On Incremental High Utility Sequential Pattern Mining

JUN-ZHE WANG and JIUN-LONG HUANG, Department of Computer Science, College of Computer Science, National Chiao Tung University

High utility sequential pattern (HUSP) mining is an emerging topic in pattern mining, and only a few algorithms have been proposed to address it. In practice, most sequence databases usually grow over time, and it is inefficient for existing algorithms to mine HUSPs from scratch when databases grow with a small portion of updates. In view of this, we propose the IncUSP-Miner$^+$ algorithm to mine HUSPs incrementally. Specifically, to avoid redundant re-computations, we propose a tighter upper bound of the utility of a sequence, called Tight Sequence Utility (TSU), and then we design a novel data structure, called the candidate pattern tree, to buffer the sequences whose TSU values are greater than or equal to the minimum utility threshold in the original database. Accordingly, to avoid keeping a huge amount of utility information for each sequence, a set of concise utility information is designed to be stored in each tree node. To improve the mining efficiency, several strategies are proposed to reduce the amount of computation for utility update and the scopes of database scans. Moreover, several strategies are also proposed to properly adjust the candidate pattern tree for the support of multiple database updates. Experimental results on some real and synthetic datasets show that IncUSP-Miner$^+$ is able to efficiently mine HUSPs incrementally.

CCS Concepts: • **Information systems** → **Data mining**;

Additional Key Words and Phrases: High utility sequential pattern mining, incremental mining, incremental high utility sequential pattern mining, utility mining

ACM Reference format:
Jun-Zhe Wang and Jiun-Long Huang. 2018. On Incremental High Utility Sequential Pattern Mining. *ACM Trans. Intell. Syst. Technol.* 9, 5, Article 55 (May 2018), 26 pages.
https://doi.org/10.1145/3178114

1 INTRODUCTION

Some recent studies have incorporated the concept of utility into classic sequential pattern mining [4, 7, 8, 12, 14, 15, 18], leading to the emergence of *high utility sequential pattern* (HUSP) mining [1, 19, 20]. In HUSP mining, the utility of a sequence represents its importance, which may be measured in terms of profit or other information valuable for users. For example, after finding that ⟨{Discrete Mathematics}, {Data Structures, Algorithms}, {Opeating Systems, Database Systems}⟩ is an HUSP, a bookstore can recommend "Operating Systems" and "Database Systems" to users who have bought "Discrete Mathematics," "Data Structures," and "Algorithms." Only a few

This work was supported in part by Ministry of Science and Technology, Taiwan, under Contracts MOST 104-2918-I-009-003, MOST 105-2221-E-009-160, MOST 105-2218-E-009-011, MOST 106-3114-E-009-011, and MOST 106-2221-E-009-152-MY3.
Authors' addresses: J.-Z. Wang and J.-L. Huang, Department of Computer Science, National Chiao Tung University, No. 1001, University Road, Hsinchu City, Taiwan 300; emails: {jzwang, jlhuang}@cs.nctu.edu.tw.

algorithms [1, 19] have been proposed to mine HUSPs, and HUSP mining has been applied to analyze purchase log, web-access log [2], and mobile commerce data [13].

However, in practice, a sequence database (SDB) is *incrementally* updated over time. For example, in a retail database, a series of transactions made by a new customer will make a new sequence *inserted* into the database, and those made by an existing customer will make a sequence *appended* to the corresponding sequence already in the SDB. When an SDB D is updated to D', a low utility sequence in D may become high utility in D'. In such cases, applying existing algorithms to mine HUSPs from scratch is obviously inefficient each time when the SDB is updated. Some challenges posed by the incremental HUSP mining are listed as below.

First, the utility of a sequence is not downward closed, meaning that the incremental frequent sequential pattern mining algorithms cannot be directly applied to find HUSPs. Although some algorithms have been proposed to incrementally mine high utility itemsets (HUIs), they cannot be applied to this problem either, since the HUSP mining is intrinsically more complex than HUI mining (e.g., the case of sequence appending is not considered in incremental HUI mining).

Second, a tree structure can be used to buffer the previous mining results and other information to avoid redundant re-computation when an SDB is updated. However, the tree structure will consume a lot of memory due to the huge number of generated tree nodes. Furthermore, a sequence may have multiple instances in one of its supersequences and each instance has its own utility. When the SDB is updated, updating the utility of each instance of each sequence buffered in the tree structure is time-consuming, thereby resulting in poor mining efficiency. Therefore, how to design a tighter upper bound of sequence utility for better tree node pruning and a concise tree node structure for efficient utility update is important.

Third, a simple way to identify the new HUSPs is to scan the whole database, which is very inefficient when the number of updated sequences is small. Therefore, how to design methods to reduce the scopes of database scans is crucial for efficient incremental HUSP mining.

Fourth, after a database update, how to adjust the tree structure for supporting subsequent database updates is important for the correctness of mining results, especially when incremental HUSP mining will be performed repeatedly.

In this article, we address all of the above challenges by proposing a new incremental HUSP mining algorithm named IncUSP-Miner$^+$. The major contributions of this article are summarized as follows:

(1) We propose a tight utility upper bound of a sequence, called TSU (standing for Tight Sequence Utility), to improve mining efficiency by eliminating more sequences not able to be high utility compared with existing HUSP mining algorithms.
(2) To facilitate incremental HUSP mining, we propose a novel data structure, called the *candidate pattern tree*, to maintain each sequence whose TSU is greater than or equal to the minimum utility threshold. Specifically, we design an efficient and compact tree node structure storing some auxiliary information of the corresponding sequence, so that the auxiliary information such as TSU and utility of the sequence can be quickly obtained without re-computation.
(3) Based on the proposed candidate pattern tree, several strategies are proposed to reduce (1) the amount of computation when a node has to be updated and (2) the scopes of database scans to further improve mining efficiency.
(4) In practice, databases are usually updated multiple times. Therefore, several strategies are proposed to correctly adjust the candidate pattern tree for supporting multiple database updates.

(5) We then propose algorithm IncUSP-Miner$^+$ to incrementally mine HUSPs based on TSU, the candidate pattern tree and the proposed strategies.
(6) Several experiments on various real and synthetic datasets are conducted to evaluate the performance of IncUSP-Miner$^+$. The experimental results show that IncUSP-Miner$^+$ outperforms the state-of-the-art HUSP mining algorithms when the database are updated.

The rest of this article is organized as follows. Section 2 gives the problem definitions and reviews the related work. Section 3 shows the preliminary concept, while Section 4 presents the details of algorithm IncUSP-Miner$^+$. Section 5 shows the experimental results, and finally, Section 6 concludes this article.

2 PROBLEM DEFINITION AND RELATED WORK

In this section, we first give the formal definition of incremental HUSP mining in Section 2.1, and then we review related studies on HUSP mining in Section 2.2.

2.1 Problem Definition

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of distinct items. A nonempty subset $X \subseteq I$ is called an itemset and $|X|$ represents the size of X. In this article, X is denoted as $[x_1, x_2, \ldots, x_k]$, where $x_j \in I$, for $1 \leq j \leq k$. For brevity, the brackets are omitted if an itemset contains only one item. Without loss of generality, all items within an itemset are assumed to be arranged lexicographically. A sequence $\langle E_1 E_2 \ldots E_m \rangle$ is an ordered list of itemsets, where $E_i \subseteq I$, $\forall\, i = 1, \ldots, m$. The *length* of sequence $\langle E_1 E_2 \ldots E_m \rangle$, denoted as l, is defined as $l \equiv \Sigma_{i=1}^{m} |E_i|$, and a sequence with length l is called an l-sequence. Sequence $t = \langle X_1 X_2 \ldots X_k \rangle$ is called a subsequence of another sequence $s = \langle Y_1 Y_2 \ldots Y_m \rangle$, denoted as $t \sqsubseteq s$, if there exists k integers $1 \leq j_1 < j_2 < \cdots < j_k \leq m$ such that $X_1 \subseteq Y_{j_1}$, $X_2 \subseteq Y_{j_2}, \ldots, X_k \subseteq Y_{j_k}$, and we say that s has an *instance* of t at position $\langle j_1, j_2, \ldots, j_k \rangle$. A sequence database D is a set of tuples (sid, s), where sid is the ID of sequence s. Each item i of the j-th itemset Y_j in a sequence s is associated with a positive real number $q(i, j, s)$, which is called the *internal utility* or quantity of item i with the jth itemset in sequence s. In the example SDB shown in Table 1(a), the internal utility of item a within the second itemset in s_2 (i.e., $q(a, 2, s_2)$) is 4. The *external utility* of item i, denoted as $p(i)$, is a positive real number that represents the unit profit or importance of i. The external utility of each item $i \in I$ is recorded in a profit table. Table 1(b) shows the corresponding profit table of the SDB shown in Table 1(a).

