F-NSP$^+$: A Fast Negative Sequential Patterns Mining Method with Self-adaption Data Storage Strategy

Xiangjun Dong[a,*], Yongshun Gong[a], Longbing Cao[b,**]

[a] *Qilu University of Technology, Jinan, China*
[b] *University of Technology, Sydney, Australia*

Abstract

Mining Negative sequential patterns (NSP) is much more challenging than mining positive sequential patterns (PSP) due to the high computational complexity and huge search space when obtaining the support of negative sequential candidates (NSC). Very few NSP mining algorithms are available and most of them are very inefficient since they obtain the support of NSC by scanning database repeatedly. Instead, the state-of-the-art NSP mining algorithm e-NSP only uses PSP's information stored in an array structure to 'calculate' the support of NSC by equations, without database re-scanning. This makes e-NSP highly efficient, particularly on sparse datasets. However, when datasets becomes dense, the key process to obtain support of NSC in e-NSP becomes very time-consuming and needs to be improved. In this paper, we propose a novel and efficient data structure, *bitmap*, to obtain the support of NSC. We correspondingly propose a fast NSP mining algorithm, f-NSP, which uses *bitmap* to store the PSP's information and then obtain the support of NSC only by bitwise operations, which is much faster than the hash method in e-NSP. Experimental results on real-world and synthetic datasets show that f-NSP is not only tens to hundreds of times faster than e-NSP, but also saves more than dozens of times of storage spaces than e-NSP, particularly on dense datasets with a large number of elements in a sequence or a small number of itemsets. Further, we analyze that f-NSP will consume more storage space than e-NSP when PSP's support is less than a support threshold *sdsup*, a value by our theoretical analysis of storage space. Accordingly, we propose a self-adaptive storage strategy and a corresponding algorithm f-NSP$^+$ to overcome this deficiency. f-NSP$^+$ can automatically choose bitmap or array structure to store PSPs information according to the PSPs support. Experimental results show that f-NSP$^+$ saves more than dozens of times of storage space than f-NSP and has the similar time efficiency as f-NSP.

Keywords: , Sequential patterns, negative sequential patterns, bitmap

*Corresponding author 1
**Corresponding author 2
Email addresses: d-xj@163.com (Xiangjun Dong), longbing.cao@uts.edu.au (Longbing Cao)

1. Introduction

Negative sequential patterns (NSP) refer to sequences with non-occurring items. For instance, assume $p_1 =< abc\ X >$ is a PSP; $p_2 =< ab\neg c\ Y >$ is a NSP, where a, b and c stand for medical service codes that a patient has received in health care, and X and Y stand for disease status. p_1 shows that a patient who usually receives medical services a, b and then c is likely to have disease status X, whereas p_2 indicates that patients receiving treatments of a and b but NOT c have a high probability of having disease status Y [7].

It is increasingly recognized [8, 9, 40] that such NSP, composed of both occurring and non-occurring items, can play an irreplaceable role in deeply understanding and tackling some business applications, such as the associations between treatment services and illness, which cannot be handled by mining PSP only. NSP cannot be described by classic PSP mining algorithms such as GSP [2], SPAM [3], FreeSpan [4], SPADE [5], PrefixSpan [6] and also cannot be applied by traditional sequential pattern application algorithms[29-31]. Recent years have seen very limited progress in NSP mining [7-12], and most of the existing few NSP algorithms, such as PNSP [9] and NegGSP [8], are inefficient because it is much more difficult than PSP to discover PSPs especially due to the *intrinsic problem complexities* as NSP mining does not satisfy the Apriori principle [8], *high computational complexity* as most of existing methods [8-12] calculate the support of NSC by additionally scanning the database after identifying PSP, and *large negative sequential candidates (NSC) search space* when k-size NSC are generated by conducting a joining operation on (k-1)-size NSP [27].

e-NSP is an efficient NSP mining method [7, 27] that overcomes the above challenges towards (1) identifying NSP without database re-scanning, and (2) generating a limited number of truly useful NSC. It represents the state-of-the-art progress made in NSP mining.

e-NSP works as follows. Firstly, in e-NSP [27], a series of important concepts about NSP mining have been defined, including *negative containment*, which draws a clear boundary between whether a data sequence set contains a NSC. Secondly, e-NSP converts the negative containment problem to a positive containment problem. Finally, e-NSP 'calculates' the support of NSC based on the support of corresponding PSPs, without any additional database scans. In order to do so, e-NSP uses an array to store the PSPs' sequence ID; each PSP has a corresponding array. The array stores all sequence IDs of the sequences that contain this PSP in a dataset (see Table 2). Then e-NSP uses a hash method to obtain the union sets of relevant arrays, which is the key process that e-NSP 'calculates' the support of NSC by equations. In this way, there is no need to re-scan the database after discovering PSP. As a result, e-NSP is tens to thousands of times faster than PNSP and NegGSP, especially on sparse datasets with a small number of elements in a sequence, a large number of itemsets, and under low minimum supports [7].

However, when datasets becomes dense, the efficiency of e-NSP will be decreased and need to be improved, especially on the following two key complexities: time efficiency and space efficiency.

Time efficiency. The key process that e-NSP 'calculates' support of NSC is to obtain the union sets of the arrays. There are two key factors that affect the runtime of this process. One is the number of the union sets needed to calculate the support of NSC. The other is the number of sequence ID in the arrays.

The bigger the two numbers are, the more the runtime of this process is. The number of the union sets is directly proportional to the number of elements in a sequence, and the number of sequence ID in the arrays is directly proportional to the PSPs' support. With the number of elements in a sequence or the PSPs' support increasing, the time efficiency of e-NSP will be decreased. Hence, it is essential to develop even faster NSP mining methods to handle this circumstance.

Space efficiency. As introduced above, in order to 'calculate' the support of NSC, e-NSP needs to store every PSP's sequences ID (except the size of the PSP is one) in a array. When the support of PSP is large, the number of sequence ID in the arrays is large too. Meanwhile, if the minimum support is relevantly low, a larger number of PSPs will be generated, which would consume large amounts of storage space (see more discussions in Section 5). It is thus essential to develop efficient storage structures.

To address the above critical challenges and overcome the deficiency of e-NSP, in this paper, we further propose a novel and efficient data structure, *bitmap*, to store the PSP's information and obtain the support of NSC, and a corresponding fast algorithm, f-NSP. First, f-NSP creates a bitmap whose length is the number of sequences in dataset. Then, f-NSP fills in the bitmap with one ('1') or zero ('0'). If a sequence occurs in a data sequence, then f-NSP fill in the corresponding position of the sequence's bitmap with '1'; otherwise with '0' (see Table 2). This step can use any existing PSP mining algorithms (with slight changes if necessary). Finally, f-NSP obtains the support of NSC only by 'bitwise' operations, which is much faster than the hash method in e-NSP.

Experimental results on real-world and synthetic datasets show that f-NSP is not only tens to hundreds of times faster than e-NSP, but also saves more than dozens of times of storage space than e-NSP when the minimum support min_sup is more than a support threshold $sdsup$=0.03125, a value obtained by our theoretical analysis (see more discussions in Section 5). Experimental results also show that f-NSP works particularly well on dense datasets with a large number of elements in a sequence or a small number of itemsets or high support of PSP or large number of PSP. Therefore, f-NSP substantially improve the time and space efficiency of e-NSP. It is the proposed bitmap structure and the corresponding fast method to calculate the support of NSC that makes f-NSP highly time and space efficient simultaneously.

f-NSP, however, will consume more storage space than e-NSP when PSP's support is less than $sdsup$ (see more discussions in Section 5). To address this problem, we further propose a self-adaptive storage strategy and a corresponding algorithm f-NSP$^+$. f-NSP$^+$ can automatically choose bitmap or array to store PSP's information according to the PSP's support. That is, when PSPs support is more than or equal to $sdsup$, f-NSP$^+$ uses the bitmap structure as in f-NSP; otherwise, f-NSP$^+$ uses array as in e-NSP and automatically converts array to bitmap such that f-NSP$^+$ can use bitwise operations to fast calculate the support of NSC as in f-NSP. Experimental results show that f-NSP$^+$ can save more than dozens of times of storage space than f-NSP and has the similar time efficiency as f-NSP.

The rest of this paper is organized as follows. Section 2 discusses the related work. In Section 3, we formalize the problem of mining PSP and NSP. The proposed algorithm f-NSP is detailed in Section 4.

Section 5 proposes a space optimization algorithm of f-NSP: f-NSP$^+$. The experimental results are presented in Section 6. Section 7 reveals the conclusions and future work.

2. Related Work

Sequential patterns mining, which discovers frequent subsequences as patterns in a sequence database, is an significant data mining research problem with broad applications, and has been conducted on addressing many real behaviors, such as optimization strategy [41], classification and clustering problems [39, 42, 45, 48], Phenotype structure learning [47] and so forth. It is also deemed as an important strategy to solve many pattern recognition problems. For instance, sequential patterns mining can be used as the key method to predict Human Activity [54], as well as be implemented in biological sequences analysis [38, 44, 46]. Since the first proposal of sequential pattern mining in [1], many algorithms have been successfully proposed to enhance the algorithm efficiency. However, NSP mining still has huge challenges due to its own properties as discussed in Section 1. For the readability consideration, we first discuss some fundamental concepts of NSP mining. Further discussions concerning the NSP mining algorithms can be found in [27].

firstly, we discuss the basic concept of negative containment. Different researchers present inconsistent definitions and explanations. Hsueh et al. [9] consider that data sequence $ds =< dc >$ cannot contain negative sequence $ns =< \neg(ab)c\neg d >$ since $size(ns) > size(ds)$; while authors of [8] actually allow that ds contains ns.

Another issue is how to deal with a non-occurring element. Chen [9] argues that $ds =< dc >$ cannot contain $< \neg cd >$ because $< d >$ in ds has no antecedent itemset; ds cannot contain $< c\neg d >$ because $< c >$ in ds has no successor. However, Zheng [8] actually allows that ds contains them. Furthermore, the containment position of each element is very tricky. Chen [9] proposes that a data sequence $ds =< aacbc >$ cannot contain a negative sequence $ns =< a\neg bc >$, since the opposite evidence of $< abc >$ can be found in ds. however, Zheng [9] presents a divided opinion since it matches a and c; it finds the corresponding positive element in ds for each negative element of ns, such as the second a for $\neg b$. According to our understanding, since $< e >$ means that e occurs, there is no element (including element e) that occurs before or after e. Accordingly, $< e >$ contains $< e\neg? >$, $< \neg?\ e >$, $< \neg?e\neg? >$, where "?" represents any element. The above discussion shows that there is not a consolidated concept of negative containment in the literature. More discussions are in Section 4.2 and [27].

Secondly, unlike PSP mining, few methods are available in the literature about mining NSP. We briefly introduce them. In [8], a GSP-like way is introduced to mine for NSP, called NegGSP. It discovers PSP by GSP first, then generates and prunes NSC. It then counts the support of NSC by re-scanning the database to generate negative patterns. Chen et al. [9] designed a negative NSP mining approach PNSP for mining sequential patterns in the form of $< (abc)\neg(de)(ijk) >$. This approach proceeds in terms of three stages. First, PSP are mined by traditional algorithms and all positive itemsets are derived from these PSP. Second, all negative itemsets are derived from these positive itemsets. Finally, both positive and negative itemsets

cooperate generate NSC, which are in turn joined iteratively to generate longer NSC in an Apriori-like way. This approach calculates the support of NSC by scanning the database again.

Lin et al. only handle NSP with the last element as negative [10, 13, 14]. Ref. [10] proposes an algorithm NSPM for mining such NSP. The work in [13, 14] extends that in [10], which adds fuzzy and strong constraints to NSPM respectively. In [12], a genetic algorithm is proposed to mine NSP. It generates candidates by crossover and mutation by involving a dynamic fitness function to generate as many candidates as possible and avoiding population stagnation. Only the form of $(\neg A, B)$, $(A, \neg B)$ and $(\neg A, \neg B)$ are suitable for [15], which is similar to mine negative association rules [11, 26]. It generates frequent itemsets first, then generate frequent and infrequent sequences, and finally derive NSP from the infrequent sequences. The works in [16-18] are three extended version of [15] by adding fuzzy, multiple level, multiple minimum supports conditions respectively. The work in [19] mentions the question of mining NSP, but has not given a specific method of how to mine it. Ref. [20] mines NSP in the same form as [15] in incremental transaction databases. [51] aims to catch the influence (positive and negative interactions) of items in the time series databases, which could be recognized as the sequential patterns over time.

