e-NSP: Efficient Negative Sequential Pattern Mining Based on Identified Positive Patterns Without Database Rescanning

Xiangjun Dong
School of Information
Shandong Polytechnic
University, Jinan, China
dxj@spu.edu.cn

Zhigang Zheng,Longbing Cao
QCIS, AAI
University of Technology,
Sydney, Australia
{zgzheng,lbcao}@it.uts.edu.au

Yanchang Zhao
Centrelink
Australia
yanchang.zhao@
centrelink.gov.au

Chengqi Zhang
QCIS
University of Technology,
Sydney, Australia
chengqi@it.uts.edu.au

Jinjiu Li
QCIS, AAI
University of Technology,
Sydney, Australia
jinjiu.li@eng.uts.edu.au

Wei Wei, Yuming Ou
QCIS, AAI
University of Technology,
Sydney, Australia
{wwei,yuming}@it.uts.edu.au

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 required in calculating Negative Sequential Candidates (NSC). Very few approaches are available for mining NSP, which mainly rely on re-scanning databases after identifying PSP. As a result, they are very inefficient. In this paper, we propose an efficient algorithm for mining NSP, called e-NSP, which mines for NSP by only involving the identified PSP, without re-scanning databases. First, negative containment is defined to determine whether or not a data sequence contains a negative sequence. Second, an efficient approach is proposed to convert the negative containment problem to a positive containment problem. The supports of NSC are then calculated based only on the corresponding PSP. Finally, a simple but efficient approach is proposed to generate NSC. With e-NSP, mining NSP does not require additional database scans, and the existing PSP mining algorithms can be integrated into e-NSP to mine for NSP efficiently. e-NSP is compared with two currently available NSP mining algorithms on 14 synthetic and real-life datasets. Intensive experiments show that e-NSP takes as little as 3% of the runtime of the baseline approaches and is applicable for efficient mining of NSP in large datasets.

Categories and Subject Descriptors

H.2.8 [**Data-base Applications**]: Data Mining

General Terms

Algorithms, Performance, Experimentation

Keywords

Negative Sequential Pattern, Sequence Mining, Negative Containment

1. INTRODUCTION

Negative sequential patterns (NSP) refer to sequences with non-occurring items. For instance, assume p_1=<a b c X> is a positive sequential pattern (PSP); p_2=<a b $\neg c$ Y> is a NSP. It is increasingly recognized [5][10] that such NSP, composed of both occurring and non-occurring items, can play an irreplaceable role in deeply understanding and tackling many business applications, such as the associations between treatment services and illnesses, and the detection of high impact occurring (positive behaviors) and non-occurring behavior (negative behaviors) sequences [3, 2], which cannot be handled by mining PSP only. NSP cannot be described or discovered by classic PSP mining algorithms such as GSP, SPADE, PrefixSpan and SPAM. NSP mining has seen only very limited progress in recent years [6][5][10][11], and all existing methods are very inefficient in mining NSP. This is because NSP is much more difficult to mine than PSP, in particular because of two intrinsic complexities.

High computational complexity. The existing methods calculate the support of negative sequential candidates (NSC) by additionally scanning the database after identifying PSP. This leads to additional costs and results in low efficiency in NSP mining.

Large NSC search space. The existing approaches generate k-size NSC by conducting a joining operation on (k-1)-size NSP. This leads to a huge number of NSC [6][5][7][10], which makes it difficult to search for meaningful outputs. Further, NSC does not satisfy the anti-monotony principle [10]. It is a challenge to prune the large proportion of meaningless and unnecessary NSC, and it is therefore important to develop efficient approaches for generating a limited number of truly useful NSC.

To address the above critical challenges in NSP mining and make NSP mining workable in handling real-life applications, this paper proposes a novel and efficient NSP mining approach called *e-NSP*. The main idea is as follows. To avoid additional database scanning, we convert the negative containment problem to a positive containment problem. The NSC supports are then calculated by only using a NSC's corresponding PSP information. In this way, there is no need to re-scan the database after discovering PSP. To the best of our knowledge, this is the first approach to conduct efficient NSP mining by involving PSP only, without rescanning the database. e-NSP provides a new and promising strategy for efficient NSP mining which is workable in large datasets.

The remainder of the paper is organized as follows. Section 2 discusses the related work. In Section 3, we formalize the problem of mining PSP and NSP. The e-NSP algorithm is detailed in Section 4. Section 5 presents the experiment results, which are followed by conclusions and future work in Section 6.