Definition 1. The utility of item i within the jth itemset of sequence s is defined as $u(i, j, s) = p(i) \times q(i, j, s)$. For example, $u(a, 2, s_2) = p(a) \times q(a, 2, s_2) = 3 \times 4 = 12$. The utility of itemset X contained in the jth itemset of sequence s is defined as

$$u(X, j, s) = \sum_{\forall i \in X \wedge X \subseteq Y_j} u(i, j, s). \tag{1}$$

For example, $u([a, c], 2, s_5) = u(a, 2, s_5) + u(c, 2, s_5) = 3 \times 5 + 1 \times 1 = 16$.

Definition 2. If s has an *instance* of t at position $\langle j_1, j_2, \ldots, j_k \rangle$, then the utility of the instance is defined as

$$u(t, \langle j_1, j_2, \ldots, j_k \rangle, s) = \sum_{i=1}^{k} u(X_i, j_i, s), \tag{2}$$

where $X_i \subseteq Y_{j_i}$. For example, $u(\langle a, c \rangle, \langle 1, 2 \rangle, s_2) = u(a, 1, s_2) + u(c, 2, s_2) = 3 + 6 = 9$.

Definition 3. The *utility* of sequence t in sequence s, denoted as $u(t, s)$, is defined as the *maximum* of the utilities of all instances of t in s. That is,

$$u(t, s) = \max\{u(t, \langle j_1, j_2, \ldots, j_k \rangle, s) | \forall \langle j_1, j_2, \ldots, j_k \rangle : t \sqsubseteq \langle Y_{j_1} Y_{j_2} \ldots Y_{j_k} \rangle\}. \tag{3}$$

Table 1. A Sequence Database and Its External Utility Table

(a) A Sequence Database

SID	**Original Database (D)**	**Incremental Part (Δdb)**
s_1	$[(a,5)(b,3)][(d,1)][(b,6)]$	
s_2	$[(a,1)(c,1)][(a,4)(c,6)][(c,2)][(e,1)][(d,2)]$	
s_3	$[(c,2)][(b,5)][(d,4)][(a,2)]$	
s_4	$[(e,5)][(c,2)]$	$[(c,2)(d,6)][(f,5)][(a,2)(e,5)]$
s_5	$[(a,1)(c,1)][(a,5)(c,1)(e,1)]$	$[(a,5)(c,1)(e,1)][(d,1)(e,2)]$
s_6		$[(f,2)][(b,1)][(e,4)][(d,2)][(c,1)]$

(b) An External Utility Table

Item	**External Utility**
a	3
b	4
c	1
d	2
e	1
f	2

For example, we have $u(\langle a,c\rangle, s_2) = \max\{u(\langle a,c\rangle, \langle 1,2\rangle, s_2), u(\langle a,c\rangle, \langle 1,3\rangle, s_2), u(\langle a,c\rangle, \langle 2,3\rangle, s_2)\} = \max\{9,5,14\} = 14$.

Definition 4. The utility of a sequence t in D, denoted as $u_D(t)$, is defined as

$$u_D(t) = \sum_{\forall s \in D \wedge t \sqsubseteq s} u(t,s). \tag{4}$$

For example, $u_D(\langle a,c\rangle) = u(\langle a,c\rangle, s_2) + u(\langle a,c\rangle, s_5) = 14 + 4 = 18$.

Definition 5. A sequence t is called a *high utility sequential pattern* in a sequence database D if $u_D(t) \geq min\_util$, where min_util is a user-specified minimum utility threshold. Suppose $min\_util = 20$, $\langle a,c\rangle$ is not a *high utility sequential pattern* in D, since $u_D(\langle a,c\rangle) = 18 < 20$.

For an original sequence database D, the following two types of operations: (1) sequence insertion or (2) sequence appending will update D. To facilitate the following discussions, we have the following definitions.

Definition 6 (Inserted Sequence). When a new sequence t is inserted into D, t is called an *inserted sequence* of D. In the example shown in Table 1(a), sequence $s_6 = \langle f,b,e,d,c\rangle$ is an inserted sequence in the SDB.

Definition 7 (Appended Sequence). Given two sequences $t = \langle Y_1 Y_2 \ldots Y_m\rangle$ and $t_a = \langle Z_1 Z_2 \ldots Z_k\rangle$ $s.t.$ $m > 0$ and $k > 0$, sequence $t \cdot t_a$ is defined as $\langle Y_1 Y_2 \ldots Y_m Z_1 Z_2 \ldots Z_k\rangle$, meaning that $t \cdot t_a$ is formed by appending t_a to the end of t. When $t \in D$, t' is called an *appended sequence* of D if $t' = t \cdot t_a$. In the example database in Table 1(a), appending a sequence $\langle [c,d] f [a,e]\rangle$ to sequence $s_4 = \langle e,c\rangle$ updates sequence s_4 to $s_4' = \langle e, c[c,d] f [a,e]\rangle$.

Definition 8 (Updated Sequence). A sequence t is called an *updated sequence* of D if t is either an inserted sequence or an appended sequence of D.

Definition 9 (Incremental Part). The incremental part, denoted as Δdb, of D is the set of the sequences where each sequence, say s, in Δdb satisfies one of the following two criteria.

(1) s is an inserted sequence of D, or
(2) there exists a sequence, say t, in D where $t \cdot s$ is an appended sequence of D.

Considering the example in Table 1(a), the left column represents the original database D and the right column is the incremental part Δdb. After D is updated by Δdb, the updated database is denoted as D'.

For a sequence t, its utility may increase due to database updates (i.e., $u_D(t) \neq u_{D'}(t)$), and the possible cases of utility increase are as below.

- Given an inserted sequence s' in D' and $t \sqsubseteq s'$, it is obvious that the insertion of s' will make t's utility increase by $u(t, s')$. Considering the inserted sequence s_6 in Table 1(a), since $\langle b, d\rangle \sqsubseteq s_6$, s_6 will make the utility of $\langle b, d\rangle$ increase by $u(\langle b, d\rangle, s_6) = 8$.
- Given an appended sequence $s' \in D'$ where $s' = s \cdot s_a$ and $s \in D$, if $t \not\sqsubseteq s$ but $t \sqsubseteq s'$, the update of s will make t's utility increase by $u(t, s')$. In the example shown in Table 1(a), since sequence $\langle f\rangle \not\sqsubseteq s_4$ in D but $\langle f\rangle \sqsubseteq s_4'$ in D', the update of s_4 will make the utility of $\langle f\rangle$ increase by $u(\langle f\rangle, s_4') = 10$.
- Given an appended sequence $s' \in D'$ where $s' = s \cdot s_a$ and $s \in D$, if $t \sqsubseteq s$ and s' has at least one new instance of t at position $\langle j_1, j_2, \ldots, j_k\rangle$ and $u(t, \langle j_1, j_2, \ldots, j_k\rangle, s') > u(t, s)$, $u(t, s')$ is greater than $u(t, s)$ and the update of s will make t's utility increase by $u(t, s') - u(t, s)$. For example, sequence s_5' has two new instances of sequence $\langle a, c\rangle$ at positions $\langle 1, 3\rangle$ and $\langle 2, 3\rangle$ whose utilities are 4 and 16, respectively. Since $\max\{u(\langle a, c\rangle, \langle 1, 3\rangle, s_5'), u(\langle a, c\rangle, \langle 2, 3\rangle, s_5')\} = 16 > u_D(\langle a, c\rangle, s_5) = 4$, the update of s_5' will make the utility of $\langle a, c\rangle$ increase by $16 - 4 = 12$.

It is obvious that some sequences of insufficient utility in D may become of high utility sequences in D'. Thus, the problem of incremental HUSP mining can be formulated as below.

Problem Statement. Given an original SDB D, an increment SDB Δdb, an external utility table, the minimum utility threshold *min_util*, and the set of high utility sequential patterns in D, the problem of *incremental high utility sequential pattern mining* is to identify all high utility sequential patterns in the updated SDB D' without mining the whole D'.

2.2 Related Work

The problem of HUSP mining was first addressed in Reference [1]. Due to the absence of downward closure property in sequence utility, Sequence-Weighted Utility (SWU), which is of downward closure property and is not less than the utility of a sequence, was designed to prune the search space. Based on SWU, Ahmed et al. proposed in Reference [1] an Apriori-based algorithm UL and a PrefixSpan-based algorithm US to mine HUSPs by the following two phases. In phase I, the sequences with high SWU are first found from the SDB. Then, in phase II, the actual utilities of all high SWU sequences are computed and hence all HUSPs can be identified. However, subsequent studies [9, 19] argued that the definition of utility used in Reference [1] is too specific, and hence adopted "the maximum utility of all occurrences of t in s" as the utility of t in s. Our work follows the definition of utility used in References [9, 19]. In Reference [19], Yin et al. gave a generic problem definition of HUSP mining. They modeled the search space of HUSP mining problem as a lexicographic tree, and proposed algorithm USpan to mine HUSPs by traversing the tree in a DFS manner. USpan also utilized SWU to generate candidate sequences. Although SWU is able to prune the search space, the algorithms adopting SWU (e.g., UL [1], US [1], and USpan [19]) usually suffer from the problem of massive candidate sequences generated especially when minimum utility threshold is small. In view of this, Wang et al. proposed in Reference [17] two tight utility upper bounds, named PEU and RSU, as well as several companion pruning strategies to generate less candidates, thereby achieving efficient HUSP mining.

