# Mining Top-*k* Useful Negative Sequential Patterns via Learning

Xiangjun Dong, Ping Qiu, Jinhu Lü, *Fellow, IEEE,* Longbing Cao, *Senior Member, IEEE,* and Tiantian Xu

***Abstract*—As an important tool for behavior informatics, negative sequential patterns (NSPs) (such as missing a medical treatment) are sometimes much more informative than positive sequential patterns (PSPs) (e.g., attending a medical treatment) in many applications. However, NSP mining is at an early stage and faces many challenging problems, including *(1) how to mine an expected number of NSPs*, *(2) how to select useful NSPs*, and *(3) how to reduce high time consumption*. To solve the first problem, we propose an algorithm Topk-NSP to mine the *k* most frequent negative patterns. In Topk-NSP, we first mine the top-*k* PSPs using the existing methods, and then we use an idea which is similar to top-*k* PSPs mining to mine the top-*k* NSPs from these PSPs. To solve the remaining two problems, we propose three optimization strategies for Topk-NSP. The first optimization strategy is that, in order to consider the influence of PSPs when selecting useful top-*k* NSPs, we introduce two weights, $w_P$ and $w_N$, to express the user preference degree for NSPs and PSPs respectively and select useful NSPs by a weighted support *wsup*. The second optimization strategy is to merge *wsup* and an interestingness metric to select more useful NSPs. The third optimization strategy is to introduce a pruning strategy to reduce the high computational costs of Topk-NSP. Finally, we propose an optimization algorithm Topk-NSP$^+$. To the best of our knowledge, Topk-NSP$^+$ is the first algorithm that can mine the top-*k* useful NSPs. The experimental results on four synthetic and two real-life datasets show that the Topk-NSP$^+$ is very efficient in mining the top-*k* NSPs in the sense of computational cost and scalability.**

***Index Terms*—top-*k* positive sequential patterns, top-*k* negative sequential patterns, useful patterns, weighted support, interestingness metric.**

## I. Introduction

BEHAVIOR permeates all aspects of our lives, and how to understand a behavior, especially the non-occurring behaviors (NOB) is a crucial issue in the behavior informatics [1][2][3][4]. Negative sequential pattern (NSP) mining is one of few methods available for understanding NOB [5]. NSPs refer to frequent sequences with non-occurring and occurring behaviors (also called *negative* and *positive* behaviors in behavior and sequence analysis). Sometimes, NSPs play an important role in many real-world applications, such as intrusion detection systems(IDS), intelligent transport systems (ITS), network, health and medical management systems, biomedical systems, risk management, and counter-terrorism. For instance, in IDS, $s_1 =< abcX >$ is a positive sequential pattern(PSP); $s_2 =< ab\neg cY >$ is a NSP, where *a*, *b* and *c* denote the alarm information codes indicating the alarms a network device has issued, and *X* and *Y* denote the device status. $s_1$ dicates that devices which usually issue alarm information *a*, *b* and then *c* are likely to have anomaly status *X*, whereas $s_2$ indicates that devices which issue alarms *a* and *b* but NOT *c* have a high probability of having normal status *Y*. In ITS, negative driving behavior patterns result in drivers failing to follow certain traffic rules which could cause serious traffic problems or even disasters. These situations cannot be handled by the identification of PSP alone.

However, there have not been many recent advances in NSP mining methods[6][7][8]. Most of them, except e-NSP [6], are very inefficient because they calculate the support of negative sequential candidates (NSCs) by additionally scanning the database after identifying PSPs [9][10][11][12]. E-NSP is the most time efficient method to date. It first defines negative containment that is consistent with set theory, then converts the negative containment problem to a positive containment problem, and then rapidly calculates the support of NSCs only using the corresponding PSP's information, thereby avoiding an additional scan of the database. As a result, e-NSP obtains high time efficiency. Although e-NSP effectively improves efficiency, it does not consider the following critical problems.

*(1) how to mine an expected number of NSPs.* Most of the existing NSP algorithms are based on setting a minimum threshold (such as the minimum support *ms*) [6][10][11][12][13][14][15][16]. However, because of limited professional knowledge, it is very difficult to set a rational minimum threshold and obtain an expected number of patterns [17][18][19][20]. A too small value of *ms* may lead to the generation of thousands of patterns, whereas no answer is found as a result of a too large value. Adjusting *ms* is very time consuming.

A similar problem occurred in PSP mining and was solved by top-*k* PSP mining, where *k* is the expected number of PSPs [21][22][23][24][25]. The basic idea of top-*k* PSP mining is as follows. First, it calculates the support of the first *k* positive sequences and adds them in the order set, denoted by $\{s_1,s_2,...s_k\}$, where $sup(s_i) < sup(s_{i+1})$ ($sup(s_i)$ denotes the

This work was supported partly by National Natural Science Foundation of China (71271125), Natural Science Foundation of Shandong Province, China (ZR2018MF011).

Xiangjun Dong and Tiantian Xu are with the School of Information, Qilu University of Technology (Shandong Academy of Sciences), Jinan, China(Email: d-xj@163.com), xtt-ok@163.com)

Ping Qiu is with of computing, Beijing Institute of Technology, Beijing,China(Email: 3120185493@bit.edu.cn)

Jinhu Lü is with the School of Automation Science and Electrical Engineering, State Key Laboratory of Software Development Environment, Beijing Advanced Innovation Center for Big Data and Brain Computing, Beihang University, Beijing 100190, China, also with the Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China(Email:jhlu@iss.ac.cn)

Longbing Cao is with the Advanced Analytics Institute, University of Technology, Sydney, Australia(Email: longbing.cao@ uts.edu.au)

support of $s_i$) and $1<i<k$. Second, it calculates $sup(s_{k+1})$ and compares $sup(s_{k+1})$ with $ms = sup(s_k)$ to update the order set and raise *ms*. In this way, users can obtain the expected number of PSP without setting *ms*. Several methods, such as TSP [17], SkOPUS [20] and TKS [21], have been proposed to mine top-*k* PSP. These methods, however, do not consider negative sequences and cannot be used to mine top-*k* NSPs due to the intrinsic properties caused by non-occurring items in NSP mining [6]. In fact, we have not found any methods by which to mine top-*k* NSPs so far. So, in this paper, we propose a top-*k* NSP mining method, named Topk-NSP, to mine top-*k* NSPs. We first mine top-*k* PSPs using the existing methods; then we generate NSCs in the same way as in e-NSP; finally, we use an idea which is similar to top-*k* PSP mining to mine top-*k* NSPs only according to $sup(nsp)$.

*(2) How to select useful NSPs*. Although Topk-NSP can obtain the expected number of patterns, it cannot guarantee that these patterns are useful. In order to solve this problem, several solutions have been proposed, such as SAP, SAPNSP and SAPBN methods [7][8][31]. Although these methods can be used to select useful NSPs to some degree, they do not consider the influence of PSPs when selecting useful NSPs. In real applications, some users prefer to get the NSP with high support, while others want to get the corresponding PSP with high support. For instance, suppose $p_1 =< abc >$ is a PSP, and $p_2 =< ab\neg c >$ is a NSP, and *a*, *b*, *c* denote the insurance item codes in a customer database of an insurance company. The larger the $sup(p_1)$, the greater probability that customers will buy *c* after buying *a* and *b*. Therefore, in the development of promotional programs, the insurance company must consider the influence of $sup(p_1)$ on $sup(p_2)$, rather than simply use $sup(p_2)$ to make decisions.

In order to consider the influence of PSPs when selecting useful NSP, we propose our first optimization strategy to optimize Topk-NSP in this paper. We introduce two weights, $w_P$ and $w_N$, to express a user preference degree for NSPs and PSPs respectively and use a weighted support (*wsup*) instead of $sup(nsp)$. *wsup* considers not only $sup(nsp)$ but also $sup(psp)$. Users can choose their preferred NSPs or PSPs by changing $w_P$ and $w_N$.

However, the value of *wsup* may be the same for different PSPs and NSPs. For instance, suppose that $wsup(nsp_1)$ is equal to $wsup(nsp_2)$, but $psp_1$ and $nsp_1$ are different from $psp_2$ and $nsp_2$, how to judge whether $nsp_1$ or $nsp_2$ is more useful? In order to solve this problem, we propose the second optimization strategy. We introduce an interestingness metric to judge the interest between *psp* and *nsp*, as conducted in [7][8][29][30] and we merge *wsup* and the interestingness metric to a new metric, interest *wsup*, denoted by *iwsup*, to select more useful NSPs.

*(3) How to avoid high time consumption*. Although Topk-NSP can obtain the expected number of useful NSPs with the above two optimization strategies, it needs to calculate the *iwsup* of all NSCs and compare them with the *iwsup* in the current top-*k* NSC set one by one in the process of selecting useful NSPs. This leads to high time consumption. To solve this problem, we propose the third optimization strategy that only the *iwsup* of part of NSCs needs to be calculated. That is, some NSCs have been pruned before their *iwsup* are calculated, which avoids high time consumption.

Based on the three optimization strategies, we obtain an optimizing algorithm Topk-NSP$^+$. To the best of our knowledge, Topk-NSP$^+$ is the first algorithm that can mine the top-*k* useful NSPs. The significant contributions of this paper are as follows.

Firstly, we propose the Topk-NSP method to mine top-*k* NSPs without a minimum threshold.

Secondly, in order to consider the influence of PSPs when selecting useful NSPs, we propose an optimization strategy: using two weights to express user preference and select useful NSPs by a weighted support *wsup*.

Thirdly, in order to solve the problem of choosing which NSP is more useful when their *wsup* are the same, we propose the second optimization strategy. We introduce an interestingness metric and merge it with *wsup* to *iwsup*, to judge which one is more useful.

Fourthly, in order to avoid high time consumption, we propose the third optimization strategy: we do not calculate the *iwsup* of all NSCs, i.e., we prune some NSCs before their *iwsup* are calculated.

Finally, we propose a corresponding algorithm Topk-NSP$^+$. The experiment results show that Topk-NSP$^+$ can obtain the top-*k* useful NSPs efficiently.

The remainder of the paper is organized as follows. Section II discusses the related work. Section III presents the preliminaries. Section IV proposes the Topk-NSP algorithm. The three optimization strategies and the Topk-NSP$^+$ algorithm are detailed in Section V. Section VI presents the experiment results. The conclusions and future work are detailed in Section VII.

## II. RELATED WORK

In this section, we summarize the related work from the following four aspects: (1) the research status of top-*k* PSP mining; (2) the research status of NSP mining; (3) the research status of useful patterns mining; (4) the research status of weighted sequential pattern mining.

### *A. The Research Status of Top-k PSP Mining*

In real applications, because of limited professional knowledge, users or researchers experience difficulty in directly setting a rational minimum threshold to discover an expected number of patterns [17][18][19][20][47]. A too small value for the minimum threshold may lead to the generation of thousands of patterns, whereas no answer is found as a result of a too large value. Adjusting the minimum threshold is very time-consuming. In order to solve this problem, the concept of top-*k* PSP mining is proposed [17][18][19][20].

The concept of top-*k* PSP mining first evolved from frequent item set mining to solve the difficulty in obtaining the expected number of patterns without setting *ms* [17][30][33]. TSP is a multi-pass search space traversal algorithm to mine the top-*k* closed PSPs of minimum length in a sequence database [17]. It is based on PrefixSpan [26]which can find the most frequent patterns early in the mining process and allows the dynamic raising of *ms* which is then used to prune unpromising branches in the search space. Although this approach considers the patterns appearing in the database unlike "generate-and-test" algorithms, it cannot be applied to dense databases [18][19]. To solve this problem, the TKS algorithm is proposed [18]. It uses the same vertical database representation and basic candidate generation procedure as SPAM [26]. Also, TKS involves three optimization strategies, namely extending the most promising patterns, discarding infrequent items in candidate generation and candidate pruning with a precedence map to improve efficiency. SkOPUS is another algorithm to mine top-*k* PSPs under a given measure of interest [20]. It can extract the *k* sequential patterns with the highest leverage.

The frequent sequential patterns tend to be similar to each other because they are only composed of limited items and do not always correspond to the interests of analysts. In order to solve the problem, the authors of [21] propose a method to mine various top-*k* PSPs. This method decides on the redundant sequential patterns by evaluating and deleting a variety of items. GepDSP and kDSP-miner are the top-*k* PSP mining algorithms with gap constraint, but the gap constraint of GepDSP is more flexible than kDSP-miner [22][23]. GepDSP not only allows the gap constraints between different pairs of adjacent elements in a pattern to be different but also allows different patterns to use different gap constraints. TUS is a novel method by which to avoid setting the minimum utility value of mining top-*k* high utility PSPs [24].

## B. The Research Status of NSP Mining

Discovering NSPs is very important and sometimes plays a pivotal role in the analysis of occurring behaviors in many intelligent systems and applications [6]. Unlike PSP mining, which has been widely explored, there have not been many recent advancements in NSP mining methods [6]. Next, we briefly introduce the status of NSP mining.

NegGSP is a negative version of the GSP algorithm for NSP mining [15]. It calculates the support of NSCs by re-scanning the database and generates NSPs by comparing the support of NSCs with *ms*. PNSP is another NSP mining algorithm which also uses the re-scanned database to calculate the support of NSCs and compares the support of NSCs with *ms* to generate NSPs [12]. NSPM only deals with the last element in the NSP [34]. The authors of [16] proposed a GA algorithm to mine NSPs. Their idea was derived from biological evolution, and it is generated by crossover and mutation operations, which avoids NSCs generation. The method in [35] only identifies NSPs in the form of $(\neg A, B)$, $(A, \neg B)$ and $(\neg A, \neg B)$ and it requires $A \cap B = \varnothing$, which is a normal constraint in association rule mining but a very strict constraint in sequential pattern mining [6].

