential patterns for classification," *Knowl. Inf. Syst.*, vol. 45, no. 3, pp. 731–749, Dec. 2015.

[6] C. Yang and J. Zhou, "Non-stationary data sequence classification using online class priors estimation," *Pattern Recognit.*, vol. 41, no. 8, pp. 2656–2664, Aug. 2008.

[7] Y. Zhao, G. Wang, X. Zhang, J. X. Yu, and Z. Wang, "Learning phenotype structure using sequence model," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 3, pp. 667–681, Mar. 2014.

[8] Z. Zheng, W. Wei, C. Liu, W. Cao, L. Cao, and M. Bhatia, "An effective contrast sequential pattern mining approach to taxpayer behavior analysis," *World Wide Web*, vol. 19, no. 4, pp. 633–651, Jul. 2016.

[9] Y. Song, L. Cao, X. Wu, G. Wei, W. Ye, and W. Ding, "Coupled behavior analysis for capturing coupling relationships in group-based market manipulations," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2012, pp. 976–984.

[10] A. Aztiria, J. C. Augusto, R. Basagoiti, A. Izaguirre, and D. J. Cook, "Learning frequent behaviors of the users in intelligent environments," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 43, no. 6, pp. 1265–1278, Nov. 2013.

[11] O. J. Räsänen and J. P. Saarinen, "Sequence prediction with sparse distributed hyperdimensional coding applied to the analysis of mobile phone use patterns," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 9, pp. 1878–1889, Sep. 2016.

[12] J. K. Tarus, Z. Niu, and D. Kalui, "A hybrid recommender system for *e*-learning based on context awareness and sequential pattern mining," *Soft Comput.*, vol. 22, no. 8, pp. 2449–2461, Apr. 2018.

[13] X. Amatriain, "Data mining methods for recommender systems," in *Recommender Systems Handbook*. Berlin, Germany: Springer, 2011, pp. 39–71.

[14] A. Cuzzocrea, F. Jiang, W. Lee, and C. K. Leung, "Efficient frequent itemset mining from dense data streams," in *Proc. Asia–Pacific Web Conf.* Cham, Switzerland: Springer, 2014, pp. 593–601.

[15] Y. Qin, Q. Z. Sheng, N. J. G. Falkner, S. Dustdar, H. Wang, and A. V. Vasilakos, "When things matter: A survey on data-centric Internet of Things," *J. Netw. Comput. Appl.*, vol. 64, pp. 137–153, Apr. 2016.

[16] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu, "Sequential pattern mining using a bitmap representation," in *Proc. 8th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2002, pp. 429–435.

[17] S.-C. Hsueh, M.-Y. Lin, and C.-L. Chen, "Mining negative sequential patterns for e-commerce recommendations," in *Proc. IEEE Asia–Pacific Services Comput. Conf.*, Dec. 2008, pp. 1213–1218.

[18] Z. Zheng, Y. Zhao, Z. Zuo, and L. Cao, "Negative-GSP: An efficient method for mining negative sequential patterns," in *Proc. 8th Australas. Data Mining Conf.*, 2009, pp. 1–5.

[19] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," in *Proc. Int. Conf. Extending Database Technol.* Cham, Switzerland: Springer, 1996, pp. 1–17.

[20] L. Cao, X. Dong, and Z. Zheng, "e-NSP: Efficient negative sequential pattern mining," *Artif. Intell.*, vol. 235, pp. 156–182, Jun. 2016.

[21] X. Gao, Y. Gong, T. Xu, J. Lu, Y. Zhao, and X. Dong, "Toward better structure and constraint to mine negative sequential patterns," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 2, pp. 571–585, Feb. 2023.

[22] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *Proc. 28th Annu. ACM Symp. Theory Comput.*, 1996, pp. 20–29.

[23] C. Raissi, P. Poncelet, and M. Teisseire, "Need for speed: Mining sequential patterns in data streams," *BDA, Actes des 21Iemes Journees Bases de Donnees Avancees*, Oct. 2005.

[24] H. Pinto, J. Han, J. Pei, K. Wang, Q. Chen, and U. Dayal, "Multi-dimensional sequential pattern mining," in *Proc. 10th Int. Conf. Inf. Knowl. Manag.*, Oct. 2001, pp. 81–88.

[25] S.-Y. Yang, C.-M. Chao, P.-Z. Chen, and C.-H. Sun, "Incremental mining of across-streams sequential patterns in multiple data streams," *J. Comput.*, vol. 6, no. 3, pp. 449–457, Mar. 2011.

[26] G. Lee, K.-C. Hung, and Y.-C. Chen, "Path tree: Mining sequential patterns efficiently in data streams environments," in *Advances in Intelligent Systems and Applications—Volume 1* (Smart Innovation, Systems and Technologies). Berlin, Germany: Springer, 2013, pp. 261–268.

[27] C. I. Ezeife and M. Monwar, "SSM: A frequent sequential data stream patterns miner," in *Proc. IEEE Symp. Comput. Intell. Data Mining*, Mar. 2007, pp. 120–126.
[28] L. F. Mendes, B. Ding, and J. Han, "Stream sequential pattern mining with precise error bounds," in *Proc. 8th IEEE Int. Conf. Data Mining*, Dec. 2008, pp. 941–946.
[29] C. Raissi and P. Poncelet, "Random sampling over data streams for sequential pattern mining," *La Revue MODULAD*, vol. 36, pp. 61–66, May 2007.
[30] S. K. Tanbeer, C. F. Ahmed, B.-S. Jeong, and Y.-K. Lee, "Sliding window-based frequent pattern mining over data streams," *Inf. Sci.*, vol. 179, no. 22, pp. 3843–3865, Nov. 2009.
[31] O. Shakeri, M. M. Pedram, and M. Kelarestaghi, "A fuzzy constrained stream sequential pattern mining algorithm," in *Proc. 7th Int. Symp. Telecommun. (IST)*, Sep. 2014, pp. 20–24.

NANNAN ZHANG received the bachelor's degree in software engineering. She is currently pursuing the master's degree in computer technology with the Qilu University of Technology (Shandong Academy of Sciences). Her research interests include sequential pattern mining, negative sequential pattern mining, and association rules.

XIAOQIANG REN received the bachelor's degree from the School of Computer Science, Shandong Institute of Light Industry, in 2000, and the master's degree from the School of Computer Science and Engineering, Shandong University of Science and Technology, in 2008. He is an Assistant Professor with the Qilu University of Technology. His main research interests include machine learning and data mining.

XIANGJUN DONG received the Ph.D. degree in computer applications from the Beijing Institute of Technology, China, in 2005. From 2007 to 2009, he was a Postdoctoral Fellow with the School of Management and Economics, Beijing Institute of Technology. From 2009 to 2010, he was a Visiting Scholar with the University of Technology Sydney, Australia. He is currently a Professor with the Department of Computer Science and Technology, Qilu University of Technology (Shandong Academy of Sciences), Jinan, China. His research interests include data mining, artificial intelligence, and big data. He has published more than 100 journal/conference publications, including *Artificial Intelligence*, IJCAI, IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS (TNNLS), IEEE TRANSACTIONS ON CYBERNETICS (TCYB), IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS (TII), *Pattern Recognition*, and CIKM.