However, in practice, databases always grow over time, arising the need of incremental mining. Ahmed et al. [3] devised three tree structures based on FP-tree to facilitate incremental high utility itemset mining. The first tree is designed to store items based on their lexicographic order and hence a new transaction can be inserted into the tree without adjustment. The second tree is designed to reduce the size of the tree by storing items based on their occurrence frequencies in descending order. The third tree, which stores items based on their TWU values in descending order, is able to achieve better mining efficiency. However, these tree structures and the algorithms proposed in Reference [3] cannot be applied to incremental HUSP mining.

Some prior studies have focused on incremental mining of frequent sequential patterns. ISM [11] maintains a sequence lattice that stores all frequent sequences and all sequences in the negative border for incremental mining. IncSpan algorithm [6] buffers a set of frequent and almost frequent sequences in a tree structure to increase the mining efficiency. However, Nguyen et al. claimed in Reference [10] that IncSpan is not complete and proposed an rectified algorithm of IncSpan, called IncSpan+, to find updated frequent sequences. Although IncSpan+ is able to find the complete set of frequent sequences after the SDB is updated, it fails to support multiple updates of an SDB. PBIncSpan [5] uses a prefix tree to maintain all frequent subsequences, and designed width and depth pruning strategies to reduce the search space.

All of the aforementioned algorithms are designed for mining HUSPs from a static database, incremental mining of high utility itemsets, or incremental mining of frequent sequential patterns. No effort has been put to incremental HUSP mining. To the best of our knowledge, our preliminary work [16] is the first study on incremental mining of HUSPs. This article extends our preliminary work [16] by (1) proposing several strategies to reduce the scopes of database scans, (2) proposing several strategies to support multiple database updates, and (3) providing theoretical proofs of the proposed TSU and strategies. Experimental results show that the algorithm proposed in this article (i.e., IncUSP-Miner$^+$) outperforms the algorithm proposed in Reference [16] (i.e., IncUSP-Miner) in terms of mining efficiency.

3 PRELIMINARY CONCEPT

3.1 Sequence Extension

Similar to References [4, 19], the search space of the HUSP mining problem can be represented as a lexicographic tree defined as follows.

Definition 10 ([4, 19]). A lexicographic tree is a tree structure with the root representing the empty sequence $\langle\rangle$. In a lexicographic tree, a child node, say t', of a node t is called an *extension sequence* of t and t is called the *parent sequence* of t'. Consider an l-sequence t and let the last item within t's last itemset be i_e. An extension sequence of t, say t', is an $(l+1)$-sequence that should be one of the following two types.

- *S-extension sequence*: t' is called an s-extension sequence of t if t' is generated by appending a new itemset consisting of a single item to the end of t.
- *I-extension sequence*: t' is called an i-extension sequence of t if it is formed by inserting an item, which is lexicographically greater than i_e, into the last itemset of t.

$$s_2 = \langle[(a,1),(c,1)][(a,4),(c,6)][(c,2)][(e,1)][(d,2)]\rangle$$

Instance #1: $\langle[(a,1)]\qquad\qquad\quad [(c,6)]\rangle$

Instance #2: $\langle[(a,1)]\qquad\qquad\qquad\qquad [(c,2)]\rangle$

Instance #3: $\langle\qquad\qquad\quad [(a,4)]\qquad [(c,2)]\rangle$

Fig. 1. An example of remaining sequence and remaining utility.

The process of generating an s-extension sequence is called *S-Extension*, and the process of generating an i-extension sequence is called *I-Extension*. Furthermore, if a sequence t_d can be formed by a series of I- or S-Extensions from another sequence t, t_d is called a *descendant sequence* of t. For example, $\langle a, d\rangle$ is an s-extension sequence of $\langle a\rangle$, and $\langle[a, f]\rangle$ is an i-extension sequence of $\langle a\rangle$.

To facilitate the following discussions, we give the definitions of *maximum utility* and *remaining utility* as below.

Definition 11. If $s = \langle Y_1 Y_2 \dots Y_m\rangle$ has an instance of t at position $\langle j_1, j_2, \dots, j_k\rangle$, then the last item within the last itemset in t, say item i, is called the *extension item* of t, and j_k is called an *extension position* of t in s. For example, item c is the extension item of sequence $\langle a, c\rangle$, while 2 and 3 are the extension positions of $\langle a, c\rangle$ in sequence $s_2 = \langle[(a,1)(c,1)][(a,4)(c,6)][(c,2)][(e,1)][(d,2)]\rangle$.

Definition 12. The *maximum utility* of t at extension position p in s, denoted as $u(t, p, s)$, is defined as

$$u(t,p,s) = \max\{u(t,\langle j_1, j_2, \dots, p\rangle, s) | \forall 1 \le j_1 < j_2 < \dots < p \le m : t \sqsubseteq \langle Y_{j_1} Y_{j_2} \dots Y_p\rangle\}, \quad (5)$$

which is the maximum of the utilities of all instances of t whose extension positions are p. For example, $u(\langle a, c\rangle, 3, s_2) = \max\{u(\langle a, c\rangle, \langle 1, 3\rangle, s_2), u(\langle a, c\rangle, \langle 2, 3\rangle, s_2)\} = \max\{5, 14\} = 14$.

Definition 13. Given a sequence s, the *remaining sequence* of item i within the j_kth itemset in s, denoted as $rs(i, j_k, s)$, is defined as the subsequence from the item after item i within the j_kth itemset in s to the last item of s. The *remaining utility* of a remaining sequence $rs(i, j_k, s)$, denoted as $ru(i, j_k, s)$, is the sum of the utilities of all items in $rs(i, j_k, s)$. In the example shown in Table 1(a), given a sequence $\langle a, a\rangle \sqsubseteq s_2$, the remaining sequence $rs(a, 2, s_2)$ is $\langle c, c, e, d\rangle$ and the remaining utility of $rs(a, 2, s_2)$ is $ru(a, 2, s_2) = u(c, 2, s_2) + u(c, 3, s_2) + u(e, 4, s_2) + u(d, 5, s_2) = 13$.

We use the example shown in Figure 1 to illustrate the concept of remaining sequence and remaining utility. As mentioned in Section 2.1, all items in an itemset are sorted lexicographically. The extension item of $\langle a, c\rangle$ is item c. As shown in Figure 1, $\langle a, c\rangle$ has three instances in s_2 and the positions of item c in these three instances are 2, 3, and 3, respectively. Thus, the extension positions of $\langle a, c\rangle$ in s_2 are 2 and 3. According to Definition 13, the remaining sequence $rs(c, 2, s_2) = \langle[(c,2)][(e,1)][(d,2)]\rangle$ and the remaining utility $ru(c, 2, s_2)$ equals to $2 \times 1 + 1 \times 1 + 2 \times 2 = 7$. Similarly, the remaining sequence $rs(c, 3, s_2) = \langle[(e,1)][(d,2)]\rangle$ and the remaining utility $ru(c, 3, s_2)$ equals to $1 \times 1 + 2 \times 2 = 5$.

3.2 Utility Upper Bounds

Two utility upper bounds, Prefix-Extension Utility (abbreviated as PEU) and Reduced Sequence Utility (abbreviated as RSU) are proposed in Reference [17] to facilitate efficient mining of HUSPs. To better mining efficiency, we propose in this subsection a new utility upper bound, named Tight Sequence Utility (abbreviated as TSU), which is tighter than RSU, and then use PEU and TSU to design our incremental HUSP mining algorithm, algorithm IncUSP-Miner$^+$, in Section 4.

3.2.1 *The Prefix Extension Utility.*

Definition 14 ([17]). The PEU of sequence t in sequence s at position p with extension item i, denoted as $PEU(t,p,s)$, is defined as

$$PEU(t,p,s) = \begin{cases} u(t,p,s) + ru(i,p,s) & \text{if } ru(i,p,s) > 0, \\ 0 & \textit{otherwise}. \end{cases} \tag{6}$$

For example, we have $PEU(\langle a\rangle, 2, s_2) = 12 + 13 = 25$ and $PEU(\langle a\rangle, 4, s_3) = 0$.

The PEU of sequence t in sequence s, denoted as $PEU(t,s)$, is defined as

$$PEU(t,s) = \max\{PEU(t,p,s) | \forall \text{extension position } p \text{ of } t\}. \tag{7}$$

For example, we have $PEU(\langle a,c\rangle, s_2) = \max\{PEU(\langle a,c\rangle, 2, s_2), PEU(\langle a,c\rangle, 3, s_2)\} = \max\{9 + 7, 14 + 5\} = \max\{16, 19\} = 19$.

The PEU of sequence t in D, denoted as $PEU(t)$, is defined as

$$PEU_D(t) = \sum_{\forall s \in D \wedge t \sqsubseteq s} PEU(t,s). \tag{8}$$

For example, $PEU_D(\langle a,c\rangle) = PEU(\langle a,c\rangle, s_2) + PEU(\langle a,c\rangle, s_5) = 19 + 5 = 24$.