The e-NSP algorithm is the most time efficient method for mining NSPs to date. It transforms the negative containment problem into the positive containment problem to avoid database re-scanning [6]. E-NSPFI, E-msNSP and e-RNSP algorithms are three improved versions of e-NSP [10][11][50]. E-NSPFI mines NSPs from both frequent and infrequent positive sequences. E-msNSP adds the concept of multiple minimum supports (*MMS*) in e-NSP and e-RNSP mines repetitive sequence patterns.

## C. The Research Status of Useful Pattern Mining

It is very difficult to select useful patterns from the large number of mining results. To solve this difficulty, many methods are proposed. A method for discovering useful patterns is proposed in [36], which first builds an action tree for the specific application, and then assigns useful patterns to the corresponding nodes of the tree using data mining queries. A framework in [38] is proposed to study sequences of interestingness. The patterns are obtained by this framework to study statistical dependencies to rank the serial patterns interestingness. The authors of [39] apply a domain-independent method to model the domain knowledge and propose several methods to mine useful patterns as well as the top-*k* useful patterns. The author of [43] applies multivariate variable-length sequences to a similarity search.The author of [49] propose a distributed programming model for mining business-oriented transactional datasets by using an improved MapReduce framework on Hadoop, which overcomes not only the single processor and main memory-based computing, but also highly scalable in terms of increasing database size. The authors of [48] propose a new problem: multiple-instance association rule mining, and mine robust and useful patterns from multiple instance datasets. The authors of [49] propose an approach to select useful patterns from a set of patterns by using multi criteria approach.

The number of NSPs is much greater than PSPs. Therefore, it is much more difficult to select useful patterns after mining NSPs [7][8][32]. Next, we briefly introduce the status of useful NSP mining. SAPNSP is a method to select these useful positive and negative sequential patterns [8]. It improved the Wu's method [31][33] to analyze the correlation between elements in a sequential pattern. SAP uses the correlation coefficient to select useful positive and negative sequential patterns and SAPBN uses a Bayesian network (BN) to select useful positive and negative sequential patterns[7][31].

## D. The Research Status of Weighted Sequential Pattern Mining

Weights are very important for expressing users' interest and selecting accurate patterns[40][41]. We briefly introduce weighted sequential pattern mining. The authors of [40] first proposed the problem of weighted sequential pattern mining to find weighted PSPs from sequence database. IUA finds weighted PSPs from sequence databases [41]. It proposes a tightening strategy to obtain more accurate weighted upper-bounds for subsequences in mining. In [42], a method is proposed to mine interest weighted negative association rules from large databases and deletes contrary rules. The authors of [45] propose PCA-WSVM (principal component analysis-weighted support vector machine ) to solve the load forecasting problem and [46] propose a novel building cooling load forecasting approach by combining support vector regression (SVR) and the ant colony algorithm (ACO) is proposed.

## III. PRELIMINARY

### A. Positive Sequential Patterns - PSPs

Let $I = \{x_1, x_2, ..., x_n\}$ be a set of items. An itemset is a subset of *I*, and it is an unordered set of distinct items. A sequence is an ordered list of itemsets. A sequence *s* is denoted by $< s_1 s_2 ... s_l >$, where $s_j \subseteq I$ ($1 \leqslant j \leqslant l$). $s_j$ is also called an element of the sequence, and denoted as $(x_1 x_2 ... x_m)$, where $x_k$ is an item, $x_k \in I$ ($1 \leqslant k \leqslant m$), *j* is the *id* of the element. For simplicity, the bracket is omitted if an element only has one item, i.e., element $(x)$ is coded *x*.

The *size* of sequence *s*, denoted as $size(s)$, is the total number of elements in *s*. *s* is a *k*-size sequence if $size(s) = k$. For example, a given sequence $s =< (ab)cd >$ is composed of 3 elements $(ab)$, *c* and *d*. Therefore, *s* is a 3-size sequence, i.e., $size(s) = 3$.

Sequence $s_\alpha =< \alpha_1 \alpha_2 ... \alpha_n >$ is called a sub-sequence of sequence $s_\beta =< \beta_1 \beta_2 ... \beta_m >$ and $s_\beta$ is a super-sequence of $s_\alpha$, denoted as $s_\alpha \subseteq s_\beta$, if there exists $1 \leqslant j_1 < j_2 < ... < j_n \leqslant m$ such that $\alpha_1 \subseteq \beta_{j1}$, $\alpha_2 \subseteq \beta_{j2}, ..., \alpha_n \subseteq \beta_{jn}$. We also say that $s_\beta$ contains $s_\alpha$. For example, $< b >, < ad >$and $< (ab)d >$ are all subsequences of $< (ab)cd >$.

A *sequence database* D is a set of tuples $< sid, ds >$, where *sid* is the *sequence_id* and *ds* is the *data sequence*. The number of tuples in D is denoted as |D|. The set of tuples containing sequence *s* is denoted as $\{< s >\}$. The support of *s*, denoted as $sup(s)$, is the number of tuples that are contained in $\{< s >\}$, where $\{< s >\}$ is the set of all tuples that contains sequence *s* in D. That is, $sup(s) = |\{< s >\}| = |\{< sid, ds >, < sid, ds > \in D \wedge (s \subseteq ds)\}|$. *ms* is a minimum support threshold predefined by users. Sequence *s* is called a frequent (positive) sequential patterns if $sup(s) \geqslant ms$. By contrast, *s* is infrequent if $sup(s) < ms$. Sequence *s* is called a frequent (positive) sequential patterns if $sup(s) \geqslant ms$. By contrast, *s* is infrequent if *sup*(*s*)<*ms*.

PSP mining aims to discover all positive sequences that satisfy a given minimum support. For simplicity, we often omit 'positive' when discussing positive items, positive elements and positive sequences in mining PSPs.

### B. Negative Sequential Patterns - NSPs

In a negative sequence, a non-occurring item/element is called a negative item/element. The number of negative elements in *ns* is denoted by *neg-size(ns)*. *ns* is a *m*-size and *n*-neg-size negative sequence if $size(ns) = m$ and $neg - size(ns) = n$.

*1) Three Constraints:* In real applications, the number of NSCs and the identified negative sequences are often large, and many of them are meaningless [6]. The number of NSCs may be huge or even infinite if no constraints are added. This makes NSP mining very challenging. In order to solve this problem, some constraints are introduced in many existing methods. E-NSP uses three constraints and gives corresponding reasons. In fact, these constraints are a bit strict and not necessary in real applications. We will loosen these constraints in our future work. This paper also uses the same constraints which are introduced in details in the following subsections. We first introduce the definition of a positive partner which is used in the constraints.

***Definition 1. Positive Partner.*** The positive partner of a negative element $\neg e$ is *e*, denoted as $p(\neg e)$, i.e., $p(\neg e) = e$; the positive partner of positive element *e* is *e* itself, i.e., $p(e) = e$. The positive partner of a negative sequence $ns =< s_1 ... s_k >$ can be obtained by converting all negative elements in *ns* to their positive partners, denoted as $p(ns)$, i.e.,$p(ns) = \{< s'_1 ... s'_k > | s'_i = p(s_i), s_i \in ns\}$. For example, $p(< \neg(ab)c\neg d >) =< (ab)cd >$.

***Constraint 1. Frequency constraint.*** For simplicity, this paper only focuses on the negative sequences *ns* whose positive partners are frequent, i.e., $sup(p(ns)) \geqslant ms$.

***Constraint 2. Format constraint.*** Continuous negative elements in an NSC are not allowed.

**Example 1.** $< \neg(ab)c\neg d >$ satisfies **Constraint 2**, but $< \neg(ab)\neg cd >$ does not.

***Constraint 3. Negative element constraint.*** The smallest negative unit in an NSC is an element. If an element consists of more than one item, either all or none of the items are allowed to be negative.

**Example 2.** $< \neg(ab)cd >$ satisfies constraint 3, but $< (\neg ab)cd >$ does not, because in element $(\neg ab)$, only $\neg a$ is negative while *b* is not.

*2) NSP Concepts:* The definition of negative containment is very important to improve the efficiency of NSP mining algorithm because it affects the efficiency of calculating the support of NSCs. In e-NSP, the definition of negative containment is consistent with set theory. This paper also uses the same definitions, but we simplify them in an easily understandable way. Before we formally define negative containment, two preparatory definitions should be given first. One is a maximum positive subsequence that contains all positive elements of a negative sequence. The other is 1-neg-size maximum subsequence that contains a maximum positive subsequence and one negative element. The definitions are as follows.

***Definition 2. Maximum Positive Subsequence***. Let $ns =< s_1 s_2 ... s_m >$ be a *m*-size and *n*-neg-size negative sequence ($m - n > 0$), the subsequence that contains all positive elements with the same order as *ns* is called the maximum positive subsequence, denoted as $MPS(ns)$.

**Example 3.** Given a negative sequence $ns =< a\neg bb\neg a(cde) >$, its $MPS(ns) =< ab(cde) >$.

***Definition 3. 1-neg-size Maximum Subsequence***. Let $ns =< s_1 s_2 ... s_m >$ be a *m*-size and *n*-neg-size negative sequence, the subsequence that contains all positive elements and one negative element with the same order as *ns* is called a 1-neg-size maximum subsequence, denoted as $1 - negMS_{ns}$. The subsequence set including all 1-neg-size maximum subsequences of *ns* is called 1-neg-size maximum subsequence set, denoted as $1 - negMSS_{ns}$.

**Example 4.** For $ns =< a\neg bb\neg a(cde) >$, its $1 - negMSS_{ns} = \{< a\neg bb(cde) >, < ab\neg a(cde) >\}$.

***Definition 4. Negative containment.*** Given a data sequence *ds* and a negative sequence *ns*, *ds* contains *ns* if and only if two conditions hold: (1) *MPS(ns)*$\subseteq$*ds*; and (2) $\forall 1 - negMS \in 1 - negMSS_{ns}, p(1 - negMS) \nsubseteq ds$.

Given $ds =< a(bc)d(cde) >$ and $ns =< a\neg bb\neg a(cde) >$, *ds* contains *ns* if and only if *ds* contains $MPS(ns) =< ab(cde) >$ and *ds* does not contain $p(< a\neg bb(cde) >=< abb(cde) >$ and $p(< ab\neg a(cde) >=< aba(cde) >$, i.e., $1-negMSS_{<a\neg bb\neg a(cde)>} = \{< a\neg bb(cde) >, < ab\neg a(cde) >\}$.

From Definition 4, we can see that the negative containment now is converted to positive containment: a data sequence contains a positive sequence but does not contain some other related positive sequences. In this way, we can calculate the support of negative sequences by only using the information of corresponding positive sequences.

***Definition 5. Negative Sequential Pattern***. A negative sequence *ns* is a negative sequential pattern (NSP) if $sup(ns) \geqslant ms$.

In this paper, we only mine top-*k* NSPs from the top-*k* PSPs. Hence, we avoid setting *ms*.

## IV. TOPK-NSP ALGORITHM

In this section, the definition of top-*k* NSPs is proposed. The steps of Topk-NSP are given. Finally, the corresponding pseudo code is given.

### A. Top-k NSP Mining Definition

Because the definition of top-*k* NSPs and top-*k* PSPs are closely related, we first introduce the definition of top-*k* PSPs.

***Definition 6. top-k PSPs.*** Top-*k* PSPs is a set that contains the *k* most frequent PSPs in a sequence database D. That is, for each pattern $s \in$top-*k* PSP, there does not exist a PSP $s' \notin top-kPSPs | sup(s') > sup(s)$.

***Definition 7. top-k NSPs.*** Top-*k* NSPs is a set that contains the *k* most frequent NSPs mined from top-*k* PSPs. That is, for each pattern $s \in$top-*k* NSPs, there does not exist an NSP $s' \notin top-kNSPs$ and $p(s') \in top-kPSPs | sup(s') > sup(s)$.

Note that *k* in top-*k* PSP and top-*k* NSP may be different value.

### B. Steps of Topk-NSP

**Step 1:** use existing algorithms to mine the top-*k* PSPs. In this paper, we improve GSP to mine the top-*k* PSPs;

**Step 2:** use *NSCgeneration* method to generate NSCs from these PSPs as follows:

For a *k*-size PSP, its NSCs are generated by changing any *m* non-contiguous elements to their negative elements, *m* =1, 2, ...,$\lceil k/2 \rceil$, where $\lceil k/2 \rceil$ is a minimum integer that is not less than $k/2$.

**Example 5.** The NSC based on $< (ab)cd >$include: *m*=1, $< \neg(ab)cd >$, $< (ab)\neg cd >$, $< (ab)c\neg d >$; *m*=2, $< \neg(ab)c\neg d >$.

**Step 3:** use equations in e-NSP to calculate the support of all NSCs as follows:

Given a *m*-size and *n*-neg-size negative sequence *ns*, for $\forall 1-negMS_i \in 1-negMSS_{ns} (1 \leqslant i \leqslant n)$, the support of *ns* in sequence database D is:

$$sup(ns) = sup(MPS(ns)) - |\cup_{i=1}^{n} p(1-negMS_i)| \quad (1)$$

where *p(1-negMS$_i$)* is a *sid* set of *ds* that contains *p(1-negMS$_i$)*, $\cup_{i=1}^{n}\{p(1-negMS_i)\}$ is the union of all *sid* of $1-negMSS_{ns}$ and $|\cup_{i=1}^{n}\{p(1-negMS_i)\}|$ the *sid* numbers of $\cup_{i=1}^{n}\{p(1-negMS_i)\}$.