The authors of [21] propose an approach to mine event-oriented negative sequential rules from infrequent sequences in forms of $< A >\Rightarrow< \neg B >$, $< \neg A >\Rightarrow< B >$, $< \neg A >\Rightarrow< \neg B >$. Based on [21], [22] further presents an approach to discover both positive and negative impact-oriented sequential rules. Authors of [23, 24] discuss the questions about sequence classification by using positive and negative patterns. The method in [25] uses positive and negative usage patterns to filter Web recommendation lists. [49] finds conditional patterns considering positive and negative rules both. [50] introduces an algorithm for discovering two new patterns in multiple databases, i.e., synthesizing heavy association rules and exceptional rules. Hu et.al. propose a special binary partition tree called High-Yield Partition Tree to mine high-utility item sets [52]. And further negative rules mining approaches can be found in [34, 35, 36, 37, 53].

Different from the above work, e-NSP [7, 27] is an efficient NSP mining method without any additional database scans after mining PSP. It only uses PSP's information stored in an array to 'calculate' the support of NSC by equations. This makes e-NSP to obtain high time efficiency, particularly on sparse datasets. However, when datasets are dense, the efficiency of e-NSP decreases, as discussed in the following sections. As shown in [27], e-NSP is the most advanced method and has been built on an innovative data structure and working mechanism. This paper will utilize bitmap strategy to further enhance e-NSP for better time and space efficiency. Even though bitwise operations is a straightforward technique to facilitate the set operations, unlike other PSP or association rules mining methods using bitmap structure, such as SPAM [3], Index-BitTableFI [33] and dEclat [34], our proposed algorithms employ different bitmap structure and operation to catch hidden sequential patterns in a very short time span. For example, f-NSP creates bitmaps for PSPs whose sizes are greater than 1. If a PSP appears in a data sequence i, then the bitmap of this PSP is set to 1 in position i ; otherwise, the bit is set to 0. The length of each bitmap is equal to the number of sequences in the database. SPAM, however, is a PSP mining method by building a depth-first tree with

Table 1: Symbol Description

Symbol	Description
min_sup	Minimum support threshold
$sdsup$	Space division support
s	Sequence
ns	Negative sequence
$P(ns)$	ns's positive partner
$EidS_s$	Elements id set of sequence s
$OPS(EidS_s)$	Order preserving sequence with $EidS_s$
$MPS(s)$	Maximum positive sub-sequence of s
1-$negMS_{ns}$	1-neg-size maximum subsequence of ns
1-$negMSS_{ns}$	1-neg-size maximum subsequence set of ns
FSE or fse	First subsequence ending position
LSB or lsb	Last subsequence beginning position
$B(s)$	the bitmap of a sequence s
$N(B(s))$	the number of one in bitmap

a large number of bitmaps, i.e. a vertical bitmap is created for each item in the dataset, and each bitmap has a bit corresponding to each transaction in the dataset. And then, SPAM uses its own defined bitwise operation to calculate each candidate's support. In a nutshell, the bitmap structure and bitwise operations of f-NSP combine a unique NSP mining strategy that adapts to the set theory in e-NSP.

3. Problem Statement

For the self-containing readability consideration, we here define PSP and NSP and their corresponding concepts, which set up the foundation for further introduction of f-NSP in the next section. The main symbols used in this paper are summarized in Table 1.

3.1. Mining Positive Sequential Patterns

Assume a set of *items* $I = \{i_1, i_2, \ldots, i_n\}$, an *itemset* is a subset of I. A *sequence* is an ordered list of *itemsets*. A sequence s is described by $< s_1 s_2 \ldots s_l >$, where $s_j \subseteq I (1 \leq j \leq l)$. s_j is also called an *element* , described as $(x_1 x_2 \ldots x_m)$, where x_k is an item, $x_k \in I (1 \leq k \leq m)$. If an element only contains one item,

the bracket is omitted, i.e., element (x) is coded x. An item in a sequence can appear at most once in an element, but can occur multiple times in different elements.

The sequence *length* is the length of sequence s, which is the total number of items in all elements in s. The sequence $size(s)$ is the size of sequence s, described as $size(s)$, it is the total number of elements in s. For example, assume a sequence $s =< a(bc)de >$ is composed of 4 elements a,(bc), d and e, meanwhile, it is also comprised of 5 items a, b, c, d and e. Hence s is a 4-size and 5-length sequence.

Sequence $s_\alpha =< \alpha_1\alpha_2 \dots \alpha_n >$ is named a *sub-sequence* of sequence $s_\beta =< \beta_1\beta_2 \dots \beta_m >$ and s_β is a *super-sequence* of s_α, denoted as $s_\alpha \subseteq s_\beta$, if there exists $1 \leq j_1 < j_2 < \dots < j_n \leq m$ such that $\alpha_1 \subseteq \beta_{j_1}, \alpha_2 \subseteq \beta_{j_2}, \dots, \alpha_n \subseteq \beta_{j_n}$. We also say that s_β contains s_α. For example, $< c >$, $< ac >$ and $< (ab)d >$ are all sub-sequences of $< (ab)cd >$.

We use a set of tuples $< sid, ds >$ to represent a sequence dataset D, where ds is the data sequence and sid is the number of sequence. $\mid D \mid$ is the number of tuples in D. The set of tuples containing sequence s is denoted as $\{< s >\}$. The *support count* $sup\_c(s)$ is the frequency of $\{< s >\}$ occurring in D, i.e., $sup\_c(s) =\mid \{< s >\} \mid =\mid \{< sid, ds >, < sid, ds > \in D \wedge (s \subseteq ds)\} \mid$. The *support* of s, denoted by $sup(s)$, is the percentage of $\mid \{< s >\} \mid$ in D, i.e., $sup(s) =\mid \{< s >\} \mid / \mid D \mid$. The *minimum support threshold* min_sup is predefined by users.

A sequence s is called a frequent (*positive*) sequential pattern if $sup(s) \geq min\_sup$. By contrast, s is *infrequent* if $sup(s) < min\_sup$. PSP mining aims to discover all positive sequences that satisfy the minimum support. For simplicity, we often omit 'positive' when discussing positive items, positive elements and positive sequences in mining PSP.

3.2. Mining Negative Sequential Patterns

In real applications, the number of negative sequences is large, and many of them are not meaningful. In order to reduce the number of NSC and discover meaningful NSP efficiently, constraints must be added to negative sequences. Here we define the problem of NSP mining, taking three constraints in [7].

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 $ns =< s_1 \dots s_k >$ is to change all negative elements in ns to their positive partners, denoted as $p(ns)$, i.e., $p(ns) = \{< s'_1 \dots s'_k >\mid s'_i = p(s_i), s_i \in ns\}$. 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., $sup(p(ns)) > min\_sup$.

Constraint 2. Format constraint. Continuous negative elements in a NSC are not allowed, because we cannot tell the right order of two continuous negative elements if there is no positive element between them. For example, $< \neg(ab)c\neg d >$ satisfies Constraint 2, but $< \neg(ab)\neg cd >$ does not.

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

Similar to the settings in [8], this is to avoid the complexity of handling partially negative element. For example, $< \neg(ab)cd >$ satisfies Constraint 3, but $< (\neg ab)cd >$ does not because, in element $(\neg ab)$, only $\neg a$ is negative while b is not.

Definition 2. Positive/Negative Element-id Set. *Element-id* is the order number of an element in a sequence. Given a sequence $s =< s_1 s_2 \dots s_m >$, $id(s_i) = i$ is the element identifier of element s_i. Element-id set $EidS_s$ of s is the set that includes all elements and their ids in s, i.e., $EidS_s = \{(s_i, id(s_i)) \mid s_i \in s = (s_1, 1), (s_2, 2), \dots, (s_m, m)\}(1 \leq i \leq m)$.

The set including all positive and negative element-ids of a sequence s is called *positive and negative element-id set* of s, denoted as $EidS_s^+$, $EidS_s^-$, respectively. For example, $s =< \neg(ab)c\neg d >$, $EidS_s^+ = \{(c, 2)\}$, $EidS_s^- = \{(\neg(ab), 1), (\neg d, 3)\}$.

Definition 3. Order-preserving Sequence. For any subset $EidS'_s = \{(\alpha_1, id_1), (\alpha_2, id_2), \dots, (\alpha_p, id_p)\}(1 < p \leq m)$ of $EidS_s$, $\alpha =< \alpha_1\alpha_2 \dots \alpha_p >$, if $\forall \alpha_i, \alpha_{i+1} \in \alpha(1 \leq i < p)$, there exists $id_i < id_{i+1}$, then α is called an order-preserving sequence of $EidS'_s$, denoted as $\alpha = OPS(EidS'_s)$. For example, given $s =< \neg(ab)c\neg d >$, its $EidS_s = \{(\neg(ab), 1), (c, 2), (\neg d, 3)\}$, $EidS_s^+ = \{(c, 2)\}$, $EidS_s^- = \{(\neg(ab), 1), (\neg d, 3)\}$. We can get $OPS(EidS_s^+) =< c >$. Also, if $EidS'_s = \{(\neg(ab), 1), (c, 2)\}$, we can create a sequence $OPS(EidS'_s) =< \neg(ab)c >$.

Definition 4. Sub-sequence and Super-sequence of Negative Sequence. Sequence s_α is called a *sub-sequence* of a negative sequence s_β, and s_β is a *super-sequence* of s_α, if $\forall EidS'_{s_\beta}$, $EidS'_{s_\beta}$ is a subset of $EidS_{s_\beta, s_\alpha} = OPS(EidS'_{s_\beta})$, denoted as $s_\alpha \subseteq s_\beta$. If s_α is a negative sequence, it is required to satisfy Constraint 2, which means that there must not be continuous negative elements in s_α. For example, given $s_\beta =< \neg(ab)cd >$ and $s_\alpha =< \neg(ab)d >$, $EidS_{s_\beta} = \{(\neg(ab), 1), (c, 2), (d, 3)\}$, $EidS'_{s_\beta} = \{(\neg(ab), 1), (d, 3)\}$ is a subset of $EidS_{s_\beta . s_\alpha}$ is a sub-sequence of s_β since $s_\alpha = OPS(EidS'_{s_\beta})$.

Definition 5. Maximum Positive Sub-sequence. We use $n - neg - size$ to denote a negative sequence containing n negative elements. Let $ns =< s_1 s_2 \dots s_m >$ be an *m-size* and *n-neg-size* negative sequence $(m - n > 0)$, $OPS(EidS_{ns}^+)$ is called the maximum positive sub-sequence of ns, denoted as $MPS(ns)$. For example, given a negative sequence $s =< \neg(ab)cd >$, $EidS^+ = \{(c, 2), (d, 3)\}$, its maximum positive sub-sequence is $MPS(s) =< cd >$.

Definition 6. Negative Sequential Pattern. A negative sequence s is a negative sequential pattern (NSP) if its support is not less than the threshold *min_sup*.

4. f-NSP Algorithm

In this section, we introduce f-NSP. Since f-NSP is built on e-NSP [7, 27], its main definitions and NSC generation method are similar to e-NSP. After reviewing e-NSP, we introduce the important definitions, data structure, the method to calculate NSC's supports, and the corresponding algorithm f-NSP.

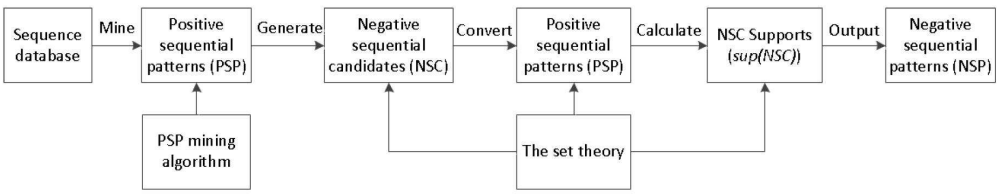


Figure 1: The framework of e-NSP

4.1. The Framework of e-NSP

The framework and working mechanism of e-NSP [7, 27] is illustrated in Figure 1. Given a sequence database, e-NSP works on the following steps to discover NSP.