Let $u_D(t)$ and $SWU_D(t)$ be, respectively, the utility and SWU of t in D. The following theorem is proved in Reference [17].

THEOREM 1 ([17]). *Given a sequence t, $PEU_D(t) \leq SWU_D(t)$. In addition, for each descendant sequence t' of t, $u_D(t') \leq PEU_D(t)$.*

According to Theorem 1, PEU is an tighter upper bound of the utility of a sequence than SWU.

3.2.2 *The Tight Sequence Utility.*

In this subsection, a novel upper bound of the utility of a sequence, named TSU, is defined as follows to facilitate the design of our incremental HUSP mining algorithm.

Definition 15 (TSU for 1-sequences). Consider a sequence $s \in D$ and a 1-sequence t whose extension item is i. Let p_1 be the smallest extension position of t in s. The TSU of t in sequence s, denoted as $TSU(t,s)$, is defined as

$$TSU(t,s) = u(i,p_1,s) + ru(i,p_1,s). \tag{9}$$

Considering the example in Table 1(a), sequence $\langle c\rangle$ has three extension positions in s_2, which are 1, 2, and 3, respectively. By Definition 15, $TSU(\langle c\rangle, s_2) = u(c,1,s_2) + ru(c,1,s_2) = 1 + (12 + 6 + 2 + 1 + 4) = 26$.

Definition 16 (TSU for l-sequences where $l \geq 2$). Consider an l-sequence t whose extension item is item i. Suppose t is an extension sequence of an $(l-1)$-sequence α. For a sequence $s \in D$ and $t \sqsubseteq s$, let p_1 be the smallest extension position of t in s and ap_1 be the smallest extension position of α in s. The TSU of t in sequence s, denoted as $TSU(t,s)$, is defined as

$$TSU(t,s) = \begin{cases} u(\alpha, ap_1, s) + u(i,p_1,s) + ru(i,p_1,s) & \text{if } u(\alpha,s) = u(\alpha,ap_1,s) \\ PEU(\alpha,s) & \textit{otherwise}. \end{cases} \tag{10}$$

Example 3.1. For example, sequence $\langle [a,c]c\rangle$ is a subsequence of $s_2 = \langle [(a,1)(c,1)][(a,4)(c,6)] [(c,2)][(e,1)][(d,2)]\rangle$. The smallest extension position of its parent sequence, $\langle [a,c]\rangle$, in s_2 is 1. Since $u(\langle [a,c]\rangle, s_2) = 18$ is not equal to $u(\langle [a,c]\rangle, 1, s_2) = 4$, by definition, we have $TSU(\langle [a, c]c\rangle, s_2) = PEU(\langle [a,c]\rangle, s_2) = 29$. Consider another sequence $\langle a,d\rangle$ contained in $s_1 = \langle [a,b]d,b\rangle$.

Since the extension positions of all the instances of $\langle a\rangle$ are 1, it is obvious that $u(\langle a\rangle, s_1) = u(\langle a\rangle, 1, s_1) = 15$. By definition, we have $TSU(\langle a, d\rangle, s_1) = u(\langle a\rangle, 1, s_1) + u(d, 2, s_1) + ru(d, 2, s_1) = 15 + 2 + 24 = 41$.

Lemma 1. *Given two sequences s and t where $t \sqsubseteq s$, $TSU(t, s) \le RSU(t, s)$ holds.*

Proof. Let the length of t be l, the extension item of t be i, and extension positions of t in s be $p_1, p_2, \ldots, p_k$ where $p_1 < p_2 < \cdots < p_k$. We prove this lemma by considering the following two cases.

- *The case that l=1:*
  Let α be the parent sequence of t. Since t is a 1-sequence, i is the only item of t and α is the empty sequence $\langle\rangle$. According to Reference [17], $RSU(t, s) = PEU(\alpha, s) = PEU(\langle\rangle, s)$, where $PEU(\langle\rangle, s)$ is the sum of the utilities of all items in s. According to Definition 15, $TSU(t, s) = u(i, p_1, s) + ru(i, p_1, s) \le PEU(\langle\rangle, s) = RSU(t, s)$.
- *The case that $l \ge 2$:*
  Let α be the parent sequence of t with length $l - 1$ and the extension positions of α in s be $ap_1, ap_2, \ldots, ap_{ak}$ where $ap_1 < ap_2 < \cdots < ap_{ak}$. We prove this case by considering the following two conditions.
  - Condition that $u(\alpha, s) \ne u(\alpha, ap_1, s)$:
    When $u(\alpha, s) \ne u(\alpha, ap_1, s)$, by Definition 16, we have $TSU(t, s) = PEU(\alpha, s)$. According to Reference [17], $RSU(t, s) = PEU(\alpha, s)$. As a result, we have $TSU(t, s) = PEU(\alpha, s) = RSU(t, s)$.
  - Condition that $u(\alpha, s) = u(\alpha, ap_1, s)$:
    Since t is an extension sequence of α, we have $ap_1 \le p_1$. In addition, the instance of i at position p_1 is in $rs(i, ap_1, s)$ and $rs(i, p_1, s)$ is a subsequence of $rs(i, ap_1, s)$. Thus, $u(i, p_1, s) + ru(i, p_1, s) \le ru(i, ap_1, s)$. Therefore, we can derive the following equation:

$$\begin{aligned} TSU(t, s) &= u(\alpha, ap_1, s) + u(i, p_1, s) + ru(i, p_1, s) \\ &\le u(\alpha, ap_1, s) + ru(i, ap_1, s) \\ &\le \max_{1 \le j \le ak}(u(\alpha, ap_j, s) + ru(i, ap_j, s)). \end{aligned} \tag{11}$$

    According to Reference [17], we have $RSU(t, s) = PEU(\alpha, s) = \max_{1 \le j \le ak}(u(\alpha, ap_j, s) + ru(i, ap_j, s))$. By combining the above two equations, we can derive $TSU(t, s) \le RSU(t, s)$.

Finally, this lemma can be proved by the proofs of the above two cases. □

Lemma 2. *Given two sequences s and t where $t \sqsubseteq s$, $PEU(t, s) \le PEU(\alpha, s)$ holds where α is the parent sequence of t.*

Proof. Let the extension positions of t in s be $p_1, p_2, \ldots, p_k$, where $p_1 < p_2 < \cdots < p_k$ and the extension item of t be i. Assume that $u(t, p_{jmax}, s) + ru(i, p_{jmax}, s)$ is the largest among all $u(t, p_j, s) + ru(i, p_j, s)$, where $1 \le j \le k$. Let $inst_t$ be the instance of t at extension position p_{jmax} whose utility is $u(t, p_{jmax}, s)$. By definition, $PEU(t, s) = u(t, p_{jmax}, s) + ru(i, p_{jmax}, s)$. Let the extension positions of α in s be $ap_1, ap_2, \ldots, ap_{ak}$, where $ap_1 < ap_2 < \cdots < ap_{ak}$. We can get an instance of α, denoted as $inst_\alpha$, by removing the latest item i from $inst_t$. Let the extension item of α be i_α and the extension position of $inst_\alpha$ be ap_{aj} where $1 \le aj \le ak$ and $ap_{aj} \le p_{jmax}$. By definition, the utility of $inst_\alpha$ equals $u(\alpha, ap_{aj}, s)$.

It is clear that $inst_t$'s utility is equal to $inst_\alpha$'s utility plus the utility of i at position p_{jmax}. Therefore, we have $u(t, p_{jmax}, s) = u(\alpha, ap_{aj}, s) + u(i, p_{jmax}, s)$. Since $ap_{aj} \le p_{jmax}$, the instance of i at position p_{jmax} is in $rs(i_\alpha, ap_{aj}, s)$ and $rs(i, p_{jmax}, s)$ is a subsequence of $rs(i_\alpha, ap_{aj}, s)$. Thus, we have $u(i, p_{jmax}, s) + ru(i, p_{jmax}, s) \le ru(i_\alpha, ap_{aj}, s)$. Finally, we can prove this lemma by the following equation:

$$\begin{aligned} PEU(t,s) &= u(t,p_{jmax},s) + ru(i,p_{jmax},s) \\ &= u(\alpha,ap_{aj},s) + u(i,p_{jmax},s) + ru(i,p_{jmax},s) \\ &\le u(\alpha,ap_{aj},s) + ru(i_\alpha,ap_{aj},s) \\ &\le \max_{1\le j\le ak}(u(\alpha,ap_j,s) + ru(i_\alpha,ap_j,s)) \\ &\le PEU(\alpha,s). \end{aligned} \tag{12}$$ □

Lemma 3. *Given two sequences s and t where $t \sqsubseteq s$, $u(t,s) \le TSU(t,s)$ holds. In addition, $u(t',s) \le TSU(t,s)$ holds for each t's extension sequence t'.*

Proof. Please see Appendix A for the proof. □

Definition 17. The TSU of a sequence t in the sequence database D, denoted as $TSU_D(t)$, is defined as

$$TSU_D(t) = \sum_{\forall s\in D \wedge t\sqsubseteq s} TSU(t,s). \tag{13}$$

Consider the SDB shown in Table 1(a). Since $\langle[a,c]c\rangle$ has instances only in s_2 and s_5, we have $TSU_D(\langle[a,c]c\rangle) = TSU(\langle[a,c]c\rangle,s_2) + TSU(\langle[a,c]c\rangle,s_5) = 29 + 21 = 50$.