2. RELATED WORK

Very limited research is available in the literature on mining NSP. [10] proposes a negative version of GSP algorithm NegGSP to mine for NSP. [5] proposes a PNSP approach for mining positive and negative sequential patterns in the form of $<(abc)\ \neg(de)\ (ijk)>$. [6] only handles NSP with the last element as negative. [11] proposes a genetic algorithm for mining NSP. [9] proposes an approach to mining event-oriented negative sequential rules. [7] identifies NSP in the form of $(\neg a,b)$, $(a,\neg b)$ and $(\neg a,\neg b)$, which is similar to [9]. The above approaches either re-scan the database or do not directly address NSP mining. In addition, different researchers present inconsistent definitions and explanations about the basic concept of negative containment [4]. [5] considers that data sequence $ds=<d\ c>$ cannot contain $<\neg(ab)\ c\ \neg d>$, $<\neg c\ d>$,and $<c\ \neg d>$,while [10] allows that *ds* contains them.

3. PROBLEM STATEMENT

In association rule mining, a non-occurring item is called a *negative item* and an occurring item is called a *positive item* [8]. This tradition is followed in NSP mining. Those sequences consisting of at least one negative item are called *negative sequences*. Sequential pattern mining (as it is usually called) mainly focuses on occurring items, namely *positive sequences* [6][5][7][10]. The sequences in source data are called *data sequences*.

3.1 Positive Sequential Patterns - PSP

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of *items*. An *itemset* is a subset of I. A *sequence* is an ordered list of itemsets. A sequence s is denoted by $<s_1\ s_2\ \ldots\ s_l>$, where $s_j \subseteq I\ (1 \leqslant j \leqslant l)$. s_j is also called an *element* of the sequence, and denoted as $(x_1 x_2 \ldots x_m)$, where x_k is an item, $x_k \in I\ (1 \leqslant k \leqslant m)$. For simplicity, the brackets are omitted if an element only has one item, i.e., element (x) is coded x. An item can occur at most once in an element, but can appear multiple times in different elements of a sequence.

The *length* of sequence s, denoted as *length(s)*, is the total number of items in all elements in s. s is a k-length sequence if $length(s)=k$. The *size* of sequence s, denoted as $size(s)$, is the total number of elements in s. s is a k-size sequence if $size(s)=k$.

Sequence $s_\alpha=<\alpha_1\ \alpha_2\ \ldots\ \alpha_n>$ is called a *sub-sequence* of sequence $s_\beta=<\beta_1\ \beta_2\ \ldots\ \beta_m>$ and s_β is a *super-sequence* of s_α, denoted as $s_\alpha \subseteq s_\beta$, if there exists $1 \leqslant j_1 < j_2 < \ldots < j_n \leqslant m$ such that $\alpha_1 \subseteq \beta_{j_1}$, $\alpha_2 \subseteq \beta_{j_2}$, ..., $\alpha_n \subseteq \beta_{j_n}$. We also say that s_β contains s_α.

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 $\{<s>\}$, i.e., $sup(s)=|\{<s>\}|=|\{<sid,ds>, <sid,ds> \in D \wedge (s \subseteq ds)\}|$. *min_sup* is a minimum support threshold which is predefined by users. Sequence s is called a frequent *(positive) sequential pattern* if $sup(s) \geqslant min\_sup$.

3.2 Negative Sequential Patterns - NSP

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 [9][11][5][10][6]. This paper also refers to the constraints in these existing papers. The only difference is that we mine NSP only from frequent PSP. The reason is that PSP is most useful for users to make decisions so far.

In order to formalize the constraints, we provide a definition as following.

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 $ns=<s_1 \ldots s_k>$ is to change all negative elements in *ns* to their positive partners, denoted as $p(ns)$, i.e., $p(ns)=\{<s'_1 \ldots s'_k> \mid s'_i=p(s_i),\ s_i \in ns\}$. For example, $p(<\neg(ab)\ c\ \neg d>)=<(ab)\ c\ d>$.

Constraint 1. Frequency constraint. This paper only focuses on the negative sequences *ns* whose positive partner is frequent, i.e., $sup(p(ns)) \geqslant min\_sup$. While [5] and [10] only require the positive partner of each element in *ns* is frequent.

Constraint 2. Format constraint. Continuous negative elements in a NSC are not allowed. For example, $<\neg(ab)\ \neg c\ d>$ is not allowed. This constraint is same as [5][10].

Constraint 3. Element negative constraint. The minimum negative unit in a NSC is an element [10]. For example, $<(\neg ab)\ c\ d>$ does not satisfy this constraint, $<\neg(ab)\ c\ \neg d>$ does.

In this paper, we assume that a negative sequence implicitly satisfies the above three constraints.