Step 1. All PSP are mined by GSP; All PSPs (in addition to $1-size$ PSPs) and their *sid* sets are stored in an array. For example, a positive sequence $< ab >$, its *sid* sets is 1, 2, 3 means $< ab >$ is contained in the data sequence with *Sid* 1,2, and 3. The data structure of e-NSP is shown in Table 2.

Step 2. NSC are generated based on the identified PSP in terms of three constraints proposed in Section 3.2;

Step 3. The generated NSC are converted to their corresponding PSP in terms of the negative conversion strategy;

Step 4. The supports of NSC are calculated based on the supports of their corresponding PSPs;

Step 5. Finally, NSP are identified from the NSC to satisfy certain support criteria.

4.2. Negative Containment

As a sub-sequence (e.g., $s_1 =< d >$) may occur more than once in its super-sequence (e.g., $s_2 =< a\ (bc)\ d\ (cde) >$), we need to know the exact positions of s_2 containing s_1 from the left and right sides of s_2. For this, below we define *Negative Containment* and relevant concepts.

Definition 7. First Sub-sequence Ending Position/Last Sub-sequence Beginning Position. Given a data sequence $ds =< d_1 d_2 \dots d_t >$ and a positive sequence α,

(1) if $\exists p(1 < p \leq t)$, $\alpha \subseteq < d_1 \dots d_p > \wedge \alpha \not\subseteq < d_1 \dots d_{p-1} >$, then p is called the *First Sub-sequence Ending Position*, denoted as $FSE(\alpha, ds)$; if $\alpha \subseteq < d_1 >$ then $FSE(\alpha, ds) = 1$;

(2) if $\exists q(1 \leq q < t)$, $\alpha \subseteq < d_q \dots d_t > \wedge \alpha \not\subseteq < d_{q+1} \dots d_t >$, then q is called the *Last Sub-sequence Beginning Position*, denoted as $LSB(\alpha, ds)$; if $\alpha \subseteq < d_t >$ then $LSB(\alpha, ds) = t$;

(3) if $\alpha \not\subseteq ds$, then $FSE(\alpha, ds) = 0$, $LSB(\alpha, ds) = 0$.

For example, given $ds =< a(bc)d(cde) >$. $FSE(< a >, ds) = 1$, $FSE(< c >, ds) = 2$, $FSE(< cd >, ds) = 3$, $LSB(< a >, ds) = 1$, $LSB(< c >, ds) = 4$, $LSB(< cd >, ds) = 2$, $LSB(< (cd) >, ds) = 4$.

The definition of *a data sequence containing a negative sequence* is as follows.

Definition 8. Negative Containment. Let $ds =< d_1 d_2 \dots d_t >$ be a data sequence, $ns =< s_1 s_2 \dots s_m >$ be an $m-size$ and *n-neg-size* negative sequence, (1) if $m > 2t+1$, then *ds does not contain ns*; (2) if $m = 1$ and $n = 1$, then *ds contains ns* when $p(ns) \not\subseteq ds$; (3) otherwise, *ds contains ns* if, $\forall(s_i, id(s_i)) \in EidS^-_{ns}(1 \leq i \leq m)$, one of the following three cases holds:

(a) $(lsb = 1)$ or $(lsb > 1) \wedge p(s_1) \not\subseteq < d_1 \dots d_{lsb-1} >$, when $i = 1$

(b) $(fse = t)$ or $(0 < fse < t) \wedge p(s_m) \not\subseteq < d_{fse+1} \dots d_t >$, when $i = m$

(c) $(fse > 0 \wedge lsb = fse + 1)$ or $(fse > 0 \wedge lsb > fse + 1) \wedge p(s_i) \not\subseteq < d_{fse+1} \dots d_{lsb-1} >$, when $1 < i < m$, where $fse = FSE(MPS(< s_1 s_2 \dots s_{i-1} >), ds)$, $lsb = LSB(MPS(< s_{i+1} \dots s_m >), ds)$.

In the above definition, Case (a) indicates that the first element in ns is negative. "$(lsb > 1) \wedge p(s_1) \not\subseteq < d_1 \dots d_{lsb-1} >$" means that $< d_{lsb} \dots d_t >$ contains $MPS(< s_2 \dots s_m >)$ but $< d_1 \dots d_{lsb-1} >$ does not contain $p(s_1)$. "$lsb = 1$" means that the last sub-sequence's beginning position is 1, so $p(s_1)$ cannot be contained by ds. Case (b) indicates that the last element in ns is negative. Case (c) indicates that the negative element is between the first and last element in ns. "$lsb > fse + 1$" ensures there is at least one element in "$< d_{fse+1} \dots d_{lsb-1} >$". "$fse > 0 \wedge lsb = fse + 1$" means that d_{fse} and d_{lsb} are contiguous elements, so $p(s_i)$ cannot be contained between them.

4.3. Negative Conversion

In order to solve the negative containment problem by using the corresponding positive sequences, we define the *1-neg-size Maximum Sub-sequence* as follows [7, 27].

Definition 9. 1-neg-size Maximum Sub-sequence. For a negative sequence ns, its sub-sequences that include $MPS(ns)$ and one negative element e is called a $1-neg-size$ maximum sub-sequences, denoted as $1-negMS = OPS(EidS^+_{ns}, e)$, where $e \in EidS^-_{ns}$. The sub-sequence set including all 1-neg-size maximum sub-sequences of ns is called 1-neg-size maximum sub-sequence set, denoted as $1-negMSS_{ns}$, $1-negMSS_{ns} = \{OPS(EidS^+_{ns}, e) \mid \forall \in EidS^-_{ns}\}$. For example, 1) $ns =< \neg(ab)c\neg d >$, $1 - negMSS_{ns} = \{< \neg(ab)c >, < c\neg d >\}$; 2) $ns' =< \neg a(bc)d\neg(cde) >$, $1 - negMSS_{ns'} = \{< \neg a(bc)d >, < (bc)d\neg(cde) >\}$.

Corollary . Negative Conversion Strategy. Given a data sequence $ds =< d_1 d_2 \dots d_t >$, and $ns =< s_1 s_2 \dots s_m >$, which is an *m-size* and *n-neg-size* negative sequence, the negative containment problem can be converted to the following problem: data sequence ds contains negative sequence ns if and only if the two conditions hold: (1) $MPS(ns) \subseteq ds$; and (2) $\forall 1 - negMS \in 1 - negMSS_{ns}$, $p(1 - negMS) \not\subseteq ds$.

For example, given $ds =< a(bc)d(cde) >$, 1) if $ns =< a\neg dd\neg d >$, $1 - negMSS_{ns} = \{< a\neg dd >, < ad\neg d >\}$, then ds does not contain ns because $p(< a\neg dd >) =< add > \subseteq ds$; 2) if $ns' =< a\neg bb\neg a(cde) >$, $1 - negMSS'_{ns} = \{< a\neg bb(cde) >, < ab\neg a(cde) >\}$, then ds contains ns because $MPS(ns) =< ab(cde) > \subseteq ds \wedge p(< a\neg bb(cde) >) \not\subseteq ds \wedge p(< ab\neg a(cde) >) \not\subseteq ds$.

This corollary converts the problem *whether a negative sequence is contained in a data sequence* to the problem *whether a data sequence does not contain some related positive sequences.* It enables us to calculate the support of negative sequences by only using the information of corresponding positive sequences.

4.4. NSC Generation

In order to generate all non-redundant NSC from PSP, the key process of generating a NSC is to convert non-contiguous elements in a positive pattern to their negative partners [7, 27]. In detail, for a k-size PSP, its NSC are generated by changing any m non-contiguous element(s) to its (their) negative one(s), $m = 1, 2, \ldots, \lceil k/2 \rceil$, where $\lceil k/2 \rceil$ is a minimum integer that is not less than $k/2$.

For example, the NSC based on $< (xy)ab >$ include: (1) $m = 1$, $< \neg(xy)ab >$,$< (xy)\neg ab >$, $< (xy)\ a\neg b >$; and (2) $m = 2$, $< \neg(xy)a\neg b >$.

4.5. Calculating the NSC Supports

In e-NSP, the supports of NSC are calculated as follows. Given a *m-size* and *n-neg-size* negative sequence ns, for $\forall 1 - negMS_i \in 1 - negMSS_{ns}(1 \leq i \leq n)$, the support of ns in sequence database D is:

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

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

$$sup(ns) = sup(MPS(ns)) - sup(p(ns)) \tag{2}$$

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

$$sup(< \neg e >) = \mid D \mid - sup(< e >) \tag{3}$$

From Equation (1), we can see that e-NSP 'calculates' $sup(ns)$ only based on calculating the union sets of $\{p(1 - negMS_i)\}$, without any additional database scans.

4.6. The Data Structure of f-NSP

As discussed in Section 1, in order to 'calculate' the support of NSC based on the support of corresponding PSPs, e-NSP uses an array to store the PSP's ID (as shown in Table 2). From Equation (1), we can see that the runtime of e-NSP is mainly in calculating $\cup_{i=1}^{n} \{p(1 - negMS_i)\}$. This process will be very time-consuming when the sets' number in $\cup_{i=1}^{n} \{p(1 - negMS_i)\}$ and the number of sequence IDs in the array are large (n and $p(1 - negMSi)$ are large). In order to overcome this deficiency, we propose a novel and efficient data structure, *bitmap*, to store the PSP's information.

f-NSP uses bitmap to represent the PSP data, and calculates NSC supports through bitwise operations. f-NSP creates bitmaps for PSPs whose sizes are greater than 1. If a PSP appears in sequence i, then the bitmap of this PSP is set to 1 in position i; otherwise, the bit is set to 0. The length of each bitmap is equal to the number of sequences in the database.

The f-NSP data structure based on the dataset in Table 3 is illustrated in Table 2, its data structure includes the first, second, fourth columns. The first column stores PSP, the second column holds its support, and the fourth column encloses its bitmap. For example, the bitmap of ab is | 1 | 1 | 1 | 0 | 0 |. This means that ab is contained in the first three data sequences.

Table 2: f-NSP vs. e-NSP Data Structures

PSP	**Support**	**{sid}** for e-NSP	**{sid}** for f-NSP
$< ab >$	3	{1,2,3}	| 1 | 1 | 1 | 0 | 0 |
$< abc >$	2	{1,3}	| 1 | 0 | 1 | 0 | 0 |
...	...	...	...

The process of generating f-NSP data structure is shown in Algorithm 1. In order to efficiently search PSP from the PSPs' list (the first column in table 2), we use a hash table to store this list. (1) For each pattern in PSP, f-NSP creates a hash table to store its related information (Line 2). (2) If the pattern is 1-size PSP, we do not record its bitmap because the equations do not need to calculate the union bitmap of those 1-size PSP (Lines 4-5). (3) We store the other PSP information into the hash table, including the PSP's support and bitmap (Line 7).

Algorithm 1: f-NSP Data structure generation

INPUT: All PSP and their related information;
OUTPUT: NSP;
(1)CreateHash(PSP){
(2)**CREATE** $PSPHash$;
(3) **FOR** (each pattern p in PSP)
(4) **IF**(p.size=1)
(5) $PSPHash$.put(p,p.support_count);
(6) **ELSE**
(7) $PSPHash$.put(p,p.support_count,p.BitMap);
(8) //**END IF**
(9) //**END FOR**
(9)}

4.7. Calculating the Supports of Negative Sequences in f-NSP

As shown in Equation (1), most of the runtime in e-NSP mining is consumed in calculating the union sets $\cup_{i=1}^{n}\{p(1 - negMS_i)\}$ by a hash method. This method is very time-consuming when n or $p(1 - negMS_i)$ is large. Based on the bitmap structure, f-NSP only uses OR bitwise operation on the bitmap of n or $p(1 - negMS_i)$ $(1 \leq i \leq n)$, instead of the hash method, to calculate $\cup_{i=1}^{n}\{p(1 - negMS_i)\}$. That is, the number of 1s in $OR_{i=1}^{n}\{p(1 - negMS_i)\}$ is the element number in $\cup_{i=1}^{n}\{p(1 - negMS_i)\}$.