Theorem 2. *Given a sequence t and a sequence database D, $u_D(t) \le TSU_D(t) \le RSU_D(t) \le SWU_D(t)$ holds. In addition, for each descendant sequence t' of t, $u_D(t') \le TSU_D(t)$ also holds.*

Proof. $RSU_D(t) \le SWU_D(t)$ has been proven in Reference [17]. According to Lemmas 1 and 3, we have $u(t,s) \le TSU(t,s) \le RSU(t,s)$ for each sequence s where $t \sqsubseteq s$. By the definitions of $u_D(t)$, $TSU_D(t)$ and $RSU_D(t)$, it is obvious that $u_D(t) \le TSU_D(t) \le RSU_D(t)$.

In addition, according to Lemma 3, we have $u(t',s) \le TSU(t,s)$. Since t' is a descendant sequence of t, $t' \sqsubseteq s$ implies $t \sqsubseteq s$. Then, we have

$$\begin{aligned} u_D(t') &= \textstyle\sum_{\forall s\in D\wedge t'\sqsubseteq s} u(t',s) \\ &\le \textstyle\sum_{\forall s\in D\wedge t'\sqsubseteq s} TSU(t,s) \\ &\le \textstyle\sum_{\forall s\in D\wedge t\sqsubseteq s} TSU(t,s) \\ &\le TSU_D(t). \end{aligned} \tag{14}$$ □

According to Theorem 2, we conclude that TSU is a tighter upper bound of the sequence utility than RSU, and we will use TSU and PEU to design algorithm IncUSP-Miner$^+$ in Section 4.

4 THE PROPOSED APPROACH

4.1 Overview

Figure 2 shows the architecture of the proposed incremental HUSP mining approach. Our approach consists of two phases, the initial phase and the incremental phase. In the initial phase, we propose algorithm USP-Miner incorporating TSU and PEU to mine HUSPs on the origin database D and keep the HUSPs in the list $HUSP$. To facilitate better performance in the incremental phase, we propose the concept of the *candidate pattern tree* and in addition to HUSPs, USP-Miner will construct a candidate pattern tree T as well.

After Δdb is inserted into D, algorithm IncUSP-Miner$^+$ is employed to incrementally mine the HUSPs in D' based on the tree T in the incremental phase. The basic idea of IncUSP-Miner$^+$ is to traverse each node in T from the root in a depth-first manner and skip those nodes not appearing in the updated sequences. Since some sequences of insufficient utility in D may become of high utility in D', IncUSP-Miner$^+$ needs to adjust T to get correct mining results. In addition, some sequences of insufficient utility in D may be still not of high utility in D'. Therefore, we also design some

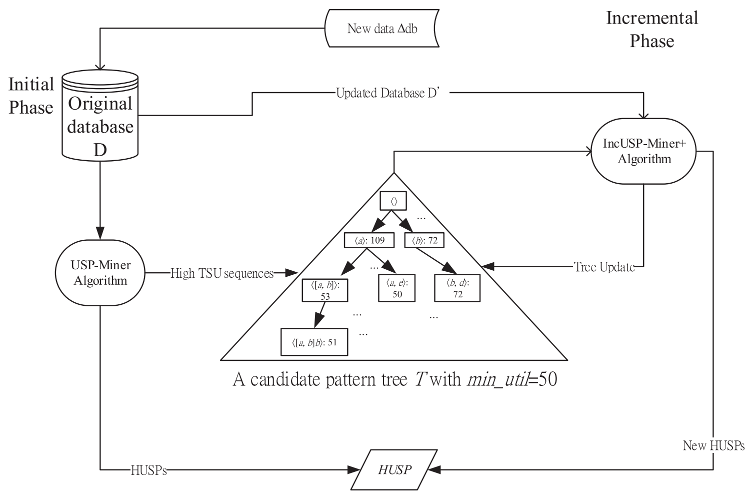


Fig. 2. Architecture of the proposed approach.

strategies to identify such sequences so that IncUSP-Miner$^+$ is able to safely prune them for better mining efficiency.

The details of algorithm USP-Miner are described in Section 4.2, while the details of algorithm IncUSP-Miner$^+$ are given in Section 4.3.

4.2 Initial Phase

4.2.1 Efficient Utility Calculation. Considering two sequences s and t where $t \sqsubseteq s$, it is possible that s contains multiple instances of t at different positions. A naive way to obtain the utility of t in s is to find the positions of all instances of t in s, compute the utility of each instance, and obtain the maximum of the utilities of all instances. It is obvious that such a naive way is inefficient. This motivates us to design an efficient way to calculate the maximum utility t in s.

Lemma 4. *Given two sequences s and t where $t \sqsubseteq s$, suppose that s has multiple instances of t and let the set of the distinct extension positions of sequence t in s be $P = \{p_1, p_2, \ldots, p_q\}$. The utility of t in s (i.e., $u(t,s)$) equals* $\max\{u(t, p_i, s) | \forall p_i \in P\}$.

Proof. Let the set of all instances of t in s be O and let $O_{p_i} \subseteq O$ be the set of instances of t in s with extension position p_i. In addition, according to Definition 12, we have

$$u(t, p_i, s) = max_{\forall inst \in O_{p_i}}\{utility\ of\ inst\}.$$

According to Definition 3, $u(t,s)$ is defined as the maximum of utilities of all instances of t in s. It is clear that $O = \bigcup_{i=1}^{q} O_{p_i}$, and $O_{p_i} \bigcap O_{p_j} = \emptyset$, for all $i \neq j$. Thus, we can prove this lemma by the following equation:

$$\begin{aligned} u(t,s) &= max_{\forall inst \in O}\{utility\ of\ inst\} \\ &= \max_{\forall p_i \in P}\left\{max_{\forall inst \in O_{p_i}}\{utility\ of\ inst\}\right\} \\ &= \max_{\forall p_i \in P}\{u(t, p_i, s)\}. \end{aligned} \tag{15}$$
□

With Lemma 4, we can efficiently derive $u(t,s)$ from the maximum of the maximum utilities $u(t,p,s)$ of t at each extension position p. In the following, we will show how to efficiently calculate the maximum utility of each t's extension sequence, say t', at each extension position p (i.e., $u(t', p, s)$) based on the maximum utility of t at all extension positions.

Calculating Utility of 1*-sequences.* It is easy to calculate the utility of a 1−sequence t at each extension position with one database scan. In the example shown in Table 1(a), the utilities of $\langle a \rangle$ at all extension positions in s_2 are $u(\langle a \rangle, 1, s_2) = 3$ and $u(\langle a \rangle, 2, s_2) = 12$.

Calculating Utility of l-sequences ($l \geq 2$). To perform the extension process of t in s, we first record all extension positions of t in s, since they are the qualified positions for t to form i- or s-extension sequences. We also record the maximum utility of the instances of t in s at each extension position p to facilitate efficient calculation of the utility of each extension sequence of t, say t', in s.

Theorem 3. *Given a sequence t, one t's i-extension sequence t', and a sequence s in D where $t' \sqsubseteq s$, the maximum utility of t' in s at extension position p (i.e., $u(t', p, s)$) can be derived by $u(t', p, s) = u(t, p, s) + u(i', p, s)$, where item i' is the extension item of t'.*

Proof. Since t' is an i-extension sequence of t, the instances of t' at extension position p can be formed by appending the extension item of t' (i.e., i') to the last item within the last itemset of all instances of t at extension position p. So, the maximum utility of t' at extension position p (i.e., $u(t', p, s)$) can be calculated by the following equation.

$$u(t', p, s) = \max\{u(t, \langle j_1, j_2, \ldots, p \rangle, s) + u(i', p, s) | \\ \forall 1 \leq j_1 < j_2 < \cdots < p \leq m : t \sqsubseteq \langle Y_{j_1} Y_{j_2} \ldots Y_p \rangle\}$$

Since the utility of the extension item i' in the pth itemset of s ($u(i', p, s)$) is fixed, the above equation can be rewritten as below:

$$u(t', p, s) = \max\{u(t, \langle j_1, j_2, \ldots, p \rangle, s) | \\ \forall 1 \leq j_1 < j_2 < \cdots < p \leq m : t \sqsubseteq \langle Y_{j_1} Y_{j_2} \ldots Y_p \rangle\} + u(i', p, s).$$

Finally, according to Definition 12, we can rewrite the above equation as $u(t', p, s) = u(t, p, s) + u(i', p, s)$, thereby proving this lemma. □

For example, the utility of sequence $\langle [a, c] \rangle$ in s_2 at extension position 1 (i.e., $u(\langle [a, c] \rangle, 1, s_2)$), can be obtained by adding the utility of extension item c (i.e., $u(c, 1, s_2)$) and $u(\langle a \rangle, 1, s_2)$, which is $3 + 1 = 4$.