Because of constraint 3, the definition of sub-sequence in positive sequence is not suitable for that in negative sequence. Now we formally redefine it by the definitions of element-id set and order preserving sequence. *Element id* is the order number of an element in a sequence. Given a sequence $s=<s_1\ s_2\ \ldots\ s_m>$, $id(s_i)=i$ is the element id 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., $EidS_s=\{(s_i,\ id(s_i)) \mid s_i \in s\}=\{(s_1,1),\ (s_2,2),\ \ldots,\ (s_m,m)\}\ (1 \leqslant i \leqslant m)$. The set including all positive/negative element-ids is called *positive/negative element-id set* of s, denoted as $EidS_s^+$, $EidS_s^-$ respectively.

For any subset $EidS'_s=\{(\alpha_1,id_1),\ (\alpha_2,id_2),\ \ldots,\ (\alpha_p,id_p)\}\ (1<p \leqslant m)$ of $EidS_s$, $\alpha=<\alpha_1\ \alpha_2\ \ldots\ \alpha_p>$, if $\forall\ \alpha_i,\ \alpha_{i+1} \in \alpha$

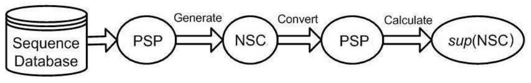

Figure 1: Framework of e-NSP

(1⩽i<p), there exist $id_i<id_{i+1}$, then α is called an *order-preserving sequence* of $EidS'_s$, denoted as α=*OPS*($EidS'_s$).

Sequence s_α is called a *sub-sequence* of negative sequence s_β, and s_β is a *super-sequence* of s_α, if $\forall EidS'_{s_\beta}$, $EidS'_{s_\beta}$ is subset of $EidS_{s_\beta}$, s_α=*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. Specially, the sub-sequence containing all positive elements, *OPS*($EidS^+_s$) is called the *Maximum Positive Sub-sequence* of s, denoted as *MPS(s)*.

Example 1. Given s=<¬(ab) c d>, $EidS^+$={(c,2), (d,3)}, *MPS(s)*=<c d>, *OPS*({(c,2),(¬(ab),1)})=<¬(ab) c> is a sub-sequence of s.

Definition 2. Negative Sequential Pattern (NSP)

A negative sequence s is a negative sequential pattern (N-SP) if its support is not less than the threshold *min_sup*.

4. E-NSP ALGORITHM

Figure 1 shows the framework and working mechanism of e-NSP. First, it mines all PSP by traditional PSP mining algorithm, then it generates NSC based on PSP, after that, it calculates supports of NSC by converting them to calculating support of corresponding PSP.

4.1 Negative Containment

Because a sub-sequence (e.g., s_1=<d>) may occur more than one times in its super-sequence (e.g., s_2=<a (bc) d (cde)>), we need to know the positions that s_2 contains s_1 from left and right sides of s_2. It is very important to our Negative Containment Definition. Therefore we give the following definition.

Definition 3. First Sub-sequence Ending Position / Last Sub-sequence Beginning Position

Given a data sequence ds=<d_1 d_2 ... d_t> and a positive sequence α, (1) if $\exists p$ *(1<p⩽t)*, $\alpha\subseteq$<d_1 ... d_p>$\wedge\alpha\nsubseteq$<d_1 ... d_{p-1}>, then p is called the *First Sub-sequence Ending Position*, denoted as *FSE(α,ds)*; if $\alpha\subseteq$<d_1> then *FSE(α,ds)=1*; (2) if $\exists q$ *(1⩽q<t)*, $\alpha\subseteq$<d_q ... d_t>$\wedge\alpha\nsubseteq$<d_{q+1} ... d_t>, then q is called the *Last Sub-sequence Beginning Position*, denoted as *LSB(α,ds)*; if $\alpha\subseteq$<d_t> then *LSB(α,ds)=t*; (3) if $\alpha\nsubseteq ds$, then *FSE(α,ds)=0*, *LSB(α,ds)=0*.

Example 2. Given ds=<a (bc) d (cde)>. *FSE(<a>,ds)=1*, *FSE(<c>,ds)=2*, *FSE(<c d>,ds)=3*, *LSB(<a>,ds)=1*, *LSB(<c>,ds)=4*, *LSB(<c d>,ds)=2*, *LSB(<(cd)>,ds)=4*.

Our definition of a data sequence containing a negative sequence is as follows. We use *n-neg-size* to denote a negative sequene containing n negative elements.