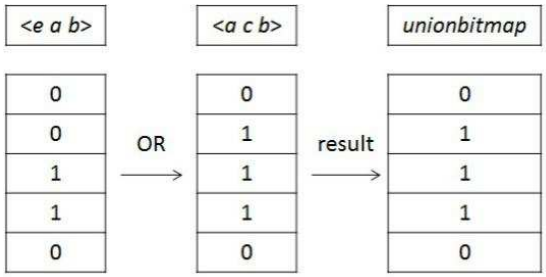


Figure 2: The OR process in the bitwise operation

Formally, assume s is a PSP, its bitmap is denoted by $B(s)$ and the number of 1s in $B(s)$ is denoted by $N(B(s))$. Given an $m-size$ and $n-neg-size$ negative sequence ns, Equation (1) is rewritten as:

$$sup(ns) = sup(MPS(ns)) - N(OR_{i=1}^{n}\{B(p(1-negMS_i))\})/ \mid D \mid \quad (4)$$

We illustrate the above OR operation process by Figure 2. Suppose a PSP is $< eacb >$ and $sup\_c(ab)$=5, one of its NSC ns is $< \neg ea \neg cb >$. Correspondingly, $MPS(ns) =< ab >$, $P(1-negMS_1)=< eab >$, $P(1-negMS_2)=< acb >$. Suppose $B(< eab >) = \mid 0 \mid 0 \mid 1 \mid 1 \mid 0 \mid$ and $B(< acb >) = \mid 0 \mid 1 \mid 1 \mid 1 \mid 0 \mid$. The *unionbitmap* of $B(< eab >)$ OR $B(< acb >)$ is shown in Figure 2. Hence, we can easily obtain $N(unionbitmap)$=3 and then have $sup\_c(< \neg ea \neg cb >)$=2 by Equation (4).

4.8. The f-NSP Algorithm

We here introduce our f-NSP algorithm in Algorithm 2 which mines for NSP based only on identifying PSP. In the algorithm, we use *support count* instead of *support* for convenience when implementing the algorithm.

Algorithm 2: f-NSP Algorithm

INPUT: Sequence dataset D and min_sup;

OUTPUT: NSP;

(1)PSP = GSP(D);

(2)CreateHash(PSP);

(3)**FOR** (each *psp* in PSP)

(4) NSC = NSC_Generation(*psp*);

(5) **FOR** (each *nsc* in NSC)

(6) **IF** ($nsc.size == 1$ && $nsc.neg\_size == 1$)

(7) Calculate *sup* **by Equation** (3) ;

(8) **ELSE IF** ($nsc.size > 1$ && $nsc.neg\_size == 1$)

(9) Calculate *sup* **by Equation** (2);

(10) **ELSE**

Table 3: Example: Data Set

Sid	Data Sequence
1	$< abc >$
2	$< a(ab) >$
3	$< (ae)(ab)c >$
4	$< aa >$
5	$< d >$

(11) Calculate *sup* **by Equation** (1);
(12) //**END IF**
(13) **IF** ($nsc.support\_count / \mid D \mid > min\_sup$)
(14) $NSP.add(nsc)$;
(15) //**END IF**
(16) //**END OF LINE (5)**
(17)//**END OF LINE (3)**
(18)**RETURN** NSP;

The f-NSP algorithm consists of the following steps. (1) Line 1 finds all PSP from the sequence database based on the GSP algorithm [2] (we use GSP as an example, any PSP algorithms can be used). All PSP and their bitmaps are saved in a hash table $PSPHash$ (Line 2);

(2) Each PSP generates NSC(s) by using the NSC generation method in Section 4.4 (Line 4);

(3) The supports of nsc with $1 - neg - size$ are calculated by Equation (2) and Equation (3) (Line 5-9). The supports of other nsc are easily calculated by Equation (4) (Line 10-17). Specifically, we firstly obtain each $1 - negMS's$ bitmap in the $1 - negMSS_{nsc}$. Secondly, the union bitmap is obtained by using the OR operation. Thirdly, the supports of the nsc are calculated by Equation (4) (Lines 11-17). Finally, whether a nsc is an NSP or not is determined by comparing its support with min_sup (Line 18-19).

(4) Return the results and end the whole algorithm (Line 23).

4.9. An Example

The above sections introduce key concepts and components as well as the f-NSP algorithm. This section illustrates how to mine for NSP by using f-NSP. The sequence database is shown in Table 3 (adopted from [9]). In the example, we set $min\_sup = 40\%$.

The process is as follows.

(1) Mine all PSPs by GSP, and fill in the f-NSP data structures, which are shown in Table 4.

(2) Use the NSC generation method to generate all NSC.

Table 4: Example: Results of Positive Patterns

PSP	Support count	Size	Bitmap
$< a >$	4	1	-
$< b >$	3	1	-
$< c >$	2	1	-
$< aa >$	3	2	| 0 | 1 | 1 | 1 | 0 |
$< ab >$	3	2	| 1 | 1 | 1 | 0 | 0 |
$< ac >$	2	2	| 1 | 0 | 1 | 0 | 0 |
$< bc >$	2	2	| 1 | 0 | 1 | 0 | 0 |
$< (ab) >$	2	1	-
$< abc >$	2	3	| 1 | 0 | 1 | 0 | 0 |
$< a(ab) >$	2	2	| 0 | 1 | 1 | 0 | 0 |

(3) Use Equations (2)-(4) to calculate the supports of these NSC. The results are shown in Table 5, and the resulting NSP are marked in bold.

From this example, we can see that $< ac >$ and $< a\neg c >$, $< a(ab) >$ and $< a\neg(ab) >$ are frequent patterns because their supports are higher than min_sup.

5. Theoretical Analysis of f-NSP Time and Space Efficiency

In order to obtain a theoretical conclusion of the time and space efficiency of f-NSP, in this section we compare it with e-NSP.

5.1. Runtime Analysis of f-NSP vs. e-NSP

As discussed above, the runtime of e-NSP is mainly consumed in calculating the union set of $\{p(1 - negMS_i)\}$, i.e., comparing the $Sids$ stored in arrays by using a hash method. The number of comparing times determines the algorithm's runtime. Therefore, the runtime analysis of e-NSP is converted to the analysis of comparison times. From Equation (1) we can see that e-NSP is very sensitive to the number of sets in $\{p(1 - negMS_i)\}$ and the number of sequence IDs in the arrays, while f-NSP almost has nothing to do with them per Equatoin (4).

5.1.1. Analysis of Comparison Times in e-NSP

The runtime analysis of e-NSP comparing with PNSP and NegGSP has already been done in [27]. So we introduce the relevant analysis methods and equations in this section. To understand the comparison times in e-NSP, we first need to know the comparison times of calculating a $k - size$ and $m - neg - size$ NSC, denoted by $CT_{k,m}$; then we need to know the number of NSCs generated from a PSP, $\mid NSC_{k,\forall m} \mid$. Accordingly, we can obtain the total comparison times in e-NSP by $\sum_{i=3}^{k} |PSP_i| * (|NSC_{i,\forall m(m\geq 2)}| * CT_{k,m})$,

Table 5: Example: Results of NSC and Supports ($min\_sup = 40\%$)

PSP	**NSC**	**Related PSP**	**Sup**
$< a >$	$< \neg a >$	$< a >$	1
$< b >$	$< \neg \mathbf{b} >$	$< b >$	**2**
$< c >$	$< \neg \mathbf{c} >$	$< c >$	**3**
$< aa >$	$< \neg aa >$	$< a >, < aa >$	1
	$< a \neg a >$	$< a >, < aa >$	1
$< ab >$	$< \neg ab >$	$< b >, < ab >$	0
	$< a \neg b >$	$< a >, < ab >$	1
$< ac >$	$< \neg ac >$	$< c >, < ac >$	0
	$< \mathbf{a} \neg \mathbf{c} >$	$< a >, < ac >$	**2**
$< bc >$	$< \neg bc >$	$< c >, < bc >$	0
	$< b \neg c >$	$< b >, < bc >$	1
$< (ab) >$	$< \neg \mathbf{(ab)} >$	$< (ab) >$	**3**
$< a(ab) >$	$< \neg a(ab) >$	$< (ab) >, < a(ab) >$	0
	$< \mathbf{a} \neg \mathbf{(ab)} >$	$< a >, < a(ab) >$	**2**
$< abc >$	$< \neg abc >$	$< bc >, < abc >$	0
	$< a \neg bc >$	$< ac >, < abc >$	0
	$< ab \neg c >$	$< ab >, < abc >$	1
	$< \neg ab \neg c >$	$< b >, < ab >, < bc >$	0

where $\mid PSP_i \mid$ is the number of $i-size$ PSPs. Below, we just present the mainly analysis equations. Please find details in [27] if necessary.

$$CT_{k,m} = \sum_{i=1}^{m} |\{p(1-negMSi)\}| = \sum_{i=1}^{m} sup(p(1-negMS_i)), \tag{5}$$

where $|\{p(1-negMS_i)\}|$ is the number of $sids$ in $\{p(1-negMS_i)\}$, i.e., the support of $p(1-negMS_i)$. Then Equation (5) can be rewritten as:

$$\begin{aligned} CT_{k,m} &= \sum_{i=1}^{m} \overline{sup}(p(1-negMS_i)) \\ &= m * \overline{sup}(p(1-negMS_i)) \\ &= m * \overline{sup}(PSP_{k-m+1}), \end{aligned} \tag{6}$$

where $\overline{sup}(p(1-negMS_i))$ is the average support of $\{p(1-negMS_i)\}$.

$|NSC_{k,m}|$ and $|NSC_{k,\forall m}|$ indicate the number of $m-neg-size$ NSCs generated from a $k-size$ PSP and the number of NSCs generated from a $k-size$ PSP, respectively. Then,

$$|NSC_{k,m}| = C_{k-m+1}^{m} = \frac{(k-m+1)!}{m! * (k-2m+1)!} (1 \leq m \leq \lceil k/2 \rceil) \tag{7}$$

$$|NSC_{k,\forall m}| = \sum_{m=1}^{\lceil k/2 \rceil} |NSC_{k,m}| \tag{8}$$

$|PSP_k|$ indicates the number of all $k-size$ PSPs. The total number of NSCs generated from all PSPs by e-NSP, $|NSC|^{e-NSP}$, is

$$|NSC|^{e-NSP} = \sum_{i=1}^{k} |PSP_i| * |NSC_{i,\forall m}| \tag{9}$$

And the comparison times of all union set operations in e-NSP, CT^{e-NSP}, is

$$\begin{aligned} CT^{e-NSP} &= \sum_{i=3}^{k} |PSP_i| * (|NSC_{i,\forall m(m\geq 2)}| * CT_{k,m}) \\ &= \sum_{i=3}^{k} |PSP_i| * (\sum_{m=2}^{\lceil i/2 \rceil} |NSC_{i,m}| * CT_{k,m}) \\ &= \sum_{i=3}^{k} |PSP_i| * (\sum_{m=2}^{\lceil i/2 \rceil} \frac{(i-m+1)!}{m! * (i-2m+1)!} * m * \overline{sup}(PSP_{i-m+1})) \end{aligned} \tag{10}$$

t^{e-NSP} denotes the time of a comparison in e-NSP. Finally, the total time to calculate the whole union sets in e-NSP, T^{e-NSP}, is

$$T^{e-NSP} = t^{e-NSP} * CT^{e-NSP}$$
$$= \sum_{i=3}^{k} |PSP_i| * (\sum_{m=2}^{\lceil i/2 \rceil} \frac{(i-m+1)!}{m! * (i-2m+1)!} * m * \overline{sup}(PSP_{i-m+1}) * t^{e-NSP}) \tag{11}$$

5.1.2. Analysis of Comparison Times in f-NSP

From Equation (4) we can see that the runtime of f-NSP is mainly consumed in calculating the union sets $\{B(p(1-negMS_i))\}$ by the OR operation and it is a bit difficult to represent its comparison times in the way same as e-NSP. We simply use t^{OR} to represent the runtime of OR two bitmaps, then the runtime of f-NSP is:

$$T^{f-NSP} = \sum_{i=3}^{k} |PSP_i| * (\sum_{m=2}^{\lceil i/2 \rceil} \frac{(i-m+1)!}{m! * (i-2m+1)!} * m * t^{OR}) \tag{12}$$

5.1.3. Runtime Comparison between f-NSP and e-NSP against Data Factors

We further assess the runtime of e-NSP, T^{e-NSP} and f-NSP, T^{f-NSP} in terms of the data factors [27] describing the characteristics of a dataset. 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 characteristics of sequential data [27].

- C: Average number of elements per sequence;
- T: Average number of items per element;
- S: Average length of maximal potentially large sequences;
- I: Average size of items per element in potentially maximally large sequences;
- DB: Number of sequences in a database; and
- N: Number of items.