If *ns* only contains a negative element, the support of *ns* is:

$$sup(ns) = sup(MPS(ns)) - sup(p(ns)) \quad (2)$$

In particular, for negative sequence $< \neg e >$,

$$sup(< \neg e >) = |D| - sup(< e >) \quad (3)$$

**Step 4:** add the first *k* NSC to top-*k* NSP, denoted by $\{nsc_1$, $nse_2$,..., $nsc_k\}$, where $sup(nsc_1) > sup(nsc_2) > \ldots > sup(nsc_k)$ and then compare the other *sup(nsc)* with $nsc_k$ to update top-*k* NSP. If $sup(nsc) > sup(nsc_k)$, delete $nsc_k$ and add the *nsc* in top-*k* NSP.

**Step 5:** output the maximum support of the first *k* NSC, i.e., top-*k* NSP. Algorithm 1. Topk-NSP Algorithm.

**Input:** Sequence dataset *D* and Parameter *k*;

**Output:** top-*k*NSP: a set that contains *k* negative sequential patterns, where the sequences are arranged in descending order of support;

```
(1) top-kPSP=∅,NSC=∅,top-kNSP=∅;
(2)top-kPSP=minePSP();
    // improved GSP algorithm to mine top-k PSP;
(3) For (each psp in top-kPSP){
(4)       Generating NSC by using NSCgeneration from psp;
(5)     For (each nsc in NSC){
(6)         If ($size(nsc)$=1) {
(7)             $sup(nsc)$ is calculated by equation (3);}
(8)             else if ($n - size(nsc)$=1){
(9)                 $sup(nsc)$ is calculated by equation (2);
(10)                }
(11)                else {
(12)                    $sup(nsc)$ is calculated by equation (1);
(13)                }
(14)        If (|{top-kNSP}| $< k$){/* {top-kNSP} is the
            number of sequences in top-kNSP;*/
(15)            Insert {$nsc$} to {top-kNSP};
            //$\forall nsp_i \in${top-kNSP},$sup(nsp_i) \geqslant sup(nsp_{i-1})$;
(16)        }else if ($sup(nsc) > sup(nsc_k)$){
(17)            Delete {$nsc_k$} from {top-kNSP};
(18)            Insert {$nsc$} to {top-kNSP};
(19)        }
(20)    }
(21) }
(22) return top-kNSP;
```

1) Line 2 finds the top-*k* PSPs from sequence database *D* using existing methods. To efficiently calculate the union set, we first improve the GSP algorithm to deal with a data structure that contains PSPs, their support values and $\{sid\}$, where $\{sid\}$ represents the set of tuples that contain the corresponding PSPs. Second, we set *k* to constrain the number of NSPs.
2) Line 4 generates NSCs from those PSPs using the *NSCgeneration* method.
3) Lines 5 to 13 calculate the support of *nsc* by equations (1), (2) and (3).

4) Lines 14 to 17 add the first *k* NSC in top-*k* NSP and from lines 17 to 20, the top-*k* NSP is updated only according to *sup(nsc)*.

5) Line 22 returns the results and ends the algorithm.

**Example 6.** Suppose *k*=10. A customer database of an insurance company containing five data sequences is shown in Fig.1(a). Fig.1 shows the process of Topk-NSP.

In Fig.1, (a) presents dataset, (b) presents top-*k* PSPs, (c) presents the NSCs and their *sup* and (d) presents top-*k* NSPs with *sup*. From (a) to (b), we use the improved GSP to mine top-*k* PSPs and from (b) to (c), we generate NSCs from each *psp* and calculate their support. At the same time, we constantly update the set of NSPs (top-*k* NSP)from (c) to (d). Now we take $< ad >, sup(< ad >) = 0.6$ and $< bd >, sup(< bd >) = 0.8$ (in Fig.1(b)) for example, to explain the process (b) to (c). By **step 2**, the NSCs generated from $< ad >$ are: $< \neg ad >, < a\neg d >$ and $sup(< \neg ad >) = sup(MPS(< \neg ad >)) - sup(< ad >) = 1 - 0.6 = 0.4$; the NSCs generated from $< bd >$ are: $< \neg bd >, < b\neg d >$ and $sup(< \neg bd >) = sup(MPS(< \neg bd >)) - sup(< bd >) = 1 - 0.8 = 0.2$. The result of top-*k* NSPs is shown in (d), which is updated by *sup(nsp)*.

Although the Topk-NSP method can obtain the expected number of patterns, it does not consider the influence of PSPs when selecting useful NSPs. In addition, it needs to calculate the *iwsup* of all NSCs one by one which leads to high time consumption. Therefore, we propose three optimizations to Topk-NSP and based on these, we propose an optimized Topk-NSP$^+$.

## V. TOPK-NSP$^+$: THE OPTIMIZATION OF TOPK-NSP

The three optimizations are as follows. (1) We introduce two weights, $w_P$ and $w_N$, to express the user preference degree for NSPs and PSPs respectively and select useful NSPs by a weighted average support *wsup*. (2) We merge *wsup* and an interestingness metric to select more useful NSPs. (3) We propose a pruning strategy to improve the efficiency of Topk-NSP.

### *A. The First Optimization: Weighted Support*

As introduced in section I, in order to consider the influence of PSPs when selecting useful NSPs, we use two weights, $w_P$ and $w_N$, to express a user preference values for NSPs and PSPs respectively and use a weighted support *wsup* instead of $sup(nsp)$. We use the following equation to calculate *wsup*.

$$wsup(psp, nsp) = w_P * sup(psp) + w_N * sup(nsp), \quad (4)$$

where $w_P$+$w_N$=1.

Because $psp = p(nsp)$ in $wsup(psp, nsp)$, we use $iwsup(nsp)$ instead of $iwsup(psp, nsp)$ for simplicity. That is,

$$wsup(nsp) = w_P * sup(p(nsp)) + w_N * sup(nsp) \quad (5)$$

In the process of selecting useful NSPs, when $w_P$ is large, the influence of PSPs is strong, i.e., the user preference degree for PSPs is high, and when $w_P$ is small, the influence of PSPs is weak, i.e., the user preference degree for PSPs is low. In particular, when $w_P$=0, the influence of PSPs is none and $wsup(nsp) = sup(nsp)$, which is the Topk-NSP algorithm. Here we use an example to illustrate the effectiveness of *wsup*.

**Example 7.** Suppose $w_P$=0.5, $w_N$=0.5 and *k*=10, Fig.2 shows the process of Topk-NSP with *wsup*.

In Fig.2, (e) presents dataset, (f) presents top-*k* PSPs, (g) presents the NSCs and their *wsup* and (h) presents top-*k* NSPs with *wsup*. The difference between Fig.1 and Fig.2 is that the last column of Fig.2(g) calculates the *wsup* of all NSCs using equation (5) and the top-*k* NSP are ordered by $wsup(nsp)$ in Fig.2(h). From Fig.1(d) and Fig.2(h), we can see that $< a\neg b >, < \neg ba >$ and$< a\neg ba >$ are in (d), but not in (h); $< \neg d >, < b\neg d >$ and$< \neg aa >$ are in (h), but not in (d); although $< \neg a >, < \neg b >, < \neg ad >, < \neg bd >, < \neg ab >, < b\neg a >$ and $< \neg ab\neg a >$ are in Fig.1(d) and Fig.2(h), their orders are different, except $< \neg a >$. Obviously, $wsup(nsp)$ has a great influence on the mining results.

According to **example 7**, the support of $< ad >$ and $< \neg ad >$ are 0.6 and 0.4 respectively. For $< \neg ad >$, $wsup(< \neg ad >) = w_P * sup(< ad >) + w_N * sup(< \neg ad >) = 0.5 * 0.6 + 0.5 * 0.4 = 0.5$. The support of $< bd >$ and $< \neg bd >$ are 0.8 and 0.2 respectively. For $< \neg bd >$, $wsup(< \neg bd >) = w_P * sup(< bd >) + w_N * sup(< \neg bd >) = 0.5 * 0.8 + 0.5 * 0.2 = 0.5$. So we cannot judge which one is the more useful from $< \neg ad >$ and $< \neg bd >$ by equation 3. Therefore, we propose the second optimization strategy to improve *wsup*.

### *B. The Second Optimization: Interestingness Metric*

In order to solve the above problem, we need to find a method to judge the interest between two patterns. The authors of [32] first proposed an interestingness metric $interest(X, Y) = |s(X \cup Y) - s(X)s(Y)|$ to judge the interest between itemsets, where *X* and *Y* express different itemsets respectively. It is a good metric to express the interestingness of two variables and we also use this metric in this paper by simply replacing *X* and *Y* with NSP and PSP. According to the NSC generation method in topk-NSP, $p(nsp) = psp$, so $sup(nsp \cup psp) = 0$. Therefore, we improve the interestingness metric to judge the interest between NSPs and PSPs as follows.

$$\begin{aligned} interest(psp, nsp) &= |sup(psp \cup nsp) - sup(psp) * sup(nsp)| \\ &= sup(psp) * sup(nsp) \end{aligned} \quad (6)$$

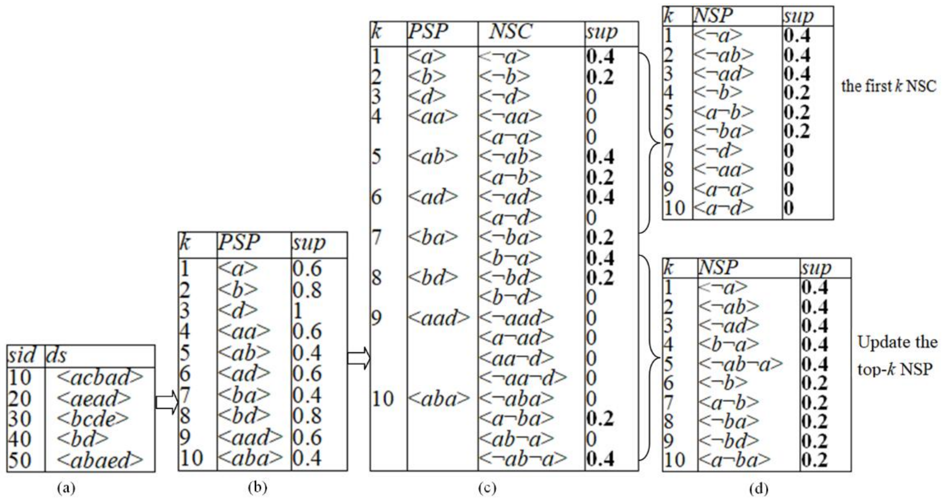
We merge *wsup(nsp)* and *interest(psp,nsp)*, denoted by *iwsup(nsp)*, as follows:

$$\begin{aligned} iwsup(nsp) = \; & sup(p(nsp)) * sup(nsp) + w_P \\ & * sup(p(nsp)) + w_N * sup(nsp) \end{aligned} \quad (7)$$

Here, we use an example to illustrate the effectiveness of *iwsup(nsp)*.

**Example 8.** Suppose $w_P$=0.5, $w_N$=0.5 and *k*=10, Fig.3 shows the process of Topk-NSP with $iwsup(nsp)$.

In Fig.3, (i) presents dataset, (j) presents top-*k* PSPs, (k) presents the NSCs and their *iwsup* and (l) presents top-*k* NSPs with *iwsup*. The difference between Fig.2 and Fig.3 is that the last column of Fig.3(k) calculates the *iwsup* of all NSCs by


the first *k* NSC
Update the top-*k* NSP

(a)

| *sid* | *ds* |
|---|---|
| 10 | <*acbad*> |
| 20 | <*aead*> |
| 30 | <*bcde*> |
| 40 | <*bd*> |
| 50 | <*abaed*> |

(b)

| *k* | *PSP* | *sup* |
|---|---|---|
| 1 | <*a*> | 0.6 |
| 2 | <*b*> | 0.8 |
| 3 | <*d*> | 1 |
| 4 | <*aa*> | 0.6 |
| 5 | <*ab*> | 0.4 |
| 6 | <*ad*> | 0.6 |
| 7 | <*ba*> | 0.4 |
| 8 | <*bd*> | 0.8 |
| 9 | <*aad*> | 0.6 |
| 10 | <*aba*> | 0.4 |

(c)