Definition 4. Negative Containment Definition

Let ds=<d_1 d_2 ... d_t> be a data sequence, ns=<s_1 s_2 ... 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 $p(ns)\nsubseteq ds$; (3) otherwise, *ds contains ns* if, $\forall(s_i,id(s_i)) \in EidS^-_{ns}$ *(1⩽i⩽ m)*, one of the following three holds:

(a) *(lsb=1)* or *(lsb>1)*$\wedge p(s_1)\nsubseteq$<d_1 ... d_{lsb-1}>, when *i=1*,

(b) *(fse=t)* or *(0<fse<t)*$\wedge p(s_m)\nsubseteq$<d_{fse+1} ... d_t>, when *i=m*,

(c) *(fse>0 ∧ lsb=fse+1)* or *(fse>0 ∧ lsb>fse+1)* ∧ $p(s_i)\nsubseteq$ <d_{fse+1} ... d_{lsb-1}>, when *1<i<m*,

where *fse=FSE(MPS(<s_1 s_2 ... s_{i-1}>),ds)*, *lsb=LSB(MPS(<s_{i+1} ... s_m>),ds)*.

Example 3. Given ds=<a (bc) d (cde)>, we have

(1) ns=<¬a c>. $EidS^-_{ns}$={(¬a,1)}. ds does not contain ns. $lsb=4>0$, but $p(s_1)$=<a>⊆<d_1 ... d_3>=<a (bc) d> (Case a).

(2) ns=<¬a a c>. $EidS^-_{ns}$={(¬a,1)}. ds contains ns because $lsb=1$ (Case a).

(3) ns=<(ab) ¬(cd)>. $EidS^-_{ns}$={(¬(cd),2)}. ds does not contain ns because $fse=0$ (Case b).

(4) ns=<(de) ¬(cd)>. $EidS^-_{ns}$={(¬(cd),2)}. ds contains ns because $fse=4 (t=4)$(Case b).

(5) ns=<a ¬d d ¬d>. $EidS^-_{ns}$={(¬d,2), (¬d,4)}. ds does not contain ns. For (¬d,2), *fse=1*, *lsb=4*, but $p(\neg d)\subseteq$ <d_2 ... d_3>=<(bc) d> (Case c). If one negative element does not satisfy the condition, we do not need to consider other negative elements.

(6) ns=<a ¬b b ¬a (cde)>. $EidS^-_{ns}$={(¬b,2), (¬a,4)}. ds contains ns. For (¬b,1), *fse=1*, *lsb=2*, *fse>0 ∧ lsb=fse+1* (Case c); For (¬a,4), *fse=2*, *lsb=4*, $p(\neg a)\nsubseteq$<d_3>=<d> (Case c).

4.2 Negative Conversion

In order to convert negative containment problems to positive containment problems, we need to define a special sub-sequence as follows.

Definition 5. 1-neg-size Maximum Sub-sequence

For a negative sequence ns, its sub-sequence that includes $MPS(ns)$ and one negative element e is called a *1-neg-size* maximum sub-sequence, 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*)| ∀ $e\in EidS^-_{ns}$}.

Corollary 1. Negative Conversion Strategy

Given a data sequence ds=<d_1 d_2 ... d_t>, and ns=<s_1 s_2 ... s_m>, which is an *m-size* and *n-neg-size* negative sequence, the negative containment definition can be converted as follows: data sequence ds contains negative sequence ns if and only if the two conditions hold: (1) $MPS(ns)\subseteq ds$; and (2) ∀ *1-negMS* ∈ *1-negMSS*$_{ns}$, *p(1-negMS)* $\nsubseteq ds$.

Example 4. Given ds=<a (bc) d (cde)>, 1) if ns=<a ¬d d ¬d>, *1-negMSS*$_{ns}$={<a ¬d d>, <a d ¬d>}, then ds does not contain ns because p(<a ¬d d>)=<a d d>⊆ ds; 2) if ns'=<a ¬b b ¬a (cde)>, *1-negMSS*$'_{ns}$={<a ¬b b (cde)>, <a b ¬a (cde)>}, then ds contains ns because *MPS(ns)*=<a b (cde)> ⊆ $ds\wedge$ p(<a ¬b b (cde)>) $\nsubseteq ds$ ∧ p(<a b ¬a (cde)>)$\nsubseteq ds$.

Corollary 1 proves that the problem of whether a data sequence contains a negative sequence is equivalent to the problem of whether the data sequence does not contain its corresponding positive sequences. The proof of Corollary 1 is omitted here because of limited space.

4.3 Support of Negative Sequence

Corollary 2.