We further derive $T^{e?NSP}$ in Equation (11) and T^{f-NSP} in Equation (12) in terms of the above data factors. The size of sequence k can be represented by S/I. Accordingly, $T^{e?NSP}$ in Equation (11) and T^{f-NSP} in Equation (12) are converted to:

$$T^{f-NSP} = \sum_{i=3}^{S/I} |PSP_i| * (\sum_{m=2}^{\lceil i/2 \rceil} \frac{(i-m+1)!}{m! * (i-2m+1)!} * m * \overline{sup}(PSP_{i-m+1}) * t^{e-NSP}) \tag{13}$$

$$T^{f-NSP} = \sum_{i=3}^{S/I} |PSP_i| * (\sum_{m=2}^{\lceil i/2 \rceil} \frac{(i-m+1)!}{m! * (i-2m+1)!} * m * t^{OR}) \quad (14)$$

Further, we observe the effect on *runtime* by changing one data factor while the others are fixed.

1) C is changed, T, S, I, DB and N are fixed. Although C does not directly appear in the above two equations, increasing C will result in the increase of m and $sup(PSP_{i-m+1})$ to some degree. Hence, T^{e-NSP} will increase too. Since t^{OR} is almost nothing to with C, $T^{e-NSP} - T^{f-NSP}$ will become greater when C increases.

2) T is changed, C, S, I, DB and N are fixed. The effect of changing T is similar to that of adjusting C. Increasing T will result in the increase of m and $sup(PSP_{i-m+1})$ to some degree. Hence, T^{e-NSP} will increase too. Since t^{OR} is almost nothing to with T, $T^{e-NSP} - T^{f-NSP}$ will become greater when T increases.

3) S is changed, C, T, I, DB and N are fixed. Increasing S will increase m and the total number of PSP. It will affect both f-NSP and e-NSP. Generally, $m * t^{OR}$ is far less than $m * (PSP_{i-m+1}) * t^{e-NSP}$, thus $T^{e-NSP} - T^{f-NSP}$ will become bigger when S increases.

4) I is changed, C, T, S, DB and N are fixed. Increasing I will affect both e-NSP and F-NSP. Both of them will increase, while e-NSP increases proportionally faster than f-NSP when I increases, and the gap thus increases too.

5) DB is changed, C, T, S, I and N are fixed. The effect of factor DB will be seen in section 7.6.

6) N is changed, C, T, S, I and DB are fixed. Increasing N will decrease $\overline{sup}(PSP_{i-m+1})$. But T^{f-NSP} has nothing to with DB. Therefore, $T^{f-NSP} - T^{e-NSP}$ will become smaller when N increases.

In summary, f-NSP performs generally more efficiently than e-NSP in terms of the various data factors. f-NSP is especially suitable for datasets with a large number of elements in a sequence, a small number of itemsets, high support of PSP, or a large number of PSP. The above empirical theoretical analysis from the data factor perspective is further verified by the corresponding experiments in Sections 7.4 and 7.5.

5.2. Space Analysis between f-NSP and e-NSP

We assume that all data in the f-NSP data structure are stored in main memory. Let $\mid PSP_1 \mid$ denote the number of 1-size PSP, $\mid PSP_{>1} \mid$ denote the total number of PSP whose size are more than 1. We only record their support values in integer (cost 32 bit, defined as $v = 32$ bit). Both e-NSP and f-NSP just need to store the support of 1-size PSP because we do not need their union set of *sid* or bitmap when calculating the

support of NSC. Let S^{f-NSP} and S^{e-NSP} denote the space consumption of f-NSP and e-NSP respectively, we have:

$$S^{f-NSP} = |PSP_1| * v + |D| * |PSP_{>1}| \qquad (bit) \tag{15}$$

In e-NSP, each PSP owns a *sid* set which contains its corresponding PSPs (as shown in Table 2), and each *sid* is stored as an integer. The space consumption of e-NSP is:

$$S^{e-NSP} = |PSP_1| * v + \sum_{i=1}^{|PSP_{>1}|} sup\_c(PSP_i) * v \qquad (bit) \tag{16}$$

Therefore, in the best case, if each $sup\_c(psp) = min\_sup * |D|$, the minimum S^{e-NSP_min} is achieved.

$$S^{e-NSP\_min} = |PSP_1| * v + min\_sup * |D| * |PSP_{>1}| * v \qquad (bit) \tag{17}$$

Equation (17) minus Equation (15) generates:

$$\begin{aligned} S^{e-NSP\_min} - S^{f-NSP} &= min\_sup * |D| * |PSP_{>1}| * v - |D| * |PSP_{>1}| \\ &= |D| * |PSP_{>1}| * (min\_sup * v - 1) \qquad (bit) \end{aligned} \tag{18}$$

f-NSP and e-NSP have the same data space consumption when:

$$\begin{aligned} S^{e-NSP\_min} - S^{f-NSP} = |D| * |PSP_{>1}| * (min\_sup * v - 1) &= 0 \\ min\_sup * v - 1 &= 0 \\ min\_sup &= 1/v \\ min\_sup &= 0.03125 \end{aligned}$$

Ideally, $min\_sup = 0.03125$ is a space division support, denoted as *sdsup*. Hence f-NSP should be more space-efficient than e-NSP when $min\_sup > sdsup$.

In addition, for each pattern of $PSP_{>1}$, its data space consumption in f-NSP is $|D|$ bits, which is denoted as S_1, and the space consumption in e-NSP is $|D| * sup(ps) * v$ bits, which is denoted as S_2. Obviously, when $sup(ps) < sdsup$, it turns out to be $S_2 < S_1$. Thus, we can utilize this property to improve the algorithm space-efficiency. As a result, a self-adaptive data storage strategy is proposed in the following section.

6. f-NSP+: f-NSP with Space Optimization

The above space analysis reveals that f-NSP may consume more space than e-NSP when *min_sup* is less than *sdsup*. In order to overcome this deficiency, we propose a self-adaptive storage strategy and a corresponding edition of f-NSP: f-NSP+.

Table 6: The self-adaptive storage strategy in f-NSP+

PSP	**Support**	**Bitmap/{sid}**	**note**
$< ab >$	0.32	|1|1|1|0|0|...|0|1|1|	$sup() >= sdsup$
$< abc >$	0.1	$\{1, 3, 7, 8, 9, 10, 20, 21, 22, 30\}$	$sup() < sdsup$
...	...	...	

6.1. Self-adaptive Storage Strategy and f-NSP+

f-NSP+ can automatically choose the bitmap or array method to store PSP information according to PSP supports. That is, when a PSP's support is more than or equal to *sdsup*, f-NSP+ selects the bitmap structure as in f-NSP; otherwise, f-NSP+ uses array as in e-NSP and automatically converts array to bitmap such that f-NSP+ can use bitwise operations to fast calculate the support of NSC as in f-NSP.

The f-NSP+ self-adaptive storage strategy is shown in Table 6. The pseudo code of f-NSP+ is shown in Algorithm 3.

Algorithm 3: f-NSP+ algorithm

INPUT: All PSP, $|D|$, *sdsup*;
OUTPUT: PSP's hash table;
(1)CreateHashPlus(PSP){
(2)**CREATE** $PSPHash$;
(3) **FOR** (each pattern p in PSP)
(4) **IF**(p.size=1)
(5) $PSPHash$.put(p,p.support);
(6) **ELSE IF** (p.support>=$sdsup$)
(7) $PSPHash$.put(p,p.support,p.bitMap);
(8) **ELSE**
(9) $PSPHash$.put(p,p.support,p.sidSet);
(11) **END FOR**
(11)**RETURN** $PSPHash$;
(12)}

---

f-NSP+ works as follows. (1) For each pattern in PSP, f-NSP+ creates a hash table to store its related information (Line 2). (2) For $1 - sizePSP$, we do not record its bitmap because we do not need to calculate the its bitmap (Lines 4-5). (3) For $k - size$ PSP ($k > 1$), f-NSP+ utilizes bitmap to store PSP's information when the PSP's support is greater than *sdsup*; otherwise, f-NSP+ utilizes array to store PSP's information (Lines 6-10).

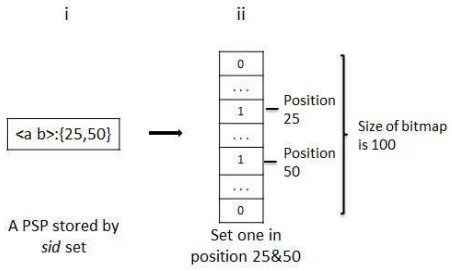


Figure 3: The bitmap conversion process in f-NSP+

6.2. Calculating the Support of Negative Sequences in f-NSP+

In order to use bitwise operations to fast calculate the supports of NSC in f-NSP+, we need to convert the array structure to the bitmap structure. We first read each *sid* from an array, then we set its corresponding position in bitmap with '1'. For simplicity, we usually use the sequence order number as its *sid*. This conversion process is illustrated in Figure 3 (assume a dataset D contains 100 data sequences).

The process of the conversion from array to bitmap structure is summarized in Algorithm 4.

Algorithm 4: Bitmap Conversion Method in f-NSP+

INPUT: $PSPHash$, PSP;
OUTPUT: The bitmap of this PSP;
(1)GetBitmap($PSPHash$, PSP){
(2) **IF**(the type of $PSPHash$.get(PSP) is not bitmap)
(3) **FOR** (int i: $PSPHash$.get(PSP));
(4) $thisBitmap$.set(i);
(5) **END IF**
(6) **RETURN** $thisBitmap$;
(7)}

6.3. Theoretical Space Analysis between f-NSP+ and f-NSP

We now theoretically analyze the space consumption between f-NSP+ and f-NSP.

Let $|PSP_{<sdsup}|$ be the total number of PSP whose support is less than *sdsup*, $\overline{sup}(PSP_{<sdsup})$ be the average support of $PSP_{<sdsup}$, and $|PSP_{\geq sdsup}|$ be the total number of PSP whose support is greater than or equal to *sdsup*. The minimum space consumption of f-NSP+, S^{f-NSP+}, is:

$$S^{f-NSP+} = |PSP_1| * v + |D| * \overline{sup}(PSP_{<sdsup}) * |PSP_{<sdsup}| * v + |D| * |PSP_{\geq sdsup}| \quad (bit) \qquad (19)$$

The minimum space consumption of f-NSP, i.e., Equation 15, is rewritten as:

$$S^{f-NSP+} = |PSP_1| * v + |D| * |PSP_{<sdsup}| + |D| * |PSP_{\geq sdsup}| \qquad (bit) \tag{20}$$

Equation 20 minus equation 19 is:

$$\begin{aligned} S^{f-NSP} - S^{f-NSP+} &= |D| * |PSP_{<sdsup}| - |D| * \overline{sup}(PSP_{<sdsup}) * |PSP_{<sdsup}| * v \\ &= |D| * |PSP_{<sdsup}|(1 - \overline{sup}(PSP_{<sdsup} * v)) \qquad (bit) \end{aligned} \tag{21}$$

When $min\_sup \geq sdsup$, $|PSP < sdsup|$=0, thus $S^{f-NSP} = S^{f-NSP+}$; when $min\_sup < sdsup$, if $|PSP_{<sdsup}| \neq 0$, $\overline{sup}(PSP_{<sdsup})$ should be between min_sup and $sdsup$, so $(1 - sup_{avg} * v) > 0$, $S^{f-NSP} > S^{f-NSP+}$. Meanwhile, with min_sup decreasing, $|PSP_{<sdsup}|$ will increase and $\overline{sup}(PSP_{<sdsup})$ will decrease, and $S^{f-NSP} - S^{f-NSP+}$ will become large. Therefore, f-NSP+ can consume less space than f-NSP when min_sup is very small.

7. Experiments and Results

We conduct experiments on 17 synthetic and real datasets to compare f-NSP with three available NSP mining methods, e-NSP [7, 27], NegGSP [8] and PNSP [9]. To compare their performance, we make PNSP and NegGSP to follow the same constraints and definitions in e-NSP. However, due to the huge gap between two mining strategies, one is to re-scan the database (i.e. NegGSP and PNSP) and the other is to calculate NSC supports by equations (i.e. f-NSP, f-NSP+ and e-NSP). We just compare f-NSP with e-NSP, NegGSP and PNSP in two datasets, DS1 and DS3, as shown in Figure 4.