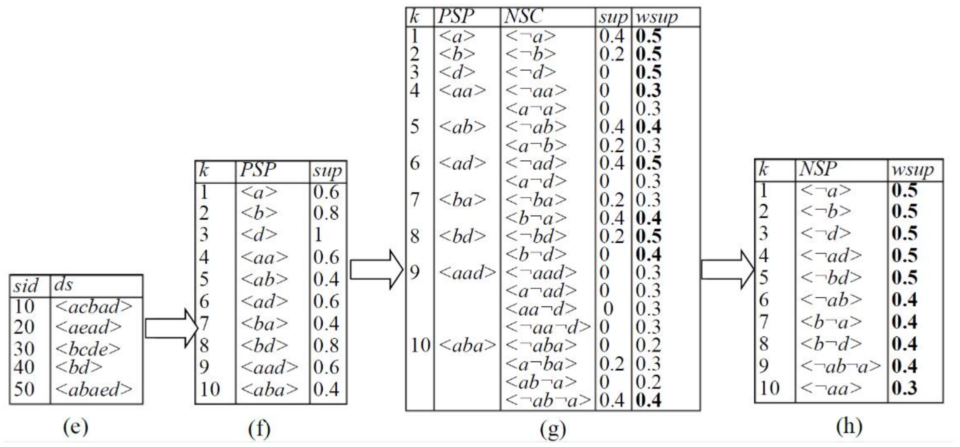
| *k* | *PSP* | *NSC* | *sup* |
|---|---|---|---|
| 1 | <*a*> | <¬*a*> | **0.4** |
| 2 | <*b*> | <¬*b*> | **0.2** |
| 3 | <*d*> | <¬*d*> | 0 |
| 4 | <*aa*> | <¬*aa*> | 0 |
| | | <*a*¬*a*> | 0 |
| 5 | <*ab*> | <¬*ab*> | **0.4** |
| | | <*a*¬*b*> | **0.2** |
| 6 | <*ad*> | <¬*ad*> | **0.4** |
| | | <*a*¬*d*> | 0 |
| 7 | <*ba*> | <¬*ba*> | **0.2** |
| | | <*b*¬*a*> | **0.4** |
| 8 | <*bd*> | <¬*bd*> | **0.2** |
| | | <*b*¬*d*> | 0 |
| 9 | <*aad*> | <¬*aad*> | 0 |
| | | <*a*¬*ad*> | 0 |
| | | <*aa*¬*d*> | 0 |
| | | <¬*aa*¬*d*> | 0 |
| 10 | <*aba*> | <¬*aba*> | 0 |
| | | <*a*¬*ba*> | **0.2** |
| | | <*ab*¬*a*> | 0 |
| | | <¬*ab*¬*a*> | **0.4** |

(d)

| *k* | *NSP* | *sup* |
|---|---|---|
| 1 | <¬*a*> | **0.4** |
| 2 | <¬*ab*> | **0.4** |
| 3 | <¬*ad*> | **0.4** |
| 4 | <¬*b*> | **0.2** |
| 5 | <*a*¬*b*> | **0.2** |
| 6 | <¬*ba*> | **0.2** |
| 7 | <¬*d*> | **0** |
| 8 | <¬*aa*> | **0** |
| 9 | <*a*¬*a*> | **0** |
| 10 | <*a*¬*d*> | **0** |

| *k* | *NSP* | *sup* |
|---|---|---|
| 1 | <¬*a*> | **0.4** |
| 2 | <¬*ab*> | **0.4** |
| 3 | <¬*ad*> | **0.4** |
| 4 | <*b*¬*a*> | **0.4** |
| 5 | <¬*ab*¬*a*> | **0.4** |
| 6 | <¬*b*> | **0.2** |
| 7 | <*a*¬*b*> | **0.2** |
| 8 | <¬*ba*> | **0.2** |
| 9 | <¬*bd*> | **0.2** |
| 10 | <*a*¬*ba*> | **0.2** |

Fig. 1: The process of Topk-NSP with *sup*.



(e)

| *sid* | *ds* |
|---|---|
| 10 | <*acbad*> |
| 20 | <*aead*> |
| 30 | <*bcde*> |
| 40 | <*bd*> |
| 50 | <*abaed*> |

(f)

| *k* | *PSP* | *sup* |
|---|---|---|
| 1 | <*a*> | 0.6 |
| 2 | <*b*> | 0.8 |
| 3 | <*d*> | 1 |
| 4 | <*aa*> | 0.6 |
| 5 | <*ab*> | 0.4 |
| 6 | <*ad*> | 0.6 |
| 7 | <*ba*> | 0.4 |
| 8 | <*bd*> | 0.8 |
| 9 | <*aad*> | 0.6 |
| 10 | <*aba*> | 0.4 |

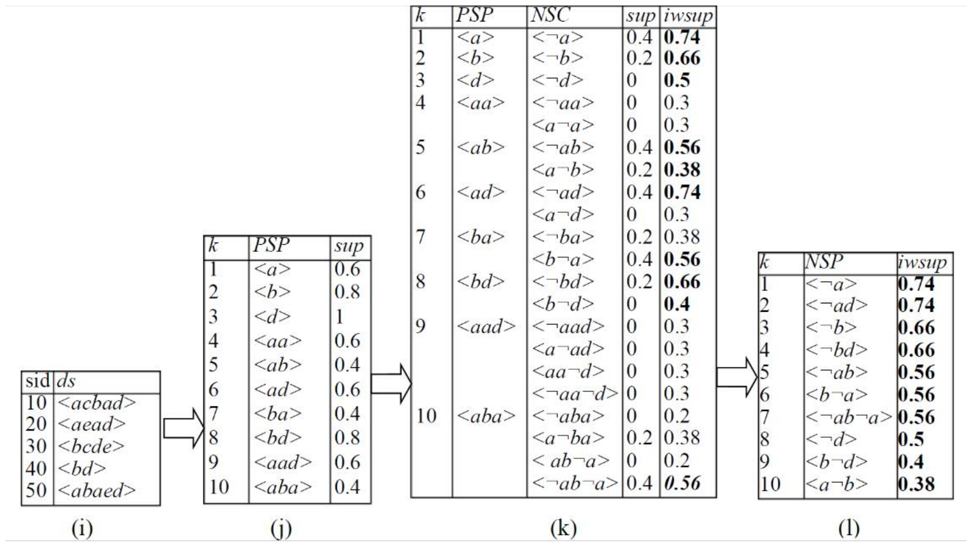
(g)

| *k* | *PSP* | *NSC* | *sup* | *wsup* |
|---|---|---|---|---|
| 1 | <*a*> | <¬*a*> | 0.4 | **0.5** |
| 2 | <*b*> | <¬*b*> | 0.2 | **0.5** |
| 3 | <*d*> | <¬*d*> | 0 | **0.5** |
| 4 | <*aa*> | <¬*aa*> | 0 | **0.3** |
| | | <*a*¬*a*> | 0 | 0.3 |
| 5 | <*ab*> | <¬*ab*> | 0.4 | **0.4** |
| | | <*a*¬*b*> | 0.2 | 0.3 |
| 6 | <*ad*> | <¬*ad*> | 0.4 | **0.5** |
| | | <*a*¬*d*> | 0 | 0.3 |
| 7 | <*ba*> | <¬*ba*> | 0.2 | 0.3 |
| | | <*b*¬*a*> | 0.4 | **0.4** |
| 8 | <*bd*> | <¬*bd*> | 0.2 | **0.5** |
| | | <*b*¬*d*> | 0 | **0.4** |
| 9 | <*aad*> | <¬*aad*> | 0 | 0.3 |
| | | <*a*¬*ad*> | 0 | 0.3 |
| | | <*aa*¬*d*> | 0 | 0.3 |
| | | <¬*aa*¬*d*> | 0 | 0.3 |
| 10 | <*aba*> | <¬*aba*> | 0 | 0.2 |
| | | <*a*¬*ba*> | 0.2 | 0.3 |
| | | <*ab*¬*a*> | 0 | 0.2 |
| | | <¬*ab*¬*a*> | 0.4 | **0.4** |

(h)

| *k* | *NSP* | *wsup* |
|---|---|---|
| 1 | <¬*a*> | **0.5** |
| 2 | <¬*b*> | **0.5** |
| 3 | <¬*d*> | **0.5** |
| 4 | <¬*ad*> | **0.5** |
| 5 | <¬*bd*> | **0.5** |
| 6 | <¬*ab*> | **0.4** |
| 7 | <*b*¬*a*> | **0.4** |
| 8 | <*b*¬*d*> | **0.4** |
| 9 | <¬*ab*¬*a*> | **0.4** |
| 10 | <¬*aa*> | **0.3** |

Fig. 2: The process of Topk-NSP with *wsup*.



(i)

| sid | *ds* |
|---|---|
| 10 | <*acbad*> |
| 20 | <*aead*> |
| 30 | <*bcde*> |
| 40 | <*bd*> |
| 50 | <*abaed*> |

(j)

| *k* | *PSP* | *sup* |
|---|---|---|
| 1 | <*a*> | 0.6 |
| 2 | <*b*> | 0.8 |
| 3 | <*d*> | 1 |
| 4 | <*aa*> | 0.6 |
| 5 | <*ab*> | 0.4 |
| 6 | <*ad*> | 0.6 |
| 7 | <*ba*> | 0.4 |
| 8 | <*bd*> | 0.8 |
| 9 | <*aad*> | 0.6 |
| 10 | <*aba*> | 0.4 |

(k)

| *k* | *PSP* | *NSC* | *sup* | *iwsup* |
|---|---|---|---|---|
| 1 | <*a*> | <¬*a*> | 0.4 | **0.74** |
| 2 | <*b*> | <¬*b*> | 0.2 | **0.66** |
| 3 | <*d*> | <¬*d*> | 0 | **0.5** |
| 4 | <*aa*> | <¬*aa*> | 0 | 0.3 |
| | | <*a*¬*a*> | 0 | 0.3 |
| 5 | <*ab*> | <¬*ab*> | 0.4 | **0.56** |
| | | <*a*¬*b*> | 0.2 | **0.38** |
| 6 | <*ad*> | <¬*ad*> | 0.4 | **0.74** |
| | | <*a*¬*d*> | 0 | 0.3 |
| 7 | <*ba*> | <¬*ba*> | 0.2 | 0.38 |
| | | <*b*¬*a*> | 0.4 | **0.56** |
| 8 | <*bd*> | <¬*bd*> | 0.2 | **0.66** |
| | | <*b*¬*d*> | 0 | **0.4** |
| 9 | <*aad*> | <¬*aad*> | 0 | 0.3 |
| | | <*a*¬*ad*> | 0 | 0.3 |
| | | <*aa*¬*d*> | 0 | 0.3 |
| | | <¬*aa*¬*d*> | 0 | 0.3 |
| 10 | <*aba*> | <¬*aba*> | 0 | 0.2 |
| | | <*a*¬*ba*> | 0.2 | 0.38 |
| | | < *ab*¬*a*> | 0 | 0.2 |
| | | <¬*ab*¬*a*> | 0.4 | ***0.56*** |

(l)

| *k* | *NSP* | *iwsup* |
|---|---|---|
| 1 | <¬*a*> | **0.74** |
| 2 | <¬*ad*> | **0.74** |
| 3 | <¬*b*> | **0.66** |
| 4 | <¬*bd*> | **0.66** |
| 5 | <¬*ab*> | **0.56** |
| 6 | <*b*¬*a*> | **0.56** |
| 7 | <¬*ab*¬*a*> | **0.56** |
| 8 | <¬*d*> | **0.5** |
| 9 | <*b*¬*d*> | **0.4** |
| 10 | <*a*¬*b*> | **0.38** |

Fig. 3: The process of Topk-NSP with *iwsup*.

equation (7) and the top-*k* NSPs are ordered by $iwsup(nsp)$ in Fig.3(l). From Fig.2(h) and Fig.3(l) we can see that $<\neg aa>$ is in Fig.2(h), but not in Fig.3(l); $<a\neg b>$is in Fig.3(l), but not in Fig.2(h); although $<\neg a>, <\neg ad>, <\neg b>, <\neg bd>, <\neg ab>, <b\neg a>, <\neg ab\neg a>, <\neg d>$ $and$ $<b\neg d>$ are in Fig.2(h) and Fig.3(l), their orders are different, except $<\neg a>$. Obviously, *iwsup(nsp)* has a greater influence on the mining results.

For $<\neg ad>$, $iwsup(<\neg ad>) = interest(p(<\neg ad>), <\neg ad>) + w_P * sup(p(<\neg ad>)) + w_N * sup(<\neg ad>) = 0.24 + 0.5 * 0.6 + 0.5 * 0.4 = 0.74$, and for $<\neg bd>$, $iwsup(<\neg bd>) = interest(p(<\neg bd>), <\neg bd>) + w_P * sup(p(<\neg bd>)) + w_N * sup(<\neg bd>) = 0.16 + 0.5*0.8 + 0.5*0.2 = 0.66$. Obviously $<\neg ad>$ is more actionable than $<\neg bd>$.

From Fig.1 to Fig.3, we can see that the *metrics weighted support* and *interestingness* can effectively change the top-*k* useful NSPs, but this cannot be seen in experiments. Therefore, in section VI (experiment results), we do not verify the efficiency of these metrics.