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

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

This can be easily derived from Corollary 1. Because $\cup_{i=1}^{n}\{p(1\text{-}negMS_i)\} \subseteq \{MPS(ns)\}$, equation 1 can be rewritten as:

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

$$= sup(MPS(ns)) - \mid \cup_{i=1}^{n}\{p(1 - negMS_i)\} \mid \qquad (2)$$

Example 5. *sup(<¬a (bc) d ¬(cde)>)=sup(<(bc) d>) - |{<a (bc) d>}∪{<(bc) d (cde)>}|;*

If ns only contains a negative element

$$sup(ns) = sup(MPS(ns)) - sup(p(ns)) \qquad (3)$$

Example 6. *sup(<(ab) ¬c d>)=sup(<(ab) d>) - sup(<(ab) c d>)*

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

$$sup(<\neg e>) = \mid D \mid - sup(<e>) \qquad (4)$$

From equation 2 we can see that $sup(ns)$ can be easily calculated if we know $sup(MPS(ns))$ and $\mid \cup_{i=1}^{n}\{p(1\text{-}negMS_i)\} \mid$. According to Constraint (1) and the negative candidate generation approach discussed in Section 4.4, $MPS(ns)$ and $p(1\text{-}negMS_i)$ are frequent. So $sup(MPS(ns))$ can be easily obtained by traditional algorithms.

Now the problem is how to calculate $\mid \cup_{i=1}^{n}\{p(1\text{-}negMS_i)\} \mid$. Our approach is as follows.

We use a data structure, which is called e-NSP data structure, like Table 2 to store the corresponding data, including PSP, support and {*sid*} (containing all ids of the tuples that contain corresponding PSP). These data are stored in a hash table to identify PSP efficiently. In order to calculate the union set efficiently, we propose two other optimization methods as follows: (1) When we calculate support of a NSC, we also utilize a hash table to accelerate search speed. Compared with the performance using common array, the search speed with hash table is far more efficient. (2) We assume that all data in Table 2 are stored in main memory. This is feasible in practice since the mainstream memory can reach gigabytes and above. We do not record the *sid* of 1-size PSP because the equations do not need to calculate the union set of those *sid* of 1-size PSP.

4.4 Negative Sequential Candidates Generation

The basic idea of generating a negative sequential candidate is to change any non-contiguous elements (not items) in a PSP to their negative ones.

Definition 6. e-NSP Candidate Generation

For a k-size PSP, its NSC are generated by changing any m non-contiguous element(s) to its (their) negative one(s), $m=1,2,\ldots,\ulcorner k/2 \urcorner$, where $\ulcorner k/2 \urcorner$ is a minimum integer that is not less than $k/2$.

Example 7. The NSC based on *<(ab) c d>* include:

m=1, *<¬(ab) c d>*,*<(ab) ¬c d>*,*<(ab) c ¬d>*;

m=2, *<¬(ab) c ¬d>*.

Obviously, for all PSP in a sequence database, we can generate all NSC that satisfy the three constraints, as described in Section 3.2.

4.5 Algorithm Pseudocode

The e-NSP algorithm, as shown in Algorithm 1, is proposed to mine for NSP using only identified PSP.

Algorithm 1: e-NSP Algorithm

```
Input: Sequence Dataset $D$ and $min\_sup$;
Output: NSP;
$PSP$ = minePSP();
HashTable $PSPHash$ = CreatePSPHashTable($PSP$);
For (each $psp$ in $PSP$){
  $NSC$ = e-NSP_Candidate_Generation($psp$);
  For (each $nsc$ in $NSC$){
    if ($nsc$.size==1 && $nsc$.neg_size==1) {
      $nsc$.support = |D| - $p(nsc)$.support;
    } else if ($nsc$.size>1 && $nsc$.neg_size==1){
      $nsc$.support = $MPS(nsc)$.support - $p(nsc)$.support;
    } else {
      1-$negMSS_{nsc}$ = {1-$negMS_i$ | 1<=i<=$nsc$.neg_size};
      HashTable $cHash$ = new HashTable();
      For (i=1; i<=$nsc$.neg_size; i++) {
        For (each $sid$ in $p(1\text{-}negMS_i)$.sidSet) {
          If ($sid$.hashcode NOT IN $cHash$)
            $cHash$.put($sid$.hashcode(),$sid$);
        $nsc$.support=$MPS(nsc)$.support - $cHash$.size();
        }
        If ($nsc$.support > $min\_sup$) NSP.add($nsc$);
} } } }
return NSP;
```

Table 1: Example Data Set

Sid	Data Sequence
10	*<a b c>*
20	*<a (ab)>*
30	*<(ae) (ab) c>*
40	*<a a>*
50	*<d>*

Table 2: Example Result - Positive Patterns

PSP	Support	{sid}
<a>	4	-
<b>	3	-
<c>	2	-
<a a>	3	{20,30,40}
<a b>	3	{10,20,30}
<a c>	2	{10,30}
<b c>	2	{10,30}
<(ab)>	2	-
<a b c>	2	{10,30}
<a (ab)>	2	{20,30}

4.6 An Example

The sequence database is shown in Table 1 [5]. The process is as follows.