In the following experiments, all positive patterns are identified by GSP. NSP are further mined by baselines, f-NSP and f-NSP+ respectively. We carry out intensive experiments to compare the difference between these NSP mining algorithms in terms of computational cost and space cost in terms of different data factors. All algorithms are implemented in Java on a PC with Intel Core i5 CPU of 3.2GHz, 32GB memory and Windows 7 Professional.

7.1. Data Sets

Four source datasets are used for the experiments. They include both real datasets and synthetic datasets generated by IBM data generator [1]. By partitioning the data, we obtain 17 datasets in total.

Dataset 1 (DS1), C8_T4_S6_I6_DB100k_N100. We further adjust $DS1$ to generate 10 additional datasets, labelled as $DS1.x$ $(x = 1, \ldots, 10)$.

Dataset 2 (DS2), C15_T8_S20_I10_DB10k_N0.2k.

Dataset 3 (DS3) is from UCI and consists of MSNBC.com anonymous web data about web page visits. Visits are recorded at the page category and are recorded in a temporal order. There are 989,818 records in

the dataset. The average number of elements in a sequence is 4, and each element only has one item. Its file size is 12M, which is relatively small since the dataset is shown in a special format without sequence IDs and element IDs.

Dataset 4 (DS4) is real application dataset about health insurance claim sequences. The data set contains 5,269 customers/sequences. The average number of elements in a sequence is 21. The minimum number of elements in a sequence is 1, and the maximum number is 144. The file size is around 5M.

Dataset 5 (DS5) is C8_T4_S6_I6_DB10k_N800_10times, which means we grow the dataset C8_T4_S6_I6_DB10k_N800 tenfold. $DS5$ is a special dataset, it has a large number of itemsets, and all of the PSPs (expect 1-size PSPs) mined from $DS5$ have the support value less than *sdsup*. Then we use this dataset to thoroughly compare the space consumption between f-NSP and f-NSP+.

Dataset 6 (DS6) is another KDD-CUP 2000 dataset and contains 59,601 sequences of click-stream data from an e-commerce. It contains 497 distinct items. The average length of sequences is 2.42 items with a standard deviation of 3.22. In this dataset, there are some long sequences. For example, 318 sequences contains more than 20 items.

Dataset 7 (DS7) is a dataset of 20,450 sequences of click stream data from the website of FIFA World Cup 98. It has 2,990 distinct items (webpages). The average sequence length is 34.74 items with a standard deviation of 24.08 items. This dataset was created by processing a part of the web logs of the world cup.

7.2. Computational Cost

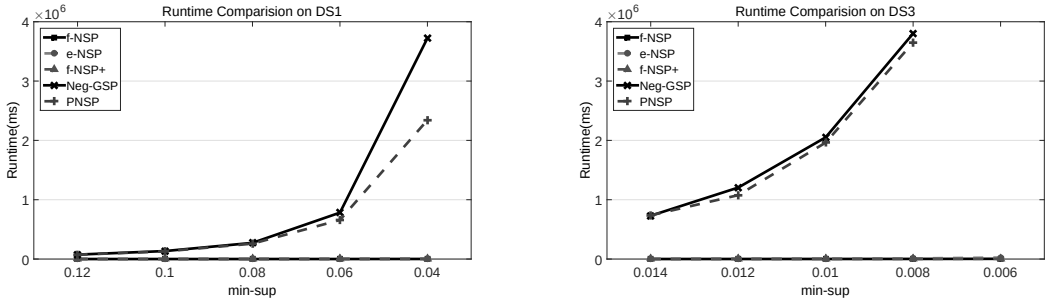
Figure 4 obviously reveals that PNSP and NegGSP are extremely Inefficient. If we embed these five results into one figure, the runtimes of f-NSP, f-NSP+ and e-NSP are almost overlapped. And we do not compare our methods with PNSP and NegGSP in the following experiments since (1) very substantial experimental outcomes are available in [27] which compared e-NSP with such baseline approaches, and (2) experiments in [27] showed that e-NSP is tens to thousands of times faster than NegGSP and PNSP.

The runtime of three approaches (f-NSP, f-NSP+ and e-NSP) is shown in Figure 5. The efficiency of f-NSP is much higher than e-NSP likewise that of f-NSP+, which is extremely close to f-NSP on $DS1-4$. For example, f-NSP consumes 10.9% to 12.3% runtime of e-NSP on $DS1$ when *min_sup* decreases from 0.05 to 0.03. When *min_sup* is high, such as the experiment on $DS2$, f-NSP takes less than 2% of the runtime of e-NSP. But when *min_sup* is very low, the runtime gap between f-NSP and e-NSP will be smaller.

7.3. Space Cost

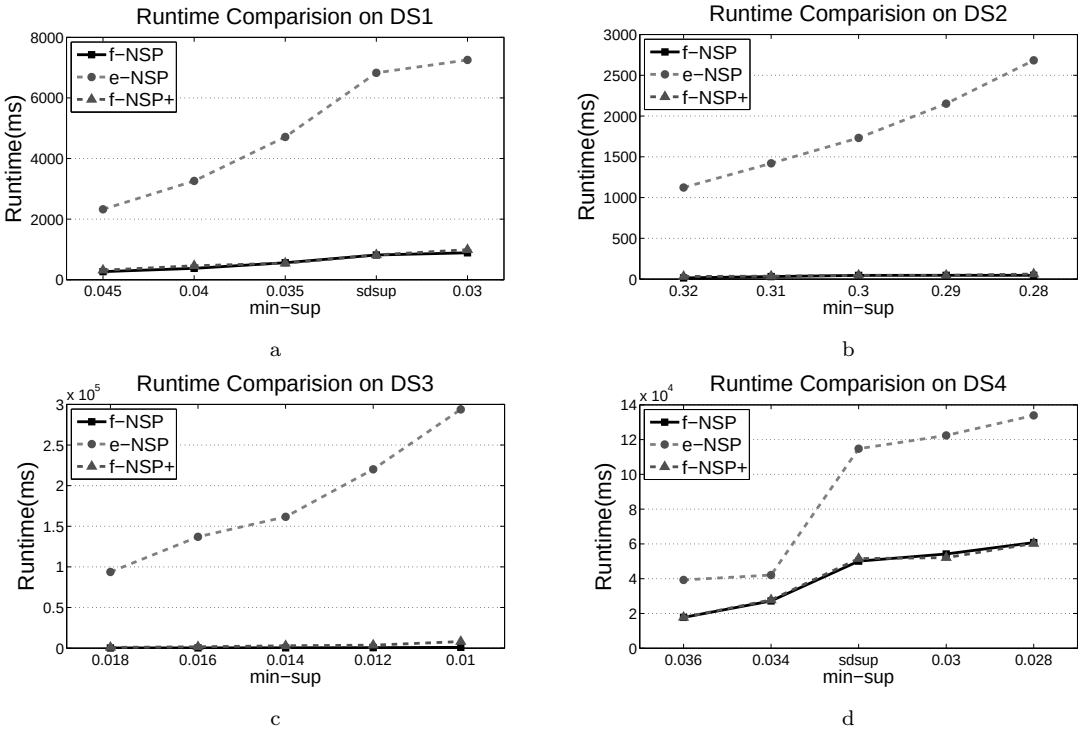
After comparing the minimum data consumption of three algorithms e-NSP, f-NSP and f-NSP+, we carry out experiments in terms of the following three cases: (1) *min_sup* is greater than *sdsup*; (2) *min_sup* is less than *sdsup*; and (3) *min_sup* is on both sides of *sdsup*.

As shown in Figure 6, we can find that both f-NSP and f-NSP+ consume less data space than e-NSP when *min_sup* is greater than *sdsup*. In the first case, the memory consumption of f-NSP and f-NSP+ are close, such as shown in the experiments on $DS2$, and f-NSP takes at least 8.1% data space consumption of



a b

Figure 4: Runtime Comparison with three baselines



a b

c d

Figure 5: Runtime Comparison

e-NSP. In the second case, f-NSP may cost more data space than e-NSP and f-NSP+ is more space-efficient than f-NSP and e-NSP, such as shown in the experimental results on $DS3$, and this phenomenon has been proved in Section 5.2. In the last case, an inflection point is detected, which means f-NSP+ will cost less data space than f-NSP when min_sup is less than $sdsup$, such as the experiments on $DS1$ and $DS4$.

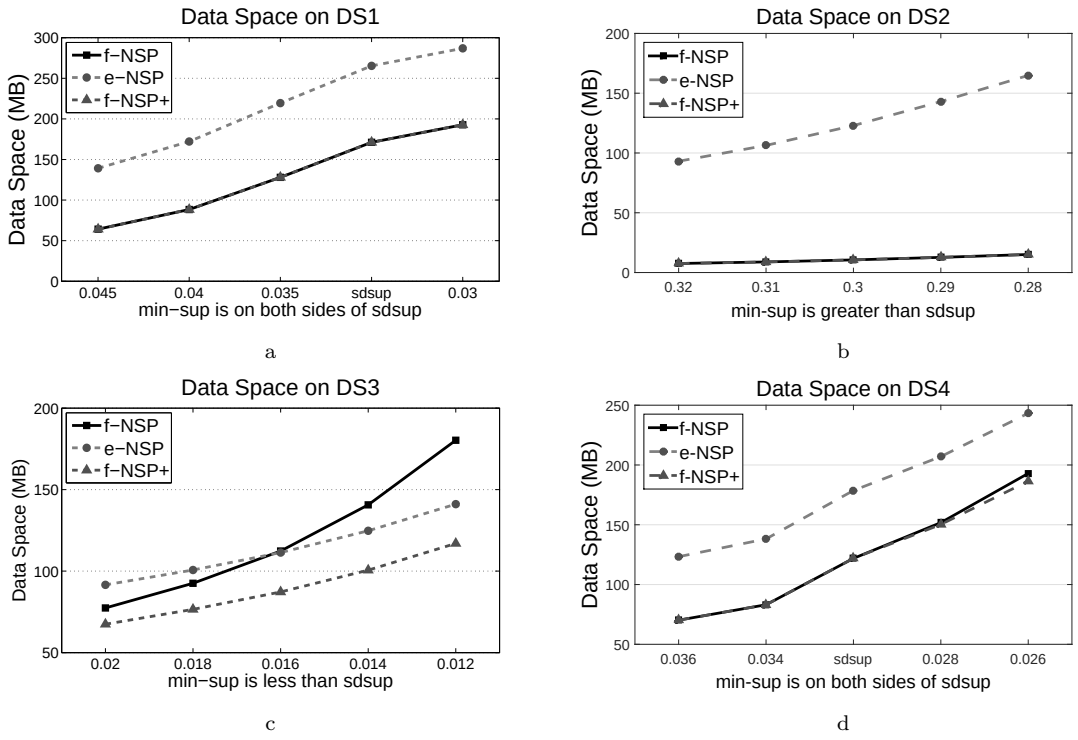


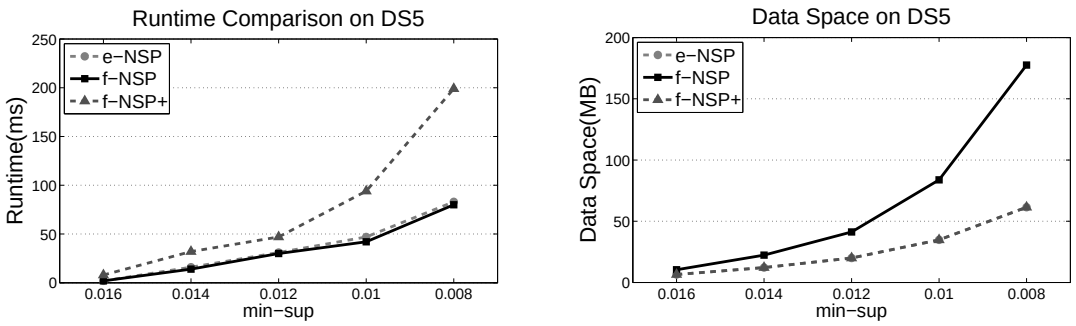
Figure 6: Data Space Consumption Comparison

7.4. Experiments on DS5

We further compare the efficiency of three approaches. All of the PSPs (expect 1-size PSPs) mined from $DS5$ have the support values less than *sdsup*. Figure 7 indicates that the runtime of f-NSP is similar to e-NSP, but f-NSP+ costs much more time than the first two approaches. Moreover, e-NSP has the same data space as f-NSP+ in terms of their use of the same data structure when *min_sup* is less than *sdsup*, while f-NSP consumes the most. Experiments show that e-NSP has its advantage of performing on datasets with a large number of itemsets, such as $DS5$. In order to compare the e-NSP, f-NSP and f-NSP+ more comprehensively, we test the impact of data characteristics in Section 7.5.