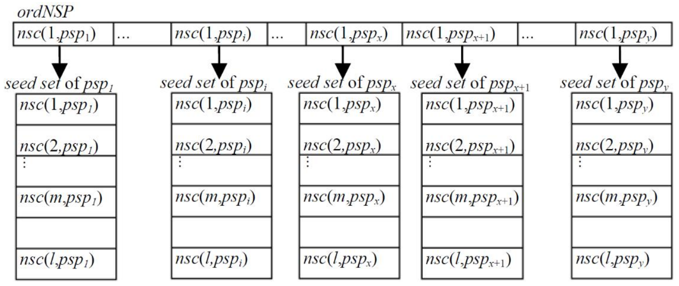
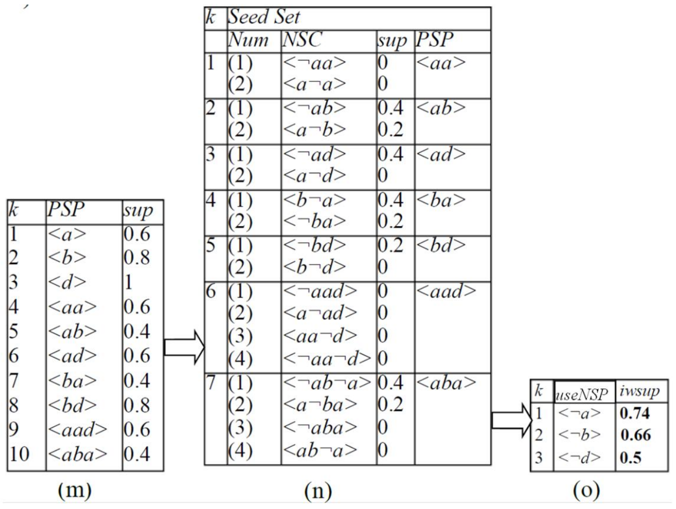


Fig. 4: Schematic Diagram of Generating Seed Sets.



| k | PSP | sup |
|---|---|---|
| 1 | <a> | 0.6 |
| 2 | <b> | 0.8 |
| 3 | <d> | 1 |
| 4 | <aa> | 0.6 |
| 5 | <ab> | 0.4 |
| 6 | <ad> | 0.6 |
| 7 | <ba> | 0.4 |
| 8 | <bd> | 0.8 |
| 9 | <aad> | 0.6 |
| 10 | <aba> | 0.4 |

(m)

| k | Seed Set | | | |
|---|---|---|---|---|
| | Num | NSC | sup | PSP |
| 1 | (1) | <¬aa> | 0 | <aa> |
| | (2) | <a¬a> | 0 | |
| 2 | (1) | <¬ab> | 0.4 | <ab> |
| | (2) | <a¬b> | 0.2 | |
| 3 | (1) | <¬ad> | 0.4 | <ad> |
| | (2) | <a¬d> | 0 | |
| 4 | (1) | <b¬a> | 0.4 | <ba> |
| | (2) | <¬ba> | 0.2 | |
| 5 | (1) | <¬bd> | 0.2 | <bd> |
| | (2) | <b¬d> | 0 | |
| 6 | (1) | <¬aad> | 0 | <aad> |
| | (2) | <a¬ad> | 0 | |
| | (3) | <aa¬d> | 0 | |
| | (4) | <¬aa¬d> | 0 | |
| 7 | (1) | <¬ab¬a> | 0.4 | <aba> |
| | (2) | <a¬ba> | 0.2 | |
| | (3) | <¬aba> | 0 | |
| | (4) | <ab¬a> | 0 | |

(n)

| k | useNSP | iwsup |
|---|---|---|
| 1 | <¬a> | **0.74** |
| 2 | <¬b> | **0.66** |
| 3 | <¬d> | **0.5** |

(o)

Fig. 5: Generating Seed Set.

## C. The Third Optimization: Pruning Strategy

From the above description we can see that, if we want to get *k* useful NSPs, we must calculate all $iwsup(nsc)$. This is very time consuming. To solve this problem, we propose a pruning strategy where only part of $iwsup(nsc)$ needs to be

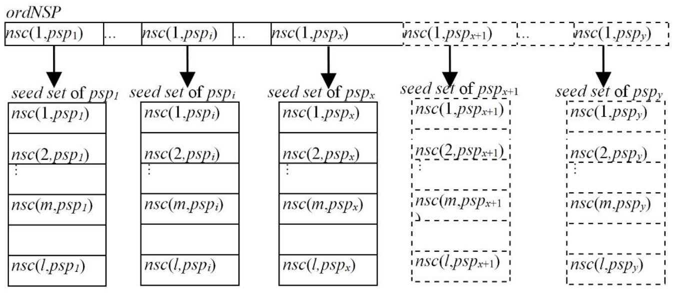


Fig. 6: Schematic Diagram of Pruning Seed Sets(1).

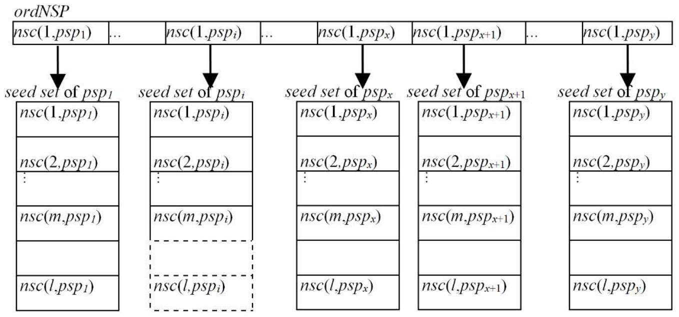


Fig. 7: Schematic Diagram of Pruning Seed Sets(2).

calculated. The strategy is divided into two parts, detailed as follows.

The first part is to build a seed set. After obtaining top-*k* PSPs, to each $psp(size(psp) > 1)$, we generate all its NSCs and put them in a set. Suppose the number of NSCs is *l*, the set is devoted by $\{nsc(1, psp_i), nsc(2, psp_i), \ldots, nsc(m, psp_i), \ldots, nsc(l, psp_i)\}$, where $sup(nsc(1, psp_i)) > sup(nsc(2, psp_i)) > \ldots > sup(nsc(m, psp_i)) > \ldots > sup(nsc(l, psp_i))$. This set is called a seed set.

In order to prune the seed sets, we must find the key code of each seed set. Suppose the number of PSPs $(size(psp) > 1)$ is *y*. In Fig.4, the *ordNSP* presents key codes and The arrow points to their corresponding seed set. The $nsc(1, psp_i)(0 \leqslant i \leqslant y)$ can identify a unique seed set. So, we set the ordered NSP set(denoted by *ordNSP*) that contains all key codes $\{nsc(1, psp_1), nsc(1, psp_2), \ldots, nsc(1, psp_y)\}$ of the seed sets. In particular, *ordNSP* satisfies $iwsup(nsc(1, psp_1)) > iwsup(nsc(1, psp_2)) \ldots > iwsup(nsc(1, psp_y))$ and $size(psp) > 1 (psp \epsilon \{psp_1, \ldots, psp_y\})$.

The schematic diagram of the seed sets is shown in Fig.4.

In particular, if $size(psp) = 1$, we calculate the $iwsup(nsc)$ of its NSC and add it into a useful NSP set (denoted by *useNSP*). The NSCs in the *useNSP* are sorted by $iwsup(nsc)$ in descending order. Suppose $w_P$=0.5, $w_N$=0.5 and *k*=10, Fig.5 shows the generation of the seed sets. Fig.5(c) expresses the situation of $size(psp) = 1$.

In Fig.5, (m) presents top-k PSPs, (n) presents seed sets and (0) presents 1-size useNSP. From (m) to (n), for each $psp(size(psp) > 1)$, we generate all its NSCs and put them in the corresponding seed set. The NSCs in the seed set are sorted by *sup(nsc)* in descending order. Now we take $<ab>$

in Fig.5 (m) for example to explain the process (m) to (n). It is all NSCs $< \neg ab >$ and $< a\neg b >$. The support of $< \neg ab >$ and $< a\neg b >$ are 0.4 and 0.2 respectively, so the seed set of $< ab >$ is $< \neg ab >, < a\neg b >$. Fig.5(o) is the *useNSP* that contains the NSCs corresponding to PSPs $(size(psp) = 1)$.

The second is to prune the seed set.

**Step 1:** calculate the *iwsup* of the first $nsc_1$ for each seed set, and then add them to *ordNSP* and *useNSP* simultaneously; The *useNSP* is denoted by $\{nsp_1, nsp_2, \ldots, nsp_k\}$, where $sup(nsp_1) > sup(nsp_2) > \ldots > sup(nsp_k)$.

**Step 2:** start from the seed set corresponding to the first *nsc* of *ordNSP*.

**Step 3:** compare $iwsup(nsc_2)$ with $iwsup(nsp_k)$ of *useNSP*. If $iwsup(nsc_2) > iwsup(nsp_k)$, delete the $nsp_k$ and add the $nsc_2$ to *useNSP*, then delete all the *nsc* of *ordNSP* that satisfy $iwsup(nsc) \leqslant iwsup(nsp_k)$. The schematic diagram of this situation is shown in Fig.6. If this seed set is not empty, calculate the $iwsup(nsc_3)$ and repeat **Step 3**. If $iwsup(nsc_2) \leqslant iwsup(nsp_k)$ or this seed set is empty, start from a new seed set that corresponds to the next *nsc* of *ordNSP* and repeat **Step 3**. Fig.7 shows the schematic diagram of the situation $(iwsup(nsc) \leqslant iwsup(nsp_2))$.

Fig.6 and Fig.7 show two cases of pruning, where the dashed line presents the pruned parts. Fig.6 shows that $iwsup(nsc(m, psp_i)) > iwsup(nsp_k)$ $(m \geqslant 2, nsp_k \in$the useful NSP set), $iwsup(nsc(1, psp_{x-1})) > iwsup(nsp_k))$ and $iwsup(nsc(1, psp_x)) \leqslant iwsup(nsp_k)$. We prune the seed sets of which the key codes are $nsc(1, psp_{x+1})...nsc(1, psp_y)$ by deleting $\{nsc(1, psp_{x+1})...nsc(1, psp_y)\}$ from *ordNSP*, then delete $nsc(n, psp_k)$ from *useNSP* (deleting is represented by a dashed line), and add the $nsc(m, psp_i)$ to *useNSP*.

Fig.7 shows that $iwsup(nsc(m, psp_i)) \leqslant iwsup(nspk)$ $(m \geqslant 2, nsp_k \in$ the useful NSP set), we do not calculate the *iwsup* of the other NSCs in the seed set of $psp_i$ (this part is represented by a dashed line)and find the next seed set that has the $nsc(1, psp_{i+1})$ of *ordNSP*.

**Step 4:** end until the last one of *ordNSP* has been traversed.

Fig.8 shows the pruning seed set. Fig.8(p) illustrates **Step 1**. Fig.8(q) and Fig.8(r) illustrate **Step 2** and **Step 3**.

## D. Topk-NSP$^+$ Algorithm analysis

Algorithm 2. Topk-NSP$^+$ Algorithm.

**Input:** Sequence dataset *D* and Parameters w$_P$, w$_N$, *k*;

**Output:** useNSP: a set that contains *k* useful negative sequential patterns, where

the sequences are arranged in descending order of *iwsup*;

```
(1) top-kPSP=$\varnothing$, NSC=$\varnothing$, ordNSP=$\varnothing$, useNSP=$\varnothing$,
      Seedset=$\varnothing$;
(2) top-kPSP=minePSP();
(3) For (each psp in top-kPSP){
(4) Generating NSC by using NSCgeneration from psp;
    $//\forall(nsc_i \epsilon$\{NSC\}, $sup(nsc_i) \geqslant sup(nsc_{i-1}))$;
(5)    If($size(psp) = 1$){
(6)        iwsup(nsc) is calculated by equation (7);
(7)        Insert {$nsc$} to {useNSP};
      $//\forall nsp_i \epsilon$\{useNSP\}$, iwsup(nsp_i) \geqslant iwsup(nsp_{i-1})$;
(8)    }
(9)    else{
(10)           Insert {NSC} to {Seedset};
(11)    }
(12) }
(13) For(each seedset in SeedSet){
(14)    For(each $nsc_1$ in {NSC}){
(15)        iwsup($nsc_1$) is calculated by equation (7);
(16)        Insert {$nsc_1$} to {ordNSP};
      $//\forall nsc_i \epsilon$\{ordNSP\}$, iwsup(nsc_i) \geqslant iwsup(nsc_{i-1})$;
(17)        Insert {$nsc_1$} to useNSP;
(18) }   }
(19) For(each nsc in ordNSP){
(20)    For(each {NSC} in SeedSet){
(21)        if($iwsup(nsc) > nsp_k$){
(22)            Delete {$nsp_k$)} from {useNSP};
(23)            Insert {$nsc$} to {useNSP};
(24)            Delete {$nsc$} from {ordNSP};
(25)            }
(26)        }
(27)    };
(28) return useNSP;
```

1) Line 2 is the same as section 4.3.

2) Line 3 to line 12, we generate corresponding NSC seed sets for each *psp*.

3) Line 5 to line 8, we calculate the $iwsup$ of NSCs correspond to PSP $size(psp) = 1$ and add it into $useNSP$.

4) Line 9 to line 11, to each $psp(size(psp) > 1)$, we generate all its NSCs and put them in a seed set $\{nsc_1, nsc_2, \ldots nsc_y\}$, where $sup(nsc_1) > sup(nsc_2) > \ldots > sup(nsc_y)$.

5) Line 13 to line 18, calculate the $iwsup$ of the first $nsc_1$ for each seed set, and then add them to $ordNSP$ and $useNSP$ simultaneously.

6) Line 19 to line 27, we prune the seed sets. We start from the seed set corresponding to the first $nsc$ of $ordNSP$, and calculate $iwsup(nsc_2)$. Then we compare $iwsup(nsc_2)$ with $iwsup(nsp_k)$ of $useNSP$ to prune the seed sets.

7) Line 28 returns the results and ends the algorithm.