(1) Mine PSP using one of the well-known algorithms, such as GSP, and fill in the e-NSP data structures, which are shown as Table 2.

(2) Use the e-NSP Candidate Generation method to generate all NSC.

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

From this example, we can see that *<a c>* and *<a ¬c>*, *<a (ab)>* and *<a ¬(ab)>* are frequent patterns, but clearly not all of them can be used to make decisions because they may be misleading. How to select the meaningful and workable patterns is one of our ongoing tasks.

Table 3: Example Result - NSC and Support (min_sup=2)

PSP	NSC	Related PSP	Sup
$<a>$	$<\neg a>$	$<a>$	1
$<b>$	$<\neg \boldsymbol{b}>$	$<b>$	**2**
$<c>$	$<\neg \boldsymbol{c}>$	$<c>$	**3**
$<a\ a>$	$<\neg a\ a>$	$<a>$, $<a\ a>$	1
	$<a\ \neg a>$	$<a>$, $<a\ a>$	1
$<a\ b>$	$<\neg a\ b>$	$<b>$, $<a\ b>$	0
	$<a\ \neg b>$	$<a>$, $<a\ b>$	1
$<a\ c>$	$<\neg a\ c>$	$<c>$, $<a\ c>$	0
	$<\boldsymbol{a}\ \neg \boldsymbol{c}>$	$<a>$, $<a\ c>$	**2**
$<b\ c>$	$<\neg b\ c>$	$<c>$, $<b\ c>$	0
	$<b\ \neg c>$	$<b>$, $<b\ c>$	1
$<(ab)>$	$<\neg (\boldsymbol{ab})>$	$<(ab)>$	**3**
$<a\ (ab)>$	$<\neg a\ (ab)>$	$<(ab)>$, $<a\ (ab)>$	0
	$<\boldsymbol{a}\ \neg (\boldsymbol{ab})>$	$<a>$, $<a\ (ab)>$	**2**
$<a\ b\ c>$	$<\neg a\ b\ c>$	$<b\ c>$, $<a\ b\ c>$	0
	$<a\ \neg b\ c>$	$<a\ c>$, $<a\ b\ c>$	0
	$<a\ b\ \neg c>$	$<a\ b>$, $<a\ b\ c>$	1
	$<\neg a\ b\ \neg c>$	$<b>$, $<a\ b>$, $<b\ c>$	0

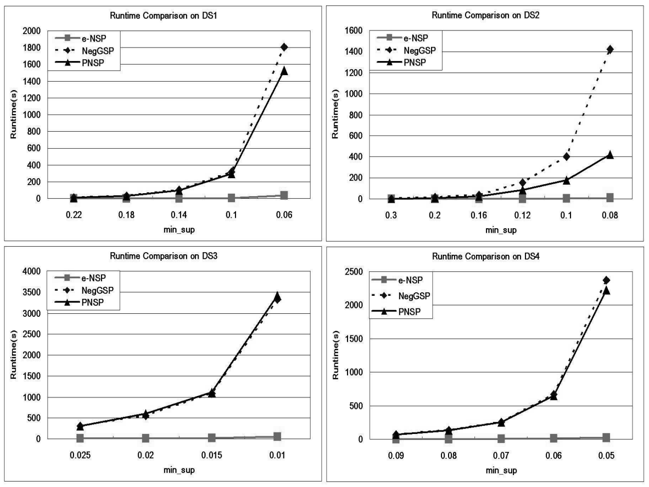

Figure 2: Runtime Comparison

5. EXPERIMENTS AND EVALUATION

We conduct experiments on 14 synthetic and real datasets to compare the efficiency of e-NSP with two baseline approaches PNSP [5] and NegGSP [10]. We select PNSP and NegGSP because they are the only available algorithms that are comparable to our algorithm. To compare their performance, we adapt PSNP and NegGSP to follow the same definitions and constraints as stated in Section 3. In the comparison, all positive patterns are identified by GSP. NSP are further mined by e-NSP, PNSP and NegGSP. We conduct intensive experiments to compare the difference between three algorithms in terms of computational costs on different data sizes and data characteristics.

All algorithms are implemented in Java in a PC with Intel Core 2 CPU of 2.9GHz, 2GB memory and Windows XP Professional SP2.

5.1 Data Sets

To describe and observe the impact of data characteristics on algorithm performance, we use following data factors: *C, T, S, I, DB* and *N*, which are defined to describe characteristics of sequence data [1].

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 maximal potentially large sequences; DB: Number of sequences (= size of Database); and N: Number of items.