7.5. Sensitivity of Data Factors

We analyze the performance f-NSP and f-NSP+, compared to e-NSP, in terms of the above defined data factors to observe the impact of the data factors (see Section 5.1) on their performance. We generate various types of synthetic datasets with different distributions. Dataset $DS1$ is extended to ten different datasets by tuning each factor, as shown in Table 7. For example, dataset $DS1.1$ (C4T4S6I6.DB10k.N100) is



a b

Figure 7: Comparison on DS5

Table 7: Sensitivity of Data Factors on e-NSP, f-NSP and f-NSP+

Dataset ID	**Data Factors**	min_sup	${}_t{}^{e-NSP}$ **(ms)**	${}_t{}^{f-NSP}$ **(ms)**	${}_t{}^{f-NSP+}$ **(ms)**	${}_t{}^{f-NSP}/{}_t{}^{e-NSP}$	${}_t{}^{f-NSP}/{}_t{}^{fNSP+}$
DS1	C8T4S6I6.DB10k.N100	0.03	7254	889	1089	12.3%	81.6%
		0.04	3260	374	429	11.5%	87.2%
		0.05	1716	187	187	10.9%	100.0%
DS1.1	**C4**T4S6I6.DB10k.N100	0.006	500	452	577	90.4%	78.3%
		0.008	280	245	296	87.5%	82.7%
		0.01	156	140	187	89.7%	74.9%
DS1.2	**C12**T4S6I6.DB10k.N100	0.1	12543	312	328	2.5%	95.1%
		0.12	6458	188	203	2.9%	92.6%
		0.14	3401	109	109	3.2%	100%
DS1.3	C8**T8**S6I6.DB10k.N100	0.2	4634	62	62	1.3%	100.0%
		0.22	2885	31	32	1.1%	96.9%
		0.24	1950	31	31	1.6%	100%
DS1.4	C8**T12**S6I6.DB10k.N100	0.3	40779	187	187	0.5%	100.0%
		0.35	14882	73	78	0.5%	93.6%
		0.4	5429	41	47	0.8%	87.2%
DS1.5	C8T4**S12**I6.DB10k.N100	0.03	5350	780	827	14.6%	94.3%
		0.04	2387	343	347	14.4%	98.8%
		0.05	1248	172	172	13.8%	100.0%
DS1.6	C8T4**S18**I6.DB10k.N100	0.03	1326	281	296	21.2%	94.9%
		0.04	624	125	141	20.0%	88.7%
		0.05	327	78	79	23.9%	98.7%
DS1.7	C8T4S6**I10**.DB10k.N100	0.03	5273	858	862	16.3%	99.5%
		0.04	2667	421	426	15.8%	98.8%
		0.05	1498	218	218	14.6%	100.0%
DS1.8	C8T4S6**I14**.DB10k.N100	0.06	5101	874	877	17.1%	99.6%
		0.07	3073	468	484	15.2%	96.7%
		0.08	1919	312	317	16.2%	98.4%
DS1.9	C8T4S6I6.DB10k.**N200**	0.02	359	156	218	43.5%	71.6%
		0.025	203	109	109	53.7%	100.0%
		0.03	39.5	23.3	0.06	0.3%	%
DS1.10	C8T4S6I6.DB10k.**N400**	0.015	75	62	73	82.7%	84.9%
		0.02	46	31	41	67.4%	75.6%
		0.025	35	31	31	88.6%	100.0%

different from $DS1$ (C8T4S6I6.DB10k.N100) on C factor, which means they have different average numbers of elements in a sequence. We mark the difference by underlining the distinct factor for each dataset in Table 7, and Figure 8 further illustrates the comparison of f-NSP with e-NSP on different data factor combinations in terms of various minimum supports.

We use t^{f-NSP}/t^{e-NSP} to show f-NSP's performance compared with e-NSP, and t^{f-NSP}/t^{f-NSP+} to show f-NSP+'s performance compared with f-NSP. From the results, we can see that factors C, T and N seriously affect the runtime of f-NSP and f-NSP+, while factors S and I do not. When factor C is high, such as $DS1.2$, f-NSP and f-NSP+ work better than that on datasets with small C. Similar results hold for T, such as $DS1.3$ and $DS1.4$ with big T, compared with $DS1$ with small T, and f-NSP takes 0.5% of the runtime of e-NSP in $DS1.4$. On the contrary, When N is small, such as in $DS1$, f-NSP and f-NSP+ work better than that with big N, such as in $DS1.9$ and $DS1.10$.

Furthermore, the runtime of f-NSP+ may be slightly higher than f-NSP when *min_sup* is less than *sdsup*, because f-NSP+ has better space consumption by using a self-adaptive data structure strategy.

7.6. Scalability Test

f-NSP calculates supports based on the bitmaps of corresponding positive patterns, thus its performance is sensitive to the size of datasets. If a dataset is huge, it produces large bitmaps. The scalability test is conducted to evaluate the f-NSP performance on large datasets. Figure 9 shows the results of f-NSP on datasets DS 6 and DS 7, in terms of different data sizes: from 10 (i.e., 8M) to 50 (40M and 2, 980, 050 sequences) times of $DS6$, and from 5 (13M) to 25 (65M and 511, 250 sequences) times of $DS7$, with various low minimum supports *min_sup* 0.0022, 0.0024, 0.0026 and 0.0028 on DS 6, and 0.18, 0.20, 0.22 and 0.24 on DS 7, respectively.

Both results on DS 6 and DS 7 in Figure 9 show that the growth of runtime of f-NSP on large scale data follows a roughly linear relationship with the data size increase on different minimum supports. The results in this scalability test show that f-NSP works particularly well on very large datasets.

7.7. Experimental Result Summary

In summary, the experiments show that:

(1) Compared with e-NSP, f-NSP not only significantly speeds up e-NSP but also reduces the data space consumption when *min_sup* is more than *sdsup*.

(2) f-NSP+ overcomes the data structure shortcoming of f-NSP and it has better space-efficiency than f-NSP when *min_sup* is less than *sdsup*. However, the runtime of f-NSP+ will be slightly higher than f-NSP.

(3) f-NSP and f-NSP+ do not perform perfectly on datasets with a large number of itemsets, such as $DS5$, but they work very well on the datasets with a big number of elements in a sequence, a big number of items in an element, and a small number of itemsets.

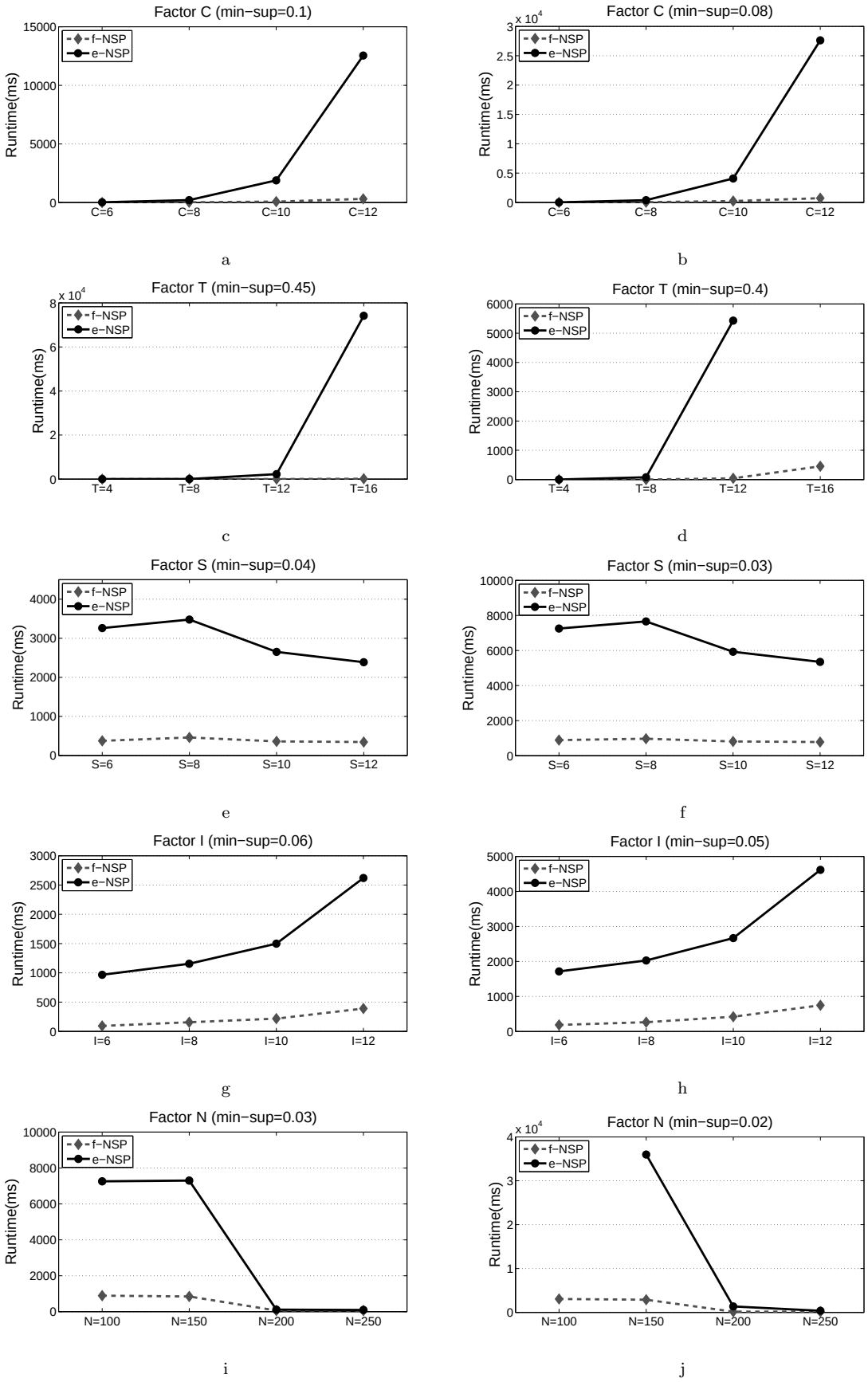
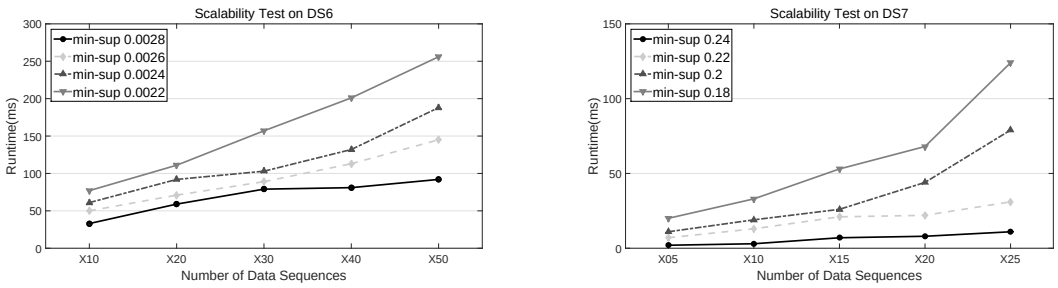


Figure 8: Runtime Comparison on Various Factors



a b

Figure 9: Scalability Test on Data Factor DB on DS6 and DS7

8. Conclusions and Future Work

Mining NSP is very challenging due to the large search space of negative candidates. Very few NSP mining algorithms are available and most of them are very inefficient since they obtain the support of NSC by scanning database repeatedly. Instead, e-NSP only uses PSP's information stored in an array structure to 'calculate' the support of NSC without database re-scanning. e-NSP is probably the best method for NSP mining. While e-NSP is good at handling sparse datasets, when datasets becomes dense, the key process to obtain supports of NSC in e-NSP becomes time-consuming. Therefore, this paper proposes an improved algorithm f-NSP which uses a novel and efficient data structure, *bitmap*, to store the PSP's information and then obtain the support of NSC only by bitwise operations, which is much faster than the hash method in e-NSP. f-NSP, however, will consume more storage space than e-NSP when PSP's support is less than *sdsup*. Thus, we also propose a self-adaptive storage strategy to be incorporated into f-NSP, forming another algorithm f-NSP+. f-NSP+ can not only automatically choose bitmap or array to store PSP's information according to the PSP's support, but also can use bitwise operations to fast calculate the support of NSC as in f-NSP.