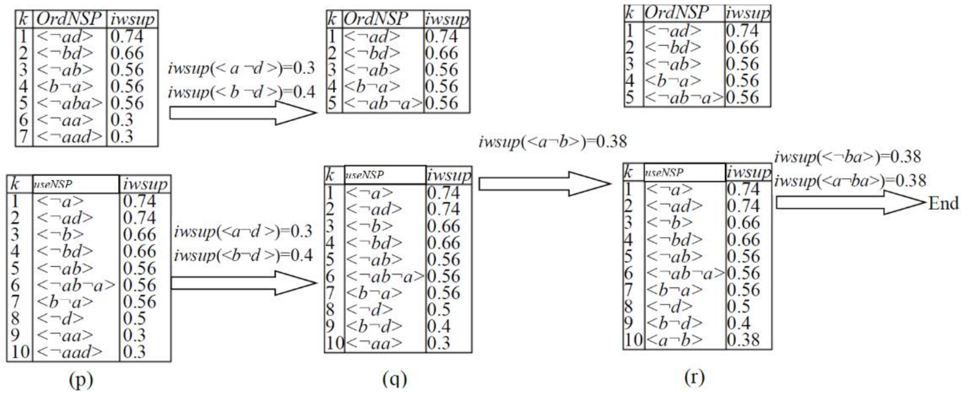
The core idea of Topk-NSP$^+$ algorithm is to mine top-*k* useful NSP from top-*k* PSP. It is to discover a set *useNSP* containing *k* NSP in a sequence database D such that for each pattern *ns*∈*useNSP* and *p(ns)*∈*top-k PSP*, there does not exist an NSP $ns' \notin useNSP$ and $p(ns') \in top-kPSP|iwsup(ns') > iwsup(ns)$. The definition of top-*k* PSP has been specifically given in the IV section.

From line 3 to line 12, Topk-NSP$^+$ algorithm first generates NSC by using *NSCgeneration* method.

*NSCgeneration*: For a *k*-size PSP, its NSC are generated by changing any *m* non-contiguous elements to their negative elements, *m* =1, 2, ...,$\lceil k/2 \rceil$, where $\lceil k/2 \rceil$ is a minimum integer that is not less than $k/2$.

Secondly, if $size(psp) = 1$, Topk-NSP$^+$ algorithm calculates the $iwsup$ of NSC and add it into $useNSP$; otherwise, Topk-NSP$^+$ algorithm builds the seed set and adds NSC to it.

***Definition 8. seed set.*** A seed set is a NSC set from which all NSC are generated from a $psp(size(psp) > 1)$ in top-*k* PSPs. Suppose the number of NSC is *l*, seed set is devoted by



| k | OrdNSP | iwsup |
|---|---|---|
| 1 | <¬ad> | 0.74 |
| 2 | <¬bd> | 0.66 |
| 3 | <¬ab> | 0.56 |
| 4 | <b¬a> | 0.56 |
| 5 | <¬aba> | 0.56 |
| 6 | <¬aa> | 0.3 |
| 7 | <¬aad> | 0.3 |

| k | OrdNSP | iwsup |
|---|---|---|
| 1 | <¬ad> | 0.74 |
| 2 | <¬bd> | 0.66 |
| 3 | <¬ab> | 0.56 |
| 4 | <b¬a> | 0.56 |
| 5 | <¬ab¬a> | 0.56 |

| k | OrdNSP | iwsup |
|---|---|---|
| 1 | <¬ad> | 0.74 |
| 2 | <¬bd> | 0.66 |
| 3 | <¬ab> | 0.56 |
| 4 | <b¬a> | 0.56 |
| 5 | <¬ab¬a> | 0.56 |

| k | useNSP | iwsup |
|---|---|---|
| 1 | <¬a> | 0.74 |
| 2 | <¬ad> | 0.74 |
| 3 | <¬b> | 0.66 |
| 4 | <¬bd> | 0.66 |
| 5 | <¬ab> | 0.56 |
| 6 | <¬ab¬a> | 0.56 |
| 7 | <b¬a> | 0.56 |
| 8 | <¬d> | 0.5 |
| 9 | <¬aa> | 0.3 |
| 10 | <¬aad> | 0.3 |

| k | useNSP | iwsup |
|---|---|---|
| 1 | <¬a> | 0.74 |
| 2 | <¬ad> | 0.74 |
| 3 | <¬b> | 0.66 |
| 4 | <¬bd> | 0.66 |
| 5 | <¬ab> | 0.56 |
| 6 | <¬ab¬a> | 0.56 |
| 7 | <b¬a> | 0.56 |
| 8 | <¬d> | 0.5 |
| 9 | <b¬d> | 0.4 |
| 10 | <¬aa> | 0.3 |

| k | useNSP | iwsup |
|---|---|---|
| 1 | <¬a> | 0.74 |
| 2 | <¬ad> | 0.74 |
| 3 | <¬b> | 0.66 |
| 4 | <¬bd> | 0.66 |
| 5 | <¬ab> | 0.56 |
| 6 | <¬ab¬a> | 0.56 |
| 7 | <b¬a> | 0.56 |
| 8 | <¬d> | 0.5 |
| 9 | <b¬d> | 0.4 |
| 10 | <a¬b> | 0.38 |

Fig. 8: pruning Seed Set.

$\{nsc(1, psp_i), nsc(2, psp_i), \ldots, nsc(m, psp_i), \ldots, nsc(l, psp_i)\}$, where $sup(nsc(1, psp_i)) > sup(nsc(2, psp_i)) > \ldots > sup(nsc(m, psp_i)) > \ldots > sup(nsc(l, psp_i))$.

From line 13 to line 18, Topk-NSP$^+$ algorithm build ordNSP that contains the key code of each seed set to prune seed sets. At the same time, the earliest *k* NSPs in useNSP are generated. And From Line 19 to line 28, Topk-NSP$^+$ prunes seed sets. The process is described in the Third Optimization. In particular, the support and *iwsup* of NSCs are calculated by equations (1), (2), (3) and (7). Among them, equations (1), (2) and (3) give a rigorous proof in the e-NSP algorithm. Equation (7) is explanation in the Second Optimization.

The relationship between Topk-NSP and Topk-NSP$^+$ is as follows. Topk-NSP is only used to mine top-k NSPs, doesn't consider its efficiency and whether the mined NSPs are useful or not. Topk-NSP$^+$ adds three optimizations to solve these problems. The first two optimizations let Topk-NSP$^+$ mine more useful NSPs than Topk-NSP. The third one let Topk-NSP$^+$ be more time efficient than Topk-NSP.

### E. Theoretical Analysis

So far we have not found an algorithm for mining top-*k* N-SPs. The same two algorithms Topk-NSP and Topk-NSP$^+$ are used for mining PSPs, generating NSCs and calculating NSCs' support. The difference is that the Topk-NSP$^+$ algorithm uses a pruning strategy when selecting useful NSPs. Therefore, we theoretically analyse the ability of the two algorithms to select useful NSPs. Let $|NSC_{c,m}|$ denote the number of *m*-neg-size NSCs generated from a *c*-size PSP. According to definition of e-NSP candidate generation and the related properties of permutations and combinations, $|NSC_{c,m}|$ can be recognized as the combinations of taking *m* elements from $(c - m + 1)$ elements and can be calculated by equation (8).

$$|NSC_{c,m}| = C^{m}_{(c-m+1)} = \frac{(c-m+1)!}{m! * (c-2m+1)!}(1 \leqslant m \leqslant \lceil c/2 \rceil) \tag{8}$$

The number of NSCs from the top-*k* PSPs is as follows:

$$k * |NSC_{\bar{c},\forall m}| = k * \sum_{m=1}^{\lceil \bar{c}/2 \rceil} |NSC_{\bar{c},m}| \tag{9}$$

where $\bar{c}$ represents the average number of elements contained in PSPs. In this experiment, we use $t_{insert}$ to express the time it takes to insert an NSC into the top-kNSP. Therefore, the total time spent by the Topk-NSP algorithm can be expressed as

$$T^{Topk-NSP} = k * \sum_{m=1}^{\lceil \bar{c}/2 \rceil} |NSC_{\bar{c},m}| * t_{insert} \tag{10}$$

We did not analyze the time to generate the seed sets because the average length of the seed sets is $\bar{c}$, and $\bar{c}$ is negligible compared to *k*. The total time spent by Topk-NSP$^+$ algorithm can be expressed as

$$T^{Topk-NSP^+} = \alpha * k * \sum_{m=1}^{\lceil \bar{c}/2 \rceil} |NSC_{\bar{c},m}| * t_{insert} \tag{11}$$

The ratio of equation (10) to (11) is $\frac{1}{\alpha}$, where $\alpha$ represents the ratio of the number of NSCs after pruning to the total number of NSC $\alpha < 1$. Its value is influenced by parameters $k, w_P, w_N$, and the datasets.

The algorithm only mines the top-*k* NSPs from the top-*k* PSPs without scanning the database. So, the runtime is mainly influenced by *k*, independent of the size of the datasets. The size of the datasets only affects the mining of the top-*k* PSPs, which is not the focus of this paper. Therefore, this paper does not involve too datasets which are too large.

## VI. Experimental Results

In order to test the proposed methods, five experiments are undertaken.

1) Test the efficiency of the Topk-NSP algorithm.
2) Test the effectiveness of the weighted support.
3) Test the effectiveness of the interestingness metric.
4) Test the efficiency of the Topk-NSP$^+$ algorithm.
5) Test the scalability of the Topk-NSP$^+$ algorithm.

From section V, we can see that *wsup* and *iwsup* can influence the experimental results, but this influence is only reflected in that the mined NSPs are different (we have described these differences in Fig.2 and Fig.3.) and they are difficult to express well through experiments because the number of mined NSPs are the same. So we do not conduct experiments on (2) and (3). In subsection B, we describe the experiments on (1). In subsection C, we describe the experiments on (4). The scalability test of the Topk-NSP$^+$ algorithm is described in subsection D.

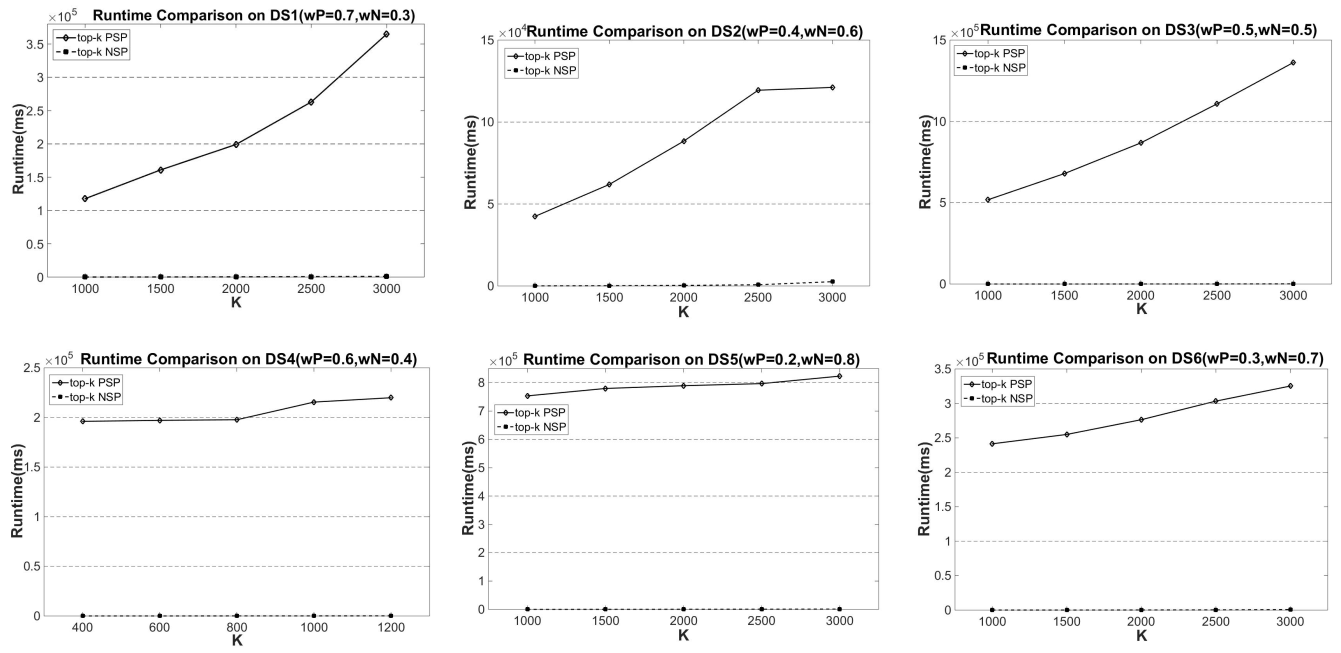


Fig. 9: Runtime of Topk-NSP on datasets DS1–DS6.