Four source datasets are used for the experiments. They include both real data and synthetic datasets generated by IBM data generator [1]. By partitioning the data, we obtain 14 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), C10_T8_S20_I10_DB10k_N0.2k.

Dataset 3 (DS3) is from UCI Datasets. There are 989,818 records. The average number of elements in a sequence is 4, and each element only has one item.

Dataset 4 (DS4) is real-application data from financial service industry. It 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.

5.2 Computational Cost

The runtime of mining NSP by the three approaches is shown in Figure 2. e-NSP takes mostly less than 3% of the runtime of PNSP and NegGSP on all datasets. For example, e-NSP spends 2.7% to 1.6% runtime of PNSP on $DS3$ when *min_sup* decreased from 0.025 to 0.01. When *min_sup* is reduced to 0.01, PNSP and NegGSP take around one hour, but e-NSP takes less than one minute, because e-NSP only needs to "calculate" NSP support based on the *sid* sets of corresponding positive patterns, while PNSP and NegGSP have to re-scan the whole dataset.

The results on the maximum length and number of negative patterns are shown in Figure 3. It is difficult to draw a reliable conclusion from them, because the characteristics of the datasets are not comparable. Therefore, we conduct a dataset characteristics analysis in following section.

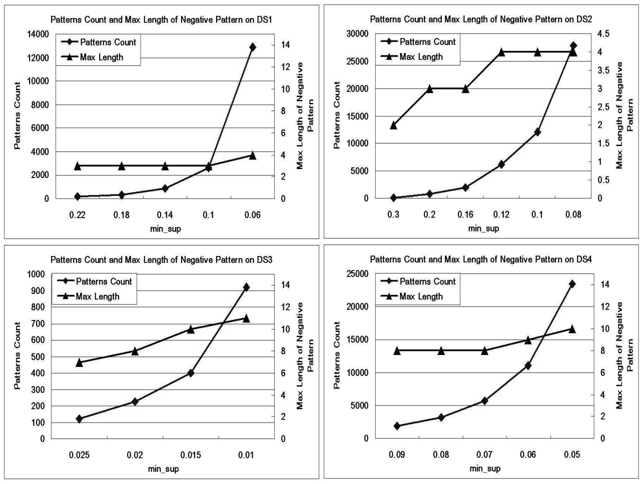

Figure 3: Maximum Length and Number of Negative Patterns

Table 4: Dataset Characteristics Analysis Result

ID	Dataset Characteristics	min sup	NGSP (t_1,s)	PNSP (t_2,s)	eNSP (t_3,s)	t_3/t_2
DS1	C8T4S6I6.DB10k.N100	0.04	1451.7	638.2	14.94	2.3%
		0.06	241.4	163.1	4.16	2.5%
		0.08	78.9	61.9	1.53	2.5%
DS1.1	**C4**T4S6I6.DB10k.N100	0.01	517.5	208.4	1.08	0.5%
		0.015	130.4	64.5	0.33	0.5%
		0.02	48.0	28.4	0.16	0.5%
DS1.2	**C12**T4S6I6.DB10k.N100	0.14	229.0	191.9	7.99	4.2%
		0.16	127.6	109.5	4.49	4.1%
		0.18	73.8	66.9	2.53	3.8%
DS1.3	C8**T8**S6I6.DB10k.N100	0.22	130.8	118.5	5.22	4.4%
		0.24	83.7	76.5	3.19	4.2%
		0.26	55.9	52.8	2.14	4.1%
DS1.4	C8**T12**S6I6.DB10k.N100	0.3	1205.2	969.3	57.55	5.9%
		0.4	133.2	123.5	6.75	5.5%
		0.5	23.6	23.0	1.06	4.6%
DS1.5	C8T4**S12**I6.DB10k.N100	0.04	1130.0	478.6	12.22	2.6%
		0.06	187.0	124.7	3.39	2.7%
		0.08	61.2	47.5	1.23	2.6%
DS1.6	C8T4**S18**I6.DB10k.N100	0.04	297.1	157.4	3.47	2.2%
		0.06	64.2	45.5	0.97	2.1%
		0.08	23.5	19.0	0.36	1.9%
DS1.7	C8T4S6**I10**.DB10k.N100	0.06	690.2	395.1	7.33	1.9%
		0.07	334.7	227.5	4.23	1.9%
		0.08	188.1	138.0	2.63	1.9%
DS1.8	C8T4S6**I14**.DB10k.N100	0.08	983.9	630.8	8.88	1.4%
		0.1	320.5	248.9	3.63	1.5%
		0.12	141.8	112.7	1.61	1.4%
DS1.9	C8T4S6I6.DB10k.**N200**	0.03	378.2	98.4	0.59	0.6%
		0.04	101.8	43.1	0.17	0.4%
		0.05	39.5	23.3	0.06	0.3%
DS1.10	C8T4S6I6.DB10k.**N400**	0.015	823.0	97.4	0.08	0.1%
		0.02	197.3	42.0	0.03	0.1%
		0.025	99.8	20.6	0.02	0.1%