The experimental results and comparisons on real and synthetic datasets clearly show that f-NSP and f-NSP+ far better than PNSP and NegGSP and substantially improve the efficiency of e-NSP both in runtime and space cost. f-NSP+ has better space efficiency but slightly longer runtime when *min_sup* is very low. e-NSP also has its advantage of performing on datasets with a large number of itemsets. Hence, the choice between f-NSP, f-NSP+ and e-NSP is obvious a space-time tradeoff, and users can select suitable method according to their own data characteristics.

Through the experiments we can find that f-NSP and f-NSP+ cannot perform perfectly on datasets with a large number of itemsets. Accordingly, we are working on finding a method to overcome this shortcoming. Moreover, from this paper, there are three constraints to restrict the characteristics of NSP, which are detailed in Section 3.2. These constraints may hide some useful NSP to be mined. Our other efforts are to release the element negative constraint and research on how to find certain items that can be negative in an element.

9. Acknowledgement

This work was partially supported by National Natural Science Foundation of China (71271125), and Natural Science Foundation of Shandong Province, China (ZR2011FM028), and Australian Research Council Linkage grant (DP130102691).

References

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In ICDE'95, pp.3-14, 1995.

[2] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements,In EDBT 96: Proc. of the 5th International Conference on Extending Database Technology, London, UK, 1996, pp. 1-7.

[3] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation,In KDD 02: Proc. of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 2002. pp. 429-435.

[4] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu. Freespan: frequent pattern-projected sequential pattern mining, In KDD 00: Proc. Of the 6th ACM SIGKDD international conference on Knowledge discovery and data mining. New York, NY, USA. 2000, pp.355-359.

[5] M. J. Zaki. Spade: An efficient algorithm for mining frequent sequences, Machine Learning, 2001, 42(1-2), pp.31-60.

[6] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth, In ICDE 01: Proc. of the 17th International Conference on Data Engineering, Washington, DC, USA, 2001. pp.215-226.

[7] X.J. Dong , Z.G. Zheng, L.B. Cao, Y.C. Zhao, C.Q. Zhang, J.J. Li, W. Wei, Y.M. Ou. E-NSP: Efficient negative sequential pattern mining based on identified positive patterns without database rescanning. International Conference on Information and Knowledge Management, Proceedings, 2011, pp. 825-830.

[8] Z. Zheng, Y. Zhao, Z. Zuo, L. Cao: Negative-GSP: An Efficient Method for Mining Negative Sequential Patterns. The 8th Australian Data Mining Conference. AusDM 09. Data Mining and Analytics, vol.101. pp. 63-67.

[9] S.C. Hsueh, M.Y. Lin, C.L. Chen.: Mining Negative Sequential Patterns for E-commerce Recommendations. Proceedings of the 2008 IEEE Asia-Pacific Services Computing Conference.2008, pp.1213-1218.

[10] N.P. Lin, H.J. Chen, and W.H. Hao. Mining negative sequential patterns. In Proc. of the 6th WSEAS International Conference on Applied Computer Science, Hangzhou, China, 2007, pp. 654-658.

[11] X. Wu, C. Zhang, and S. Zhang. Efficient mining of both positive and negative association rules. ACM Trans. Inf. Syst., 2004, pp.381 - 405.

[12] Z. Zheng, Y. Zhao, Z. Zuo, and L. Cao. An efficient ga-based algorithm for mining negative sequential patterns. In PAKDD 10, 2010, pp. 262 -273.

[13] N.P. Lin, H.J. Chen, and W.-H. Hao. Mining Negative Fuzzy Sequential Patterns. Proceedings of the 7th WSEAS International Conference on Simulation, Modeling and Optimization, Beijing, China, 2007, pp 52-57.

[14] N.P. Lin, H.J. Chen, and W.-H. Hao. Mining Strong Positive and Negative Sequential Patterns. WSEAS TRANSACTIONS on COMPUTERS. Issue 3, Volume7, 2008, pp. 119-124.

[15] W.M. Ouyang and Q.H. Huang. Mining negative sequential patterns in transaction databases. In Proc. Of 2007 International Conference on Machine Learning and Cybernetics, Hong Kong, China, 2007, pp. 830-834.

[16] W.M. Ouyang, Q.H. Huang, and S. Luo. Mining positive and negative fuzzy sequential patterns in large transaction databases. Proceedings - 5th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2008, v 5, pp. 18-23.

[17] W.M. Ouyang and Q.H. Huang. Mining positive and negative fuzzy multiple level sequential patterns in large transaction databases. Proceedings of the 2009 WRI Global Congress on Intelligent Systems, GCIS 2009, v1,pp. 500-504.

[18] W.M. Ouyang and Q.H. Huang. Mining positive and negative sequential patterns with multiple minimum supports in large transaction databases. Proceedings - 2010 2nd WRI Global Congress on Intelligent Systems, GCIS 2010, v2, pp. 190-193.

[19] V.Rastogi and V.K. Khare. Apriori Based: Mining Positive and Negative Frequent Sequential Patterns. International Journal of Latest Trends in Engineering and Technology (IJLTET), Vol. 1 Issue 3 September 2012, pp. 24 -33.

[20] V.K. Khare and V. Rastogi. Mining Positive and Negative Sequential Pattern in Incremental Transaction Databases. International Journal of Computer Applications (0975 - 8887), Vol 71 - No.1, 2013, pp. 18-22.

[21] Y. Zhao, H. Zhang, L. Cao, C. Zhang, and H. Bohlscheid. Efficient mining of event-oriented negative sequential rules. IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, 2008, pp.336 - 342.

[22] Y. Zhao, H. Zhang, L. Cao, C. Zhang, and H. Bohlscheid. Mining both positive and negative impact-oriented sequential rules from transactional data. In PAKDD 09, volume 5476, 2009, pp. 656 - 663.

[23] Y.C. Zhao, H.F. Zhang, S.S. Wu, J. Pei, L.B. Cao, C.Q. Zhang and H. Bohlscheid. Debt detection in social security by sequence classification using both positive and negative patterns. Lecture Notes in Computer Science, v 5782, 2009, pp 648-663.

[24] S. Mesbah and F. Taghiyareh. A new sequential classification to assist Ad auction agent in making decisions. 2010 5th International Symposium on Telecommunications (IST), 2010, pp. 1006 - 1012.

[25] P. Kazienko. Filtering of web recommendation lists using positive and negative usage patterns. Proceedings of the 11th international conference, KES 2007 and XVII Italian workshop on neural networks conference on Knowledge-based intelligent information and engineering systems: Part III. Springer-Verlag, 2007, pp.1016-1023.

[26] X. Dong, F. Sun, X. Han and R. Hou. Study of positive and negative association rules based on multi-confidence and chi-squared test. Advanced Data Mining and Applications. ADMA06, Springer Lecture Notes in Computer Science 4093, Springer Berlin Heidelberg, 2006, pp.100-109.

[27] L. Cao, X. Dong and Z. Zheng. e-NSP: Efficient Negative Sequential Pattern Mining, Artificial Intelligence, 235: 156-182, 2016.

[28] J. Pisharath, Y. Liu, B. Ozisikyilmaz, R. Narayanan, W.K. Liao, A. Choudhary, and G. Memik. NU-MineBench Version 2.0 Data Set and Technical Report, http://cucis.ece.northwestern.edu/projects/DMS/MineBenchDownload.html

[29] G. W. Schwartz,A. Shokoufandeh, Ontan S, et al. Using a novel clumpiness measure to unite data with metadata: finding common sequence patterns in immune receptor germline V genes[J]. Pattern Recognition Letters, 2016, 74:24-29.

[30] J. Grim. Sequential pattern recognition by maximum conditional informativity. Pattern Recognition Letters, 2014, 45(1):39-45.

[31] R. Willink. A sequential algorithm for recognition of a developing pattern with application in orthotic engineering. Pattern Recognition, 2008, 41(2):627-636.

[32] Song W, Yang B, Xu Z. Index-BitTableFI: An improved algorithm for mining frequent itemsets. Knowledge-Based Systems, 2008, 21(6):507-513.

[33] Zaki M J, Gouda K. Fast vertical mining using diffsets. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2003:326-335.

[34] Savasere A, Omiecinski E, Navathe S. Mining for strong negative associations in a large database of customer transactions. International Conference on Data Engineering, IEEE. 1998. pp.494-502.

[35] Hmlinen W. Kingfisher: an efficient algorithm for searching for both positive and negative dependency rules with statistical significance measures. Knowledge and Information Systems, 2012, 32(2):383-414.

[36] Antonie M L, Zaane O R. An associative classifier based on positive and negative rules. Dmkd, 2004:6469.

[37] Antonie M L, Zaane O R. Mining Positive and Negative Association Rules: An Approach for Confined Rules. European Conference on Principles of Data Mining and Knowledge Discovery. Springer Berlin Heidelberg, 2004:27-38.

[38] Pham T D. Spectral distortion measures for biological sequence comparisons and database searching. Pattern Recognition, 2007, 40(2):516-529.

[39] Yang C, Zhou J. Non-stationary data sequence classification using online class priors estimation. Pattern Recognition, 2008, 41(8):2656-2664.

[40] Y. Gong, T. Xu, X. Dong, et al. e-NSPFI: Efficient Mining Negative Sequential Pattern from both Frequent and Infrequent Positive Sequential Patterns. International Journal of Pattern Recognition and Artificial Intelligence,2017, 31(2)

[41] Bi C. Comparison of optimization techniques for sequence pattern discovery by maximum-likelihood. Pattern Recognition Letters, 2010, 31(14):2147-2160.

[42] Abin A A, Beigy H. Active selection of clustering constraints: a sequential approach. Pattern Recognition. 2014,47(3):1443-1458.

[43] S. Zhang, Z. Du, J. Wang. New Techniques for Mining Frequent Patterns in Unordered Trees. Cybernetics, IEEE Transactions on, Vol: 45(6), 2014, pp. 1113-1125.

[44] J. Luna, J. Romero, C. Romero, S. Ventura. On the Use of Genetic Programming for Mining Comprehensible Rules in Subgroup Discovery. Cybernetics, IEEE Transactions on, 2014, 44(12):2329-2341.

[45] Zhao Y, Yu J X, Wang G, et al. Maximal Subspace Coregulated Gene Clustering. IEEE Transactions on Knowledge and Data Engineering, 2007, 20(1):2329-2341.

[46] Li Y, Zhao Y, Wang G, et al. ELM-Based Large-Scale Genetic Association Study via Statistically Significant Pattern. IEEE Transactions on Systems Man and Cybernetics Systems,2017, (99):1-14.

[47] Zhao Y, Wang G, Zhang X, et al. Learning Phenotype Structure Using Sequence Model. IEEE Transactions on Knowledge and Data Engineering, 2014, 26(3):667-681.

[48] Fu K S, Chien Y T, Cardillo G P. A dynamic programming approach to sequential pattern recognition[J]. IEEE transactions on pattern analysis and machine intelligence, 1986 (3): 313-326.

[49] Adhikari A, Rao P R. Mining conditional patterns in a database[J]. Pattern Recognition Letters, 2008, 29(10): 1515-1523.

[50] Adhikari A, Rao P R. Synthesizing heavy association rules from different real data sources[J]. Pattern Recognition Letters, 2008, 29(1): 59-71.

[51] Adhikari J, Rao P R. Measuring influence of an item in a database over time[J]. Pattern Recognition Letters, 2010, 31(3): 179-187.

[52] Hu J, Mojsilovic A. High-utility pattern mining: A method for discovery of high-utility item sets[J]. Pattern Recognition, 2007, 40(11): 3317-3324.

[53] Rushing J A, Ranganath H S, Hinke T H, et al. Using association rules as texture features[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2001, 23(8): 845-858.

[54] Li K, Fu Y. Prediction of human activity by discovering temporal sequence patterns[J]. IEEE transactions on pattern analysis and machine intelligence, 2014, 36(8): 1644-1657.