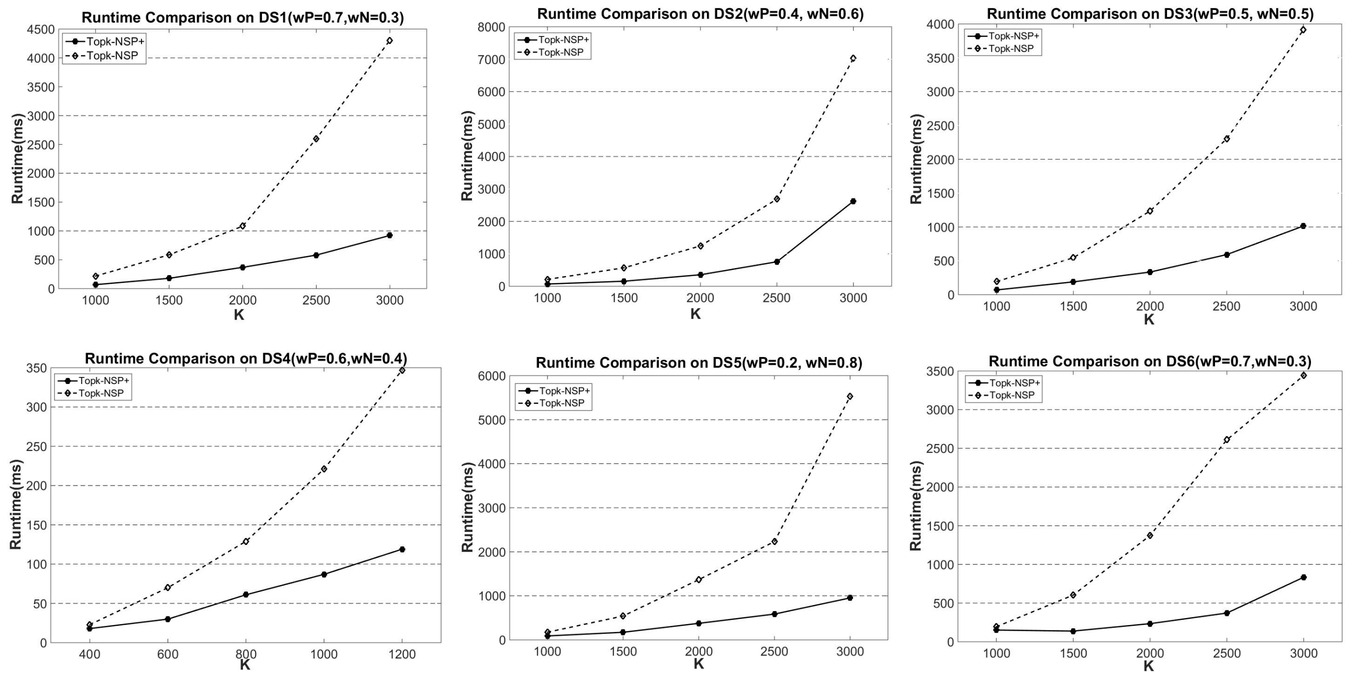


Fig. 10: Runtime of Topk-NSP$^{+}$ on datasets DS1–DS6.

TABLE I: Summary of datasets

| Dataset | sequence Numbers | distinct item Numbers | file size |
|---|---|---|---|
| DS1 | 100K | 100 | 137.9K |
| DS2 | 10K | 100 | 5.7K |
| DS3 | 100K | 200 | 8.26K |
| DS4 | 20,450 | 17 | 2.6M |
| DS5 | 59,601 | 497 | 0.8M |
| DS6 | 5,269 | Around 4K | 5.1M |

We conduct experiments on four synthetic and two real-life datasets to compare the efficiency of the two algorithms. The synthetic datasets generated by IBM and the real datasets were used for the e-NSP algorithm [6]. All the algorithms are implemented in Eclipse, running on Windows 10 PC with 32GB memory, Inter Core i7 3.4GHz CPU and all the programs are written in Java. In the experiments, $sup$ and $ms$ are expressed in terms of the percentage of the frequency $|\{< s >\}|$ compared to the number of sequences $|D|$ in the database, i.e., $|\{< s >\}|/|D|$.

## A. Datasets

Definition 9. Data Factor. A data factor describes the characteristic of underlying data from a particular perspective. We specify the following data factors: C, T, S, I, DB and N to describe the characteristics of sequential data.

C: Average number of elements per sequence;

T: Average number of items per element;

S: Average length of potentially maximal sequences;

I: Average size of items per element in potentially maximal large sequences;

DB: Number of sequences in a database;

N: Number of items.

Dataset 1 (DS1) is C12.T6.S10.I8.DB100k.N100.

Dataset 2 (DS2) is C8.T4.S8.I8.DB10k.N100.

Dataset 3 (DS3) is C10.T4.S8.I12.DB100k.N200.

Dataset 4 (DS4) is a dataset of 20,450 sequences of click stream data from the FIFA World Cup 98 website. It has 2990 distinct items (web pages). The average sequence length is 34.74 items with a standard deviation of 24.08 items. This dataset was created by processing a section of the World Cup web log.

Dataset 5 (DS5) is a KDD-CUP 2000 dataset which contains 59,601 sequences of e-commerce click streams. It contains 497 distinct items. The average length of each sequence is 2.42 items with a standard deviation of 3.22. The dataset contains some long sequences, for example, 318 sequences contain more than 20 items.

Dataset 6 (DS6) is C12.T8.S10.I12.DB10k.N300.

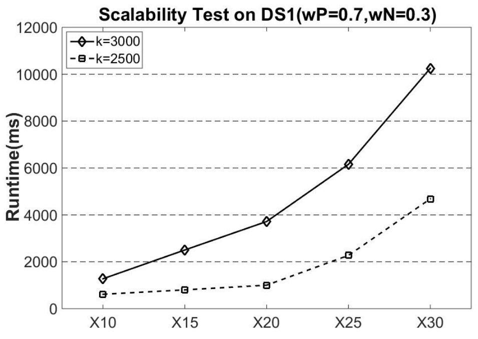
Table I summarizes the characteristics of all of the datasets.

## B. The Efficiency of Topk-NSP

In this experiment, we use the same $k$ on the top-$k$ PSPs and top-$k$ NSPs. The execution time of mining the top-$k$ PSPs and top-$k$ NSPs is shown in Fig.9. From Fig.9, we can see that mining the top-$k$ NSPs always takes much less time than the top-$k$ PSPs on all datasets. For example, mining the top-$k$ NSPs costs 0.06 % to 0.26% of mining the top-$k$ PSPs runtime on DS6 when $k$ increases from 1000 to 3000. In general, mining the top-$k$ NSPs only takes 0.01-3% of the runtime of mining the top-$k$ PSPs on all datasets DS1 to DS6. This is because we use equations to calculate the supports of NSCs and do not need to re-scan the database. Compared with scanning databases, using equations to calculate support takes little time.

## C. The Efficiency of Topk-NSP$^+$

In Fig.10, we set different $w_P$ and $w_N$ for DS1 to DS6. This is because different $w_P$ and $w_N$ can obtain the different result. The point can prove in the section V. From Fig.10, we can see that with the increase of *k*, Topk-NSP$^+$ takes less time than Topk-NSP. This is consistent with our theoretical analysis. Although the Topk-NSP$^+$ needs to order the seed set, the length of each seed set is negligible compared to *k*. Topk-NSP$^+$ can effectively prune seed sets and avoid many unnecessary operations. Thus the time efficiency of the algorithm is improved.



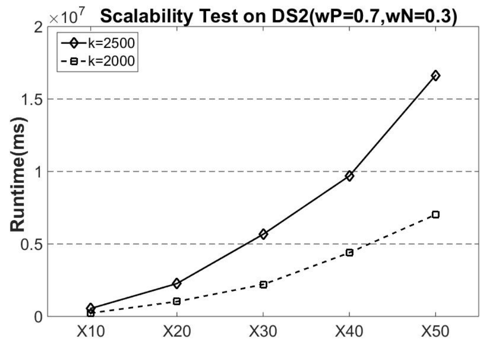


Fig. 11: Scalability test on datasets DS1 and DS2.

## D. Scalability and Memory Test

Topk-NSP$^+$ calculates the support of NSC based on the sid sets of the corresponding positive patterns. Thus, its performance is sensitive to the size of the sid sets. If a dataset is huge, it produces large sid sets. A scalability test is conducted to evaluate Topk-NSP$^+$'s performance on large datasets. Fig.11 shows the results of Topk-NSP$^+$ on datasets DS1, DS2 and in terms of different data sizes: from 5 (i.e., 73M) to 25 (227M) times of DS1, from 10 (26M) to 50(130M) times of DS2, with various $k$ 2500 and 3000 on DS1, as well as 2000 and 2500 on DS2, respectively.

On DS2, for example, when the sampled data size increases to 50 times its original size (see the results corresponding to label X50), the Topk-NSP$^+$ takes 117 seconds to obtain the results. This is around thirty times of the runtime on the 10 times (X10) data size. This indicates that the increase of 5 times the data size leads to about 30 times runtime growth.

Fig.12 shows the memory occupied by the Topk-NSP$^+$ when it runs on DS1 and DS2 and in terms of different data sizes: from 5 (i.e., 73M) to 25 (227M) times of DS1, from 10 (26M) to 50(130M) times of DS2, with various $k$ 2500 and 3000 on DS1, as well as 2000 and 2500 on DS2, respectively. On DS1, the increase of 5 times the data size leads to about 2.3 times memory growth. On DS2, the increase of 5 times the data size leads to about 2 times memory growth. This shows that with the increase of data sets, the memory consumption increase little.

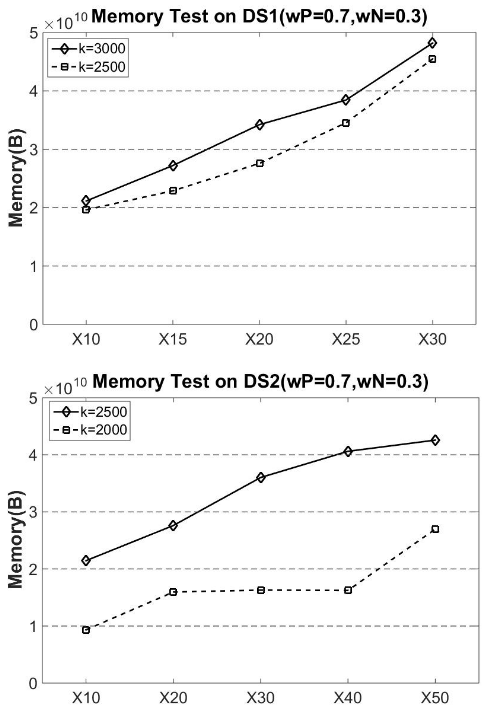


Fig. 12: Memory test on datasets DS1 and DS2.

## VII. CONCLUSIONS

NSP is an important tool for understanding complex NOB. Mining NSP is very challenging due to the problems: (1) how to mine an expected number of patterns, (2) how to select useful NSP, and (3) how to reduce the high time consumption. However, NSP discovery is becoming increasingly important for many intelligent systems and applications, as traditional PSP and association rule mining approaches cannot effectively detect such patterns and exceptions that are associated with non-occurring sequences. NSP mining has achieved very limited research outcomes and most of existing methods focus on how to design a mining algorithm and how to improve the algorithm's efficiency. In this paper, we have studied the above problems and proposed a method Topk-NSP to mine top-$k$ NSP. Next, we have proposed three optimization strategies to Topk-NSP. First, we have proposed a weighted support method to select useful NSP. Second, we have proposed an interestingness metric optimization strategy to select more useful NSP. Finally, we have proposed a pruning strategy to reduce the high time consumption of Topk-NSP. At last, we obtained an optimizing algorithm Topk-NSP$^+$. The experimental results on real-life and synthetic datasets show that Topk-NSP$^+$ is more effective in runtime.

Topk-NSP$^+$ is mainly used in behavioral analysis such as shopping behavior analysis, insurance behavior analysis and software users behavior analysis, etc. it not only avoids setting *ms*, but also considers the influence of PSPs on NSPs mining. To our best knowledge, it is the first algorithm to mine the top-$k$ NSPs and consider the influence of PSPs on NSPs mining.

Our future work is to find a more effective method to mine useful NSPs not only from top-$k$ PSPs. In addition, in pattern mining, it is an open issue to verify the correctness and completeness of patterns discovered by a pattern mining algorithm. We will explore this further in the NSP research.

## REFERENCES

[1] L.B. Cao, "*In-depth behavior understanding and use: the behavior informatics approach*," Information Science vol. 180, no. 17, pp. 3067-3085, 2010.
[2] L.B. Cao, P.S. Yu, "*Behavior computing: modeling, Analysis, Mining and Decision*," Springer Publishing Company, Incorporated, 2012.
[3] L.B. Cao, P.S. Yu, V Kumar, "*Nonoccurring behavior analytics: a new area*," IEEE Intell. Syst. vol. 30, no. 6, pp. 4-11, 2015.
[4] L.B. Cao, Y. Ou, P.S. Yu, "*Coupled behavior analysis with applications*," IEEE Trans. Knowl. Data Eng. vol. 24, no. 8, pp. 1378-1392, 2012.
[5] M.J. Zaki, "*Spade: an efficient algorithm for mining frequent sequences*," Mach. Learn, vol. 42, no. 1, pp. 31-60, 2001.
[6] L.B. Cao, X.J. Dong, Z.G. Zheng, "*e-NSP: efficient negative sequential pattern mining*," Artificial Intelligence, vol.235, pp.156-182, 2016.
[7] X.J. Dong, C.L. Liu, T.T. Xu, D.K. Wang, "*Select actionable positive or negative sequential patterns*," Journal of Intelligent and Fuzzy Systems, vol.29, no. 6, pp. 2759-2767, 2015.
[8] C.L. Liu, X.J. Dong, C.Y. Li, L. Li, "*SAPNSP: select actionable positive and negative sequential patterns based on a contribution metric*," International Conference on Fuzzy Systems and Knowledge Discovery. IEEE, 2016.
[9] Y.S. Gong, C.L. Liu, X.J. Dong, "*Research on typical algorithms in negative sequential pattern mining*," Open Automation and Control Systems Journal, vol. 7, no. 1, pp. 934-941, 2015.
[10] Y.S. Gong, T.T. Xu, X.J. Dong, G.H. Lv, "*e-NSPFI: efficient mining negative sequential pattern from both frequent and infrequent positive sequential patterns*," International Journal of Pattern Recognition and Artificial Intelligence, vol. 31, no. 2, pp. 3-14, 2016.
[11] T.T. Xu, X.J. Dong, J.L. Xu, Y.S. Gong, "*E-msNSP: efficient negative sequential patterns mining based on multiple minimum supports*. International Journal of Pattern Recognition and Artificial Intelligence,vol. 31, no. 2,pp. 3-14, 2016.
[12] S.C. Hsueh, M.Y. Lin, C.L. Chen, "*Mining negative sequential patterns for e-commerce recommendations*," IEEE Asia-Pacific Services Computing Conference, IEEE Computer Society, 2008.
[13] V.K. Khare, V. Rastogi, "*Mining positive and negative sequential pattern in incremental transaction databases*," Int. J. Comput. Appl. vol. 77, no. 1, pp. 18-22, 2013.
[14] N.P. Lin, H.J. Chen, W.H. Hao, "*Mining negative sequential patterns*," Conference on Wseas International Conference on Applied Computer Science. World Scientific and Engineering Academy and Society (WSEAS), 2007.
[15] Z.G. Zheng, Y.C. Zhao, Z.Y. Zuo, L.B. Cao, "*Negative-GSP: an efficient method for mining negative sequential patterns*," Eighth Australasian Data Mining Conference. Australian Computer Society, Inc.2009.