Theorem 4. *Given a sequence t, one t's s-extension sequence t', and a sequence s in D where $t' \sqsubseteq s$, let item i' be the extension item of t' and p be an extension position of t'. Also let $P = \{p_1, p_2, \ldots p_q\}$ be the set of extension positions of t in s. The maximum utility of t' in s at extension position p can be derived from $u(t', p, s) = \max\{u(t, p_i, s) | \forall\, p_i \in P$ and $p_i < p\} + u(i', p, s)$.*

Proof. Since t' is an s-extension sequence of t, the instances of t' at extension position p can be formed by appending a new itemset containing only the extension item i' at extension position p to all instances of t whose extension positions are smaller than p. So, the maximum utility of t' at extension position p (i.e., $u(t', p, s)$) can be calculated by the following equation:

$$u(t', p, s) = \max\{u(i', p, s) + u(t, \langle j_1, j_2, \ldots, p_i \rangle, s) | \\ \forall 1 \leq j_1 < j_2 < \cdots < p_i < p \leq m : t \sqsubseteq \langle Y_{j_1} Y_{j_2} \ldots Y_{p_i} \rangle\}.$$

Since the item utility of extension item i' in the pth itemset of s (i.e., $u(i', p, s)$) is fixed, the above equation can be rewritten as below:

$$u(t', p, s) = \max\{u(t, \langle j_1, j_2, \ldots, p_i \rangle, s) | \\ \forall 1 \leq j_1 < j_2 < \cdots < p_i < p \leq m : t \sqsubseteq \langle Y_{j_1} Y_{j_2} \ldots Y_{p_i} \rangle\} + u(i', p, s).$$

According to Definition 12, we can rewrite the above equation as $u(t', p, s) = \max\{u(t, p_i, s) | \forall p_i \in P \text{ and } p_i < p\} + u(i', p, s)$, thereby proving this lemma. □

In the example shown in Table 1(a), the utility of sequence $\langle a, c \rangle$ in s_2 at extension position 3 (i.e., $u(\langle a, c \rangle, 3, s_2)$) can be calculated by adding $\max\{u(\langle a \rangle, 1, s_2), u(\langle a \rangle, 2, s_2)\}$ and the utility of c in s_2 at extension position 3 (i.e., $u(c, 3, s_2)$), which is $\max\{3, 12\} + 2 = 14$.

According to Theorems 3 and 4, the maximum utility of an i- or s-extension sequence t' of sequence t in s at extension position p (i.e., $u(t', p, s)$) can be efficiently derived from the maximum utility of t at each extension position p_i (i.e., $u(t, p_i, s)$). Moreover, according to Lemma 4, it is efficient to derive the utility of t' in s (i.e., $u(t', s)$) from the maximum utilities of t' in s at each extension position p (i.e., $u(t', p, s)$) without finding all instances of t' in s.

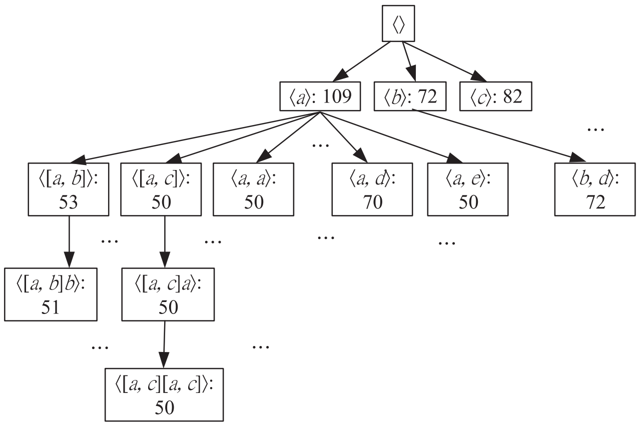
4.2.2 Buffering Sequences in the Candidate Pattern Tree. A candidate pattern tree is an extension of the lexicographic tree buffering a set of sequences in D for incremental HUSP mining. According to Theorem 2, TSU is a tighter upper bound of the utilities of t and all t's descendant sequences. Hence, we use TSU to guide the construction of the candidate pattern tree T in the initial phase. A sequence t is called a *high TSU sequence* in D when $TSU_D(t) \geq min\_util$. Otherwise, t is called a *low TSU sequence* in D. Since a high utility sequential pattern is also a high TSU sequence, all high TSU sequences are buffered in the candidate pattern tree T for better mining efficiency in the incremental phase.

To speed up utility calculation in the incremental phase, for each sequence t in the tree T, in addition to the sequence ID, $u(t, p, s)$ at each extension position p in each sequence s where $t \sqsubseteq s$, is recorded for avoiding the redundant utility calculations. Besides, we keep the information indicating whether t has no low TSU extension sequence in D. If not, then the maximal TSU value of all t's low TSU extension sequences is kept. To facilitate efficient utility calculation and support multiple database updates, each node t in T contains the following auxiliary information:

- *Seq*: the sequence of t.
- *TSU(t)*: the TSU value of t in D, i.e., $TSU_D(t)$.
- *PEU(t)*: the PEU value of t in D, i.e., $PEU_D(t)$.
- *u(t)*: the utility of t in D.
- $LM_{TSU}(t)$: the maximal TSU value of all t's low TSU extension sequences in D. This field will be 0 when none of t's extension sequences is of low TSU in D.
- *IsHUSP*: whether t is an HUSP in D. This value is set to TRUE when t is an HUSP in D. Otherwise, this value is set to FALSE.
- *Sid*: a list of sequence IDs of the sequences containing t in D.
- *PEU(t,s)*: a list of PEU values of t in each sequence s where $t \sqsubseteq s$.
- *u(t,s)*: a list of utility values of t in each sequence s where $t \sqsubseteq s$.
- $\{\langle p, u(t,p,s) \rangle\}$: a set of p and $u(t, p, s)$ pairs for each sequence s containing t (i.e., $t \sqsubseteq s$), where p is an extension position of t in s, and these pairs are arranged according to their extension positions in ascending order.

Example 4.1. Figure 3(a) shows the candidate pattern tree T of the original database D shown in Table 1(a) when $min\_util = 50$. For brevity, we only show the sequence and the TSU value of each node in Figure 3(a), and the full node structure of sequence $\langle a \rangle$ is given in Figure 3(b). Since sequence $\langle a \rangle$ has only one low TSU extension sequence $\langle [a, e] \rangle$ in D, the value of $LM_{TSU}(\langle a \rangle)$ is set to $TSU_D(\langle [a, e] \rangle) = 21$.

4.2.3 USP-Miner Algorithm. Algorithm USP-Miner is designed to build all tree nodes of the candidate pattern tree and extract HUSPs by traversing the candidate pattern tree from the root in



(a) The Candidate Pattern Tree T

Seq	PEU(t)	TSU(t)	LM$_{TSU}$(t)	u(t)	IsHUSP
⟨a⟩	103	109	21	48	false
sid	**PEU(t,s)**	**u(t, s)**	**{⟨p, u(t,p,s)⟩}**		
1	53	15	⟨1, 15⟩		
2	29	12	⟨1, 3⟩	⟨2, 12⟩	
3	0	6	⟨4, 6⟩		
5	21	15	⟨1, 3⟩	⟨2, 15⟩	

(b) The Node Structure of Sequence $\langle a \rangle$ in T

Fig. 3. The candidate pattern tree T with the full node structure.

a DFS manner. First, USP-Miner scans the database to collect all high TSU 1-sequences. For each high TSU 1-sequence t, USP-Miner builds one tree node of t as a child node of the root, inserts t into $HUSP$ when it is, and recursively expands t (i.e., builds all child nodes of t) by recursively calling USP-Miner with t as the input. When expanding a node t, USP-Miner first checks whether t's PEU value is less than min_util. If so, then according to Theorem 1, all t's descendent sequences will not be of high utility, and thus, USP-Miner can safely skip t without expansion. Otherwise, USP-Miner scans the t-projected database in D to collect all t's high TSU extension sequences. For each t's high TSU extension sequence t', USP-Miner builds the corresponding node structure as a child node of t, inserts t' into $HUSP$ if it is, and expands t' by recursively calling USP-Miner with t' as the input. The other nodes are processed in the same way. When USP-Miner terminates, the complete set of HUSPs is stored in $HUSP$ and the candidate pattern tree is built. Finally, the algorithmic form of algorithm USP-Miner is shown in Algorithm 1.

4.3 Incremental Phase

Definition 18. The updated database D' can be partitioned into two disjoint parts.

- LDB: the set of updated sequences (i.e., inserted and appended sequences) in D'.
- NDB: the set of sequences not in LDB. In other words, NDB is the set of the sequences in D and unchanged in D'.

In the example shown in Table 1(a), $NDB = \{s_1, s_2, s_3\}$ and $LDB = \{s_4, s_5, s_6\}$.