5.3 Dataset Characteristics Analysis

We analyze the dataset characteristics in terms of the above defined data factors to see the impact of the data factors (see Section 5.1) on the performance of e-NSP, compared to PNSP and NegGSP. 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 4. For example, dataset *DS1.1*(C4T4S6I6. DB10k.N100) is different to *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 4.

In Table 4, *t1*, *t2* and *t3* represent the runtime of NegGSP, PNSP and e-NSP correspondingly. We use *t3/t2* to show e-NSP's performance compared with PNSP. From the results (see Table 4), we can say that factors C, T and N seriously affect the performance of e-NSP, and factors S and I do not greatly affect it. When factor C is low, such as *DS1.1*, e-NSP works better than on datasets with big C, such as *DS1* and *DS1.2*. Similar results hold for T, such as *DS1* with small T, compared with *DS1.3* and *DS1.4* with big T. When N is high, such as in *DS1.9* and *DS1.10*, e-NSP works better than that with small N, such as in *DS1*.

6. CONCLUSIONS AND FUTURE WORK

Mining NSP is very challenging due to the large search space of negative candidates. Current NSP techniques rely on re-scaning databases after identifying positive patterns. This has been shown to be very inefficient, and little progress has been made in NSP mining. We have proposed a simple but very efficient NSP mining algorithm: e-NSP. e-NSP is based on a formal and consistent concept, negative containment, which defines how a data sequence contains a negative sequence. e-NSP encloses a negative conversion strategy to convert the problem of whether a data sequence contains a negative sequence to the problem of whether the data sequence contains some of the corresponding positive sequences. Supports of NSC are then calculated based only on the corresponding PSP. Finally, a simple but efficient approach has been proposed to generate NSC. e-NSP has been tested on both synthetic and real-world datasets and compared with existing NSP mining algorithms. The experimental results and comparisons on 14 datasets from data characteristics perspectives have clearly shown that e-NSP is much more efficient than existing approaches. e-NSP offers a new strategy for efficiently mining large scale NSP.

We are currently working on effective approaches to select the most meaningful patterns, and the application of negative sequence mining on complex behavior analysis.

Acknowledgment

This work was supported by Projects of International Cooperation Training for Shandong Province Higher Educational Excellent Backbone Teacher, Shandong Provincial Natural Science Foundation, China (No.ZR2011FM028), and by Australian Research Council Grants (DP1096218, DP0988016, LP100200774, LP0989721).

7. REFERENCES

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE' 95*, pages 3–14, 1995.

[2] L. Cao. In-depth behavior understanding and use: the behavior informatics approach. *Information Science*, 180:3067–3085, 2010.

[3] L. Cao, Y. Zhao, and C. Zhang. Mining impact-targeted activity patterns in imbalanced data. *IEEE TKDE*, 20:1053–1066, 2008.

[4] X. Dong, L. Zhao, X. Han, and H. Jiang. Comparisons of several definitions about negative containment. In *ICCNT' 11*, pages 553–556, 2011.

[5] S.-C. Hsueh, M.-Y. Lin, and C.-L. Chen. Mining negative sequential patterns for e-commerce recommendations. In *APSCC '08. IEEE*, pages 1213–1218, 2008.

[6] N. P. Lin, H.-J. Chen, and W.-H. Hao. Mining negative sequential patterns. In *WSEAS' 07*, pages 654–658, 2007.

[7] W.-M. Ouyang and Q.-H. Huang. Mining negative sequential patterns in transaction databases. In *ICMLC' 07*, volume 2, pages 830–834, 2007.

[8] X. Wu, C. Zhang, and S. Zhang. Efficient mining of both positive and negative association rules. *ACM Trans. Inf. Syst.*, 22:381–405, July 2004.

[9] 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, pages 656–663. 2009.

[10] Z. Zheng, Y. Zhao, Z. Zuo, and L. Cao. Negative-gsp: An efficient method for mining negative sequential patterns. In *Data Mining and Analytics*, volume 101, pages 63–67. 2009.

[11] Z. Zheng, Y. Zhao, Z. Zuo, and L. Cao. An efficient ga-based algorithm for mining negative sequential patterns. In *PAKDD' 10*, volume 6118, pages 262–273. 2010.