[16] Z.G. Zheng, Y.C. Zhao, Z.Y. Zuo, L.B. Cao, "*An efficient GA-based algorithm for mining negative sequential patterns*," Advances in Knowledge Discovery and Data Mining, Pacific-Asia Conference, Hyderabad, India, 2010.

[17] Z.G. Zheng, "*Negative sequential pattern mining*," PhD Thesis, 2011.

[18] P. Tzvetkov, X.F. Yan, J.W. Han, "*TSP: mining top-k closed sequential patterns*,". Knowledge and Information Systems, vol. 7, no. 4, pp. 438-457, 2005.

[19] P. Fournier-Viger, A. Gomariz, T. Gueniche, E. Mwamikazi, R. Thomas, "*TKS: efficient mining of top-k sequential patterns*," International Conference on Advanced Data Mining and Applications. Springer Berlin Heidelberg, 2013.

[20] K.B. Hathi, J.R. Ambasana, "*Top k sequential pattern mining algorithm*," Iciems, 2015.

[21] F. Petitjean, T. Li, N. Tatti, G.I. Webb, "*Skopus: mining top- k, sequential patterns under leverage*," Data Mining and Knowledge Discovery, vol. 30, no. 5, pp. 438-457, 2016.

[22] S. Sakurai, M. Nishizawa, "*A new approach for discovering top-k sequential patterns based on the variety of items*," Journal of Artificial Intelligence and Soft Computing Research, vol. 5, no. 2, pp. 141-153, 2015.

[23] C. Gao, L. Duan, G.Z. Dong, H.Q. Zhang, H. Yang, C.J. Tang, "*Mining top-k distinguishing sequential patterns with flexible gap constraints*," 17th International Conference on Web-Age Information Management, Nanchang, China, Springer Verlag, 2016.

[24] H.Yang, L. Duan, B. Hu, S. Deng, W.T. Wang, P. Qin, " *Mining top-k distinguishing sequential patterns with gap constraint*," J. Softw, vol. 26, no. 11, pp. 2994-3009, 2015.

[25] J.F. Yin, Z.G. Zheng, L.B. Cao, Y. Song, W. Wei, "*Efficiently mining top-k high utility sequential patterns*," 13th IEEE International Conference on Data Mining, Dallas, TX, United states, Institute of Electrical and Electronics Engineers Inc., 2013.

[26] J. Pei, J. Han, B. Mortazavi-Asl, J.Y. Wang, H. Pinto, "*Mining sequential patterns by pattern-growth: the prefixSpan approach*," IEEE Trans. Knowledge and Data Engineering, vol. 16, no. 10, pp. 1-17, 2001.

[27] J. Ayres, J. Flannick, J. Gehrke, T. Yiu, "*Sequential PAttern mining using a bitmap representation*," Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2002.

[28] Y.F. Li, A. Algarni, N. Zhong, "*Mining positive and negative patterns for relevance feature discovery*," ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, Dc, Usa, July. DBLP, 2010.

[29] S. Kamepalli, R. Kurra, "*Frequent negative sequential patterns: a survey*," Int. J. Comput. Eng. Technol. vol. 5, no. 3, pp. 15-121, 2014.

[30] P. Kazienko, "*Mining sequential patterns with negative conclusions*," in: DaWaK2008.

[31] X.D. Wu, C.Q. Zhang, S.C. Zhang, "*Efficient mining of both positive and negative association rules*," ACM Trans Inf Syst. vol. 22, no. 3, pp. 381-405, 2004.

[32] C.L. Liu, G.H. Lv, X.J. Dong, H.N. Yuan, X. Dong, "*Selecting actionable patterns from positive and negative sequential patterns*," Journal of Residuals Science and Technology, vol. 14, no. 1, pp. 407-419, 2017.

[33] X.J. Dong, "*Mining interesting infrequent and frequent itemsets based on minimum correlation strength*," Artificial Intelligence and Computational Intelligence. Springer Berlin Heidelberg, 2011.

[34] J.W. Han, J.Y. Wang, Y. Lu, P. Tzvetkov, "*Mining top-k frequent closed patterns without minimum support*. IEEE International Conference on Data Mining. IEEE Computer Society, Maebashi, Japan, Dec. 2002.

[35] N.P. Lin, W.H. Hao, H.J. Chen, C.I. Chang, H.E. Chueh, "*An algorithm for mining strong negative fuzzy sequential patterns*. Braga North Atlantic University Union, vol. 3, No. 1, pp. 167-172, 2007.

[36] Y. Zhao, H. Zhang, L. Cao, C. Zhang, H. Bohlscheid, "*Efficient mining of event-oriented negative sequential rules*," in: WI-IAT2008.

[37] K. Kavitha, E. Ramaraj, "*Efficient transaction reduction in actionable pattern mining for high voluminous datasets based on bitmap and class labels*," International Journal on Computer Science and Engineering, vol. 5, no. 7, pp. 664-671, 2013.

[38] Y.X. Wu, Z.Q. Tang, H. Jiang, X.D. Wu, "*Approximate pattern matching with gap constraints*," Journal of Information Science, vol. 42, no. 5, pp. 639-658, 2016.

[39] K. Wang, Y.L. Jiang, A. Tuzhilin, "*Mining actionable patterns by role models*," International Conference on Data Engineering, IEEE Xplore, 2006.

[40] P. Kanikar, D.K. Shah, "*Extracting actionable association rules from multiple datasets*," International Journal of Engineering Research and Applications, vol. 2, pp. 1295-1300, 2012.

[41] U. Yun, J.J. Leggett, "*WSpan: weighted sequential pattern mining in large sequence databases*," In: Proceedings of the 3rd international ieee conference on intelligent systems, 2006.

[42] G.C. Lan, T.P. Hong, H.Y. Lee, "*An efficient approach for finding weighted sequential patterns from sequence databases*," Applied Intelligence, vol. 41, no. 2, pp. 439-452, 2014.

[43] O.J. Rasanen, J.P. Saarinen, "*Sequence prediction with sparse distributed hyper dimensional coding applied to the analysis of mobile phone use patterns*," IEEE Transactions on Neural Networks and Learning Systems, vol. 27, no. 9, pp. 1878-1889, 2015.

[44] C.F. Ahmed. "*An efficient distributed programming model for mining useful patterns in big datasets*," Iete Technical Review, vol.30, no.1, pp.53-63, 2013.

[45] S.S. Ho, P. Dai, F. Rudzicz, "*Manifold learning for multivariate variable-length sequences with an application to similarity search*," IEEE Transactions on Neural Networks and Learning Systems, vol. 99, pp. 1333-1344, 2015.

[46] J.H. Lv, X.M. Li, L.X. Ding, L.Z. Jiang, "*Applying principal component analysis and weighted support vector machine in building cooling load forecasting*," International Conference on Computer and Communication Technologies in Agriculture Engineering, 2010.

[47] L.X. Ding, J.H. Lv, X.M. Li, L.L. Li, "*Support vector regression and ant colony optimization for HVAC cooling load prediction*," International Symposium on Computer,communication, Control and Automation Proceedings. 2010.

[48] J.M. Luna, A. Cano, V. Sakalauskas, S. Ventura, "*Discovering useful patterns from multiple instance data*," Information Sciences, vol.357, no.C, pp.23-38, 2016.

[49] F.Z.E. Mazouri, M.C. Abounaima, K. Zenkouar. "*A selection of useful patterns based on multi-criteria analysis approach*," Iccwcs'17 Proceedings of the, International Conference on Computing and Wireless Communication Systems. 2018.

[50] X.J. Dong; Y.S. Gong; L.B. Cao, "e-RNSP: An efficient method for mining repetition negative sequential patterns," IEEE Transactions on Cybernetics, DOI: 10.1109/TCYB.2018.2869907.

**Xiangjun Dong** received the Ph.D. degree in Computer Applications from Beijing Institute of Technology, China, in 2005. From 2007 to 2009, he worked as a postdoctoral fellow in School of Management and Economics, Beijing Institute of Technology, China. From 2009 to 2010, he was a Visiting Fellow of University of Technology Sydney, Australia. He is currently a Professor of School of Information, Qilu University of Technology (Shandong Academy of Sciences) in Jinan, China. His research interests include data mining, artificial intelligence, machine learning. He has published around 70 journal/conference publications including Artificial Intelligence, Neurocomputing, Sensors, CIKM and IEEE Transactions on Neural Networks and Learning Systems, and IEEE Transactions on Cybernetics.

Dr. Dong was a PC member of PAKDD(2018201620112009), PRICAI(2016-2012), AusAI(2012-2008), ADMA(2016), ADMA(2009-2006), DMBiz(2007), and a member of ACM. He was a reviewer of international journals, including the IEEE Transactions on Knowledge Discovery and Engineering, the IEEE Intelligent Systems, and the Knowledge based Systems.

**Ping Qiu** received Master's Degree in the specialty of Computer Application Technology at Qilu University of Technology (Shandong Academy of Sciences)and she continue her position as a PhD student at computer science and technology of Beijing Institute of Technology.

She has published 1 academic paper and invented 3 patents (accepted). Her research interests cover the fields of data mining, pattern recognition, association rules, positive and negative sequential pattern mining.

Miss Qiu was honored include outstanding students, 3 school two-class scholarships and 4 quality development scholarships and 2 postgraduate scholarships.

**Jinhu Lü** (M03-SM06-F13) received the Ph.D degree in applied mathematics from the Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China, in 2002. He was a Professor with RMIT University, Melbourne, VIC, Australia, and a Visiting Fellow with Princeton University, Princeton, NJ, USA. He is currently a Professor with the School of Automation Science and Electrical Engineering, State Key Laboratory of Software Development Environment, Beijing Advanced Innovation Center for Big Data and Brain Computing, Beihang University and the Academy of Mathematics and Systems Science, Chinese Academy of Sciences. He is the Chief Scientist of National Key Research and Development Program of China and a Leading Scientist of Innovative Research Groups of National Natural Science Foundation of China. His current research interests include nonlinear circuits and systems, complex networks, multiagent systems, and big data.

Dr. Lü was a recipient of the Prestigious Ho Leung Ho Lee Foundation Award in 2015, the State Natural Science Award three times from the Chinese Government in 2008, 2012, and 2016, respectively, the Australian Research Council Future Fellowships Award in 2009, the National Natural Science Fund for Distinguished Young Scholars, and a Leading Scientist of Ten Thousand Talents Program of China. He is a Highly Cited Researcher in engineering in 2014, 2015, and 2016. He was an editor in various ranks for 15 SCI journals, including the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II, the IEEE TRANSACTIONS ON NEURAL NETWORKS, the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, the IEEE TRANSACTIONS ON BIOMEDICAL CIRCUITS AND SYSTEMS, and the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-H. He served as a member in the Fellows Evaluating Committees of IEEE Circuits and Systems Society, IEEE Industrial Electronics Society, and IEEE Computational Intelligence Society.

**Longbing Cao** was awarded one PhD in Pattern Recognition and Intelligent Systems in Chinese Academy of Sciences and another in Computing Science from the University of Technology Sydney (UTS). He has been a full professor in information technology at UTS since 2009, and the Founding Director of the university's research institute Advanced Analytics Institute in 2011. He was also the Research Leader of the Data Mining Program at the Australian Capital Markets Cooperative Research Centre. He joined UTS in 2005 after he served as an editor, marketing strategist, deputy director and then Chief Technology Officer in Beijing.

Before joining UTS, he had several years of research experience in Chinese Academy of Sciences, and working experiences in managing and leading industry and commercial projects in telecommunications, banking and publishing, as a manager or chief technology officer.He has published 3 monographs, 4 edited books and 16 proceedings, 11 book chapters, and around 200 journal/conference publications including IJCAI, KDD, ICDE, ICDM, AAMAS, WWW and IEEE Transactions in the above areas. His research interests include data science, behavior informatics, data mining, machine learning, agent mining, complex intelligent systems, and in particular the enterprise applications of deep data analytics and active customer, behavior and business management in the real world.

Dr. Cao is a Senior Member of IEEE, Computer and SMC Society, and a member of ACM. He is the Chair of ACM SIGKDD Australia and New Zealand Chapter, IEEE Task Force on Data Science and Advanced Analytics, and IEEE Task Force on Behavioral, Economic and Socio-cultural Computing. He serves as associate editor and guest editor on such journals as ACM Computing Surveys, and as general co-chair such as KDD2015.

**Tiantian Xu** received her B.E. and M.E. degree in computer applications from Qilu University of Technology in 2012 and 2015, respectively. She received her Ph.D. degree in software engineering from Ocean University of China in 2018. She is a lecturer in School of Information, Qilu University of Technology (Shandong Academy of Sciences). Her research interests include pattern recognition, association rules, sequential pattern mining. She has published research papers in international journals.