ALGORITHM 1: USP-Miner(t)

```
Input: A sequence database $D$, the minimum utility threshold $\xi$, an empty set $HUSP$, and a sequence $t$
       in $T$
Output: $HUSP$: the set of HUSPs
if $PEU(t) < min\_util$ then
    return;
scan $t$-projected DB once to put i- and s-extension items into $ilist$ and $slist$, respectively;
delete low TSU items from $ilist$ and $slist$;
foreach item $i \in ilist$ do
    $t' \leftarrow I - Extension(t, i)$;
    BuildNode($t'$) and add $t'$ to $T$ as $t$'s child;
    if $u(t') \geq min\_util$ then
        insert $t'$ into $HUSP$;
    USP-Miner($t'$);
foreach item $i \in slist$ do
    $t' \leftarrow S - Extension(t, i)$;
    BuildNode($t'$) and add $t'$ to $T$ as $t$'s child;
    if $u(t') \geq min\_util$ then
        insert $t'$ into $HUSP$;
    USP-Miner($t'$);
```

After D is updated to D', we have to find new HUSPs and adjust the candidate pattern tree T to support multiple database updates. Since an HUSP is also a high TSU sequence, keeping the tree T storing all high TSU sequences in D' and keeping the auxiliary information stored in the tree T up-to-date are crucial to incremental HUSP mining. For each sequence t, we have to consider the following cases after database update.

- *Case 1*: t is a high TSU sequence in D and D'.
- *Case 2*: t is a low TSU sequence in D and becomes a high TSU sequence in D'.
- *Case 3*: t is a low TSU sequence in D and D'.

The rationales to handle the above cases are as follows:

- *Handling Case 1*: Since t is already in the tree T, if t is contained by some sequence(s) in LDB, keeping t's auxiliary information stored in T up-to-date is necessary. The proposed method to update the auxiliary information stored in tree nodes is described in Section 4.3.1. On the other hand, if t is not contained by any sequence in LDB, we can skip all tree nodes in the subtree rooted at t without updating the auxiliary information in these tree nodes. The proposed method to skip these tree nodes is given in Section 4.3.2.
- *Handling Case 2 and Case 3*: Let s be the longest prefix of t where the tree node of s is already in the tree T. If such s exists, then to add the tree node of t to the tree T, algorithm USP-Miner can be recursively applied on s and each node of the subtree rooted by s. Thus, the tree node of t will be eventually added to the tree T. Otherwise, USP-Miner can be applied on root and thus the node of t will be added to T (Case 2). An intuitive way to ensure that all new case 2 high TSU extension sequences of t are added into the tree T is always to scan the t-projected DB of D'. We observe that scanning the t-projected DB of LDB is just enough for updating the auxiliary information storing in the tree T when all t's low TSU extension sequences in D will not become high TSU in D' (Case 3). Based on this observation, we design some

strategies to reduce the scopes of some database scans in Section 4.3.3 for better mining efficiency.

To support multiple database updates, we also design some auxiliary information update strategies in Section 4.3.4. Finally, we devise algorithm IncUSP-Miner$^+$ to incrementally mine HUSPs based on the above discussions, and the details of the IncUSP-Miner$^+$ algorithm is described in Section 4.3.5.

4.3.1 Tree Node Update. When the utility of a sequence t buffered in the tree T has to be updated, scanning only the updated sequences containing t is enough to obtain the up-to-date utility of t (i.e., $u_{D'}(t)$), since only these sequences will affect t's utility in D'. For an appended sequence s' in LDB, let s be the corresponding sequence in D forming s' by sequence appending. Because $|D'|$ is usually much larger than $|LDB|$, the following equation can be used to calculate $u_{D'}(t)$ efficiently:

$$u_{D'}(t) = u_D(t) + \sum_{\forall s' \in LDB \wedge t \sqsubseteq s'} (u(t, s') - u(t, s)), \tag{16}$$

where $u(t, s)$ is 0 if s' is an inserted sequence, or s' is an appended sequence of s in D but $t \not\sqsubseteq s$. Other auxiliary information of t such as TSU and PEU can be efficiently calculated in a similar manner. Finally, we use the following example to describe the proposed method.

Example 4.2. In the example in Table 1(a), the utility of sequence $\langle a, c\rangle$ in D is $u_D(\langle a, c\rangle) = 18$. Assume that $\langle a, c\rangle$ is buffered in the tree T. From Table 1(a), we know that $u(\langle a, c\rangle, s_5)$ in D is 4 and $u(\langle a, c\rangle, s'_5)$ in D' is 16. Since s'_5 is the only appended sequence containing $\langle a, c\rangle$, $u_{D'}(\langle a, c\rangle)$ is equal to $u_D(\langle a, c\rangle)$ plus $u(\langle a, c\rangle, s'_5)$ in D' and minus $u_D(\langle a, c\rangle, s_5)$ in D, where $u_D(\langle a, c\rangle)$ and $u(\langle a, c\rangle, s_5)$ in D can be obtained from the tree node of $\langle a, c\rangle$ in the tree T. Thus, we can get $u_{D'}(\langle a, c\rangle) = 18 + 16 - 4 = 30$.

4.3.2 Node Skipping Strategy. After database updates, tree nodes in the tree T and without any instance in all sequences in LDB can be safely skipped without updating. In other words, we can skip a tree node t and all its descendant sequences when t does not have any instance in LDB. In addition, it is obvious that all t's descendant sequences will have no instance in LDB when the t-projected DB of LDB is empty or contains only $\langle\rangle$. Therefore, we propose the following node skipping strategy for better mining efficiency.

STRATEGY 1. *For a sequence t in the tree T, all t's descendant sequences can be safely skipped without updating if the t-projected DB of LDB is empty or contains only $\langle\rangle$.*

We use the following example to show the effect of Strategy 1.

Example 4.3. Consider the example in Table 1(a) and assume that $min\_util = 15$. Obviously, sequence $\langle a, c, c, e\rangle$ is buffered in the tree T, since $TSU_D(\langle a, c, c, e\rangle) = 16 > min\_util$. In LDB, there is only one appended sequence (i.e., s'_5) having an instance of $\langle a, c, c, e\rangle$. According to Strategy 1, all descendant sequences of t will be skipped without updating, since the $\langle a, c, c, e\rangle$-projected DB of LDB is $\{\langle\rangle\}$.

4.3.3 Database Scan Reduction Strategies. For each node, say t, which cannot be skipped, a naive way to check whether t's low TSU extension sequences in D will become high TSU in D' is to scan D'. Since LDB is usually much smaller than D', we propose the following two strategies to reduce the scopes of some database scans from D' to LDB to improve mining efficiency.

STRATEGY 2. *For a sequence t in the tree T, if $LM_{TSU}(t) = 0$ in D (i.e., before database update), scanning LDB is enough to (1) update the auxiliary information of t's high TSU extension sequences in D and (2) find t's new high TSU extension sequences in D'.*

Proof. $LM_{TSU}(t) = 0$ in D means that each t's extension sequence, say t', is either (1) a high TSU sequence in D or (2) of no instance in D. If t' is in the first condition, then the method proposed in Section 4.3.1 can be used to update the auxiliary information of t' in D' by scanning LDB, since the auxiliary information of t' in D is already kept in the corresponding tree node in the tree T. Otherwise, if t' is in the second condition, scanning LDB is enough to find all t''s instances in D', thereby able to determine whether t' is a new high TSU sequence in D'. □

Strategy 3. *Consider a sequence t in the tree T where $LM_{TSU}(t) > 0$ in D (i.e., before database update). If $LM_{TSU}(t)$ in D plus $PEU_{LDB}(t)$ is less than min_util, then none of t's low TSU sequences will become a high TSU sequence in D′. Thus, scanning LDB is enough to update the auxiliary information of t's high TSU extension sequences in D.*

Proof. Since $LM_{TSU}(t)$ is defined as the maximal TSU value among all t's low TSU extension sequences, $LM_{TSU}(t) > 0$ in D means that t has at least one low TSU extension sequence in D before database update. For each t's low TSU extension sequence in D, say t', we have $TSU_D(t') < min\_util$ and

$$TSU_D(t') \leq LM_{TSU}(t). \tag{17}$$

According to Definition 14, the TSU of t' is not greater than the PEU of t. Thus, we have

$$TSU_{LDB}(t') \leq PEU_{LDB}(t). \tag{18}$$

Combining Equations (17) and (18), we have

$$TSU_D(t') + TSU_{LDB}(t') \leq LM_{TSU}(t) + PEU_{LDB}(t). \tag{19}$$

Since $D' = NDB \cup LDB$, $NDB \cap LDB = \emptyset$ and $NDB \subseteq D$, we have

$$\begin{aligned} TSU_{D'}(t') &= TSU_{NDB}(t') + TSU_{LDB}(t') \\ &\leq TSU_D(t') + TSU_{LDB}(t') \\ &\leq LM_{TSU}(t) + PEU_{LDB}(t). \end{aligned} \tag{20}$$

According to the above equation, if $LM_{TSU}(t) + PEU_{LDB}(t) < min\_util$, we have $TSU_{D'}(t') < min\_util$, meaning that t' is still a low TSU sequence in D'.

In summary, for a sequence t where $LM_{TSU}(t) > 0$ in D and $LM_{TSU}(t) + PEU_{LDB}(t) < min\_util$, none of t's low TSU extension sequences in D will become a high TSU sequence in D' after database update. In addition, for each t's high TSU extension sequence in D, say t'', since the auxiliary information of t'' is already kept in the corresponding tree node in the tree T, scanning LDB is enough to update the auxiliary information of t'' in D'. □

Finally, for each node t in the tree T, if the scopes of database scans can be reduced by Strategies 2 or 3, we only have to scan the t-projected DB of LDB. Otherwise, we have to scan the t-projected DB of D' to find t's new high TSU extension sequences.

4.3.4 Supporting Multiple Database Updates.

According to the discussions in Section 4.3.3, for each node t in the tree T, keeping $LM_{TSU}(t)$ in D' equal to, or at least greater than, the maximal TSU of t's low TSU sequences in D' is important to the correctness of algorithm IncUSP-Miner$^+$ when multiple database updates occur. A naive method is to scan the t-projected DB of D' to update $LM_{TSU}(t)$ in D' after each database update. Obviously, this mehtod is inefficient. In view of this, we design the following strategies to facilitate efficient update of $LM_{TSU}(t)$ by reducing the scopes of database scans.

Strategy 4. *For a sequence t in the pattern tree T where $LM_{TSU}(t) = 0$ in D, one can scan the t-projected DB of LDB to obtain the maximal TSU, say emltsu, of t's low TSU extension sequences in the t-projected DB of LDB and update $LM_{TSU}(t)$ in D′ to emltsu.*

Proof. $LM_{TSU}(t) = 0$ in D means that each t's extension sequence, say t', is either a high TSU sequence in D or of no instance in D. After database updates, if t has at least one low TSU extension sequence in D', the instances of t's low TSU extension sequences only exist in the t-projected DB of LDB. Thus, scanning the t-projected DB of LDB is able to find the maximal TSU, say *emltsu*, of all t's low TSU extension sequences in D'. Since there is no instance of each t's low TSU extension sequence in D, we can update $LM_{TSU}(t)$ in D' to *emltsu*. □

If t has at least one low TSU extension sequences in D (i.e., $LM_{TSU}(t) > 0$ in D) and $LM_{TSU}(t)$ in D plus $PEU_{LDB}(t)$ is less than *min_util*, then according to Strategy 3, none of t's low TSU extension sequences D will become high TSU in D'. To keep the value of $LM_{TSU}(t)$ in D' up-to-date, in addition to the t-projected DB of LDB, NDB should be scanned as well. However, it is inefficient, since the size of NDB is usually much larger than the size of the t-projected DB of LDB. In fact, according to the discussions in Section 4.3.3, setting the value of $LM_{TSU}(t)$ in D' greater than the maximal TSU of all t's low TSU extension sequences in D' (i.e., greater than the precise value of $LM_{TSU}(t)$ in D') only results in some extra computation and does not affect the correctness of algorithm IncUSP-Miner$^+$. In view of this, we propose the following strategy to reduce the scopes of some database scans to improve mining efficiency.

Strategy 5. *Consider a sequence t in the candidate pattern tree T whose $LM_{TSU}(t) > 0$ in D. If $LM_{TSU}(t)$ in D plus $PEU_{LDB}(t)$ is less than min_util in D', then one can scan the t-projected DB of LDB to obtain the maximal TSU of t's low TSU extension sequences in the t-projected DB of LDB, which is denoted as emltsu, and update $LM_{TSU}(t)$ in D' to the sum of emltsu and $LM_{TSU}(t)$ in D.*

Proof. $LM_{TSU}(t) > 0$ in D means that t has at least one low TSU extension sequence in D. In addition, according to Strategy 3, $LM_{TSU}(t)$ in D plus $PEU_{LDB}(t)$ less than *min_util* in D' means that none of t's low TSU extension sequences in D will become high TSU sequence in D'. Let t' be the extension sequence of the maximal TSU value among all t's low TSU extension sequences in D'. Since $NDB \subseteq D$ and $LM_{TSU}(t)$ in D is the maximal TSU value of all t's low TSU extension sequences in D, we have

$$TSU_{NDB}(t') \leq LM_{TSU}(t) \ in \ D. \tag{21}$$

Let *emltsu* be the maximal TSU of all t's low TSU extension sequences in LDB, we have

$$TSU_{LDB}(t') \leq emltsu. \tag{22}$$

Since $D' = NDB \cup LDB$, we can derive from the above two equations that

$$\begin{aligned} TSU_{D'}(t') &= TSU_{LDB}(t') + TSU_{NDB}(t') \\ &\leq emltsu + LM_{TSU}(t) \ in \ D. \end{aligned} \tag{23}$$

According to the above equation, *emltsu* + $LM_{TSU}(t)$ in D is greater than or equal to the maximal TSU value of all t's low TSU extension sequences in D' (i.e., greater than or equal to the precise value of $LM_{TSU}(t)$ in D'), and hence, can be set as the new value of $LM_{TSU}(t)$ in D' without affecting the correctness of algorithm IncUSP-Miner$^+$. □

We now consider the sequence t where (1) t has at least one low TSU extension sequence in D so that $LM_{TSU}(t) > 0$ and (2) $LM_{TSU}(t)$ in D plus $PEU_{LDB}(t)$ is greater than or equal to *min_util*. The following strategy is proposed to efficiently handle the update of $LM_{TSU}(t) > 0$ in D.

Strategy 6. *Consider a sequence t in the candidate pattern tree T where $LM_{TSU}(t) > 0$ in D. If $LM_{TSU}(t)$ in D plus $PEU_{LDB}(t)$ is greater than or equal to min_util, then one has to scan the t-projected DB of D' to obtain the maximal TSU value, denoted as emltsu, of t's low TSU extension sequences in the t-projected DB of D', and set $LM_{TSU}(t)$ in D' to emltsu.*

Proof. It is obvious that all instances of all t's low TSU extension sequences in D' must be in the t-projected DB of D'. Thus, we can scan the t-projected DB of D' to obtain the maximal TSU value of all t's low TSU extension sequences in D' (i.e., *emltsu*) and update $LM_{TSU}(t)$ in D' to *emltsu*. □

ALGORITHM 2: IncUSP-Miner$^+$(t)

```
Input: A sequence database $D'$, $NDB$, $LDB$, $min\_util$, a candidate pattern tree $T$, $HUSP$ (the set of
       HUSPs in $D$), and a sequence $t$ in $T$.
Output: $HUSP$: the set of HUSPs in $D'$
1  $LDB|_t \leftarrow t$-projected DB of $LDB$;
2  if all sequences in $LDB|_t$ are empty sequences then
3      return;
4  if $PEU_{D'}(t) < min\_util$ then
5      return;
6  else if $PEU_D(t) < min\_util$ then
7      USP-Miner(t);
8      return;
9  $emltsu \leftarrow 0$;
10 scan $LDB|_t$ to calculate the TSU of all 1-sequences;
11 if $LM_{TSU}(t) = 0 \vee PEU_{LDB}(t) + LM_{TSU}(t) < min\_util$ then
12     foreach $t$'s extension sequence $t' \in LDB$ do
13         if $t' \in T$ then // case (1)
14             HandleCase-1($t'$);
15         else if $TSU_{LDB}(t') \geq min\_util$ then // case (2)
16             HandleCase-2($t'$);
17         else
18             if $TSU_{LDB}(t') > emltsu$ then
19                 $emltsu \leftarrow TSU_{LDB}(t')$;
20 else
21     $LM_{TSU}(t) \leftarrow 0$;
22     $NDB|_t \leftarrow t$-projected DB of $NDB$;
23     scan $NDB|_t$ to calculate the TSU of 1-sequences;
24     foreach $t$'s extension sequence $t' \in D'$ do
25         if $TSU_{D'}(t') \geq min\_util$ then
26             if $t' \in LDB$ then
27                 if $t' \in T$ then // case (1)
28                     HandleCase-1($t'$);
29                 else // case (2)
30                     HandleCase-2($t'$);
31         else
32             if $TSU(t') > emltsu$ then
33                 $emltsu \leftarrow TSU(t')$;
34 $LM_{TSU}(t) \leftarrow LM_{TSU}(t) + emltsu$;
```

4.3.5 IncUSP-Miner$^+$ Algorithm. After an SDB is updated, IncUSP-Miner$^+$ is executed with input $\langle\rangle$ (i.e., the the root of T) to mine the HUSPs of D' incrementally. IncUSP-Miner$^+$ is a recursive algorithm and the behavior of IncUSP-Miner$^+$ with input t is as follows. If all sequences in the t-projected DB of LDB (i.e., $LDB|_t$) are empty sequences, then IncUSP-Miner$^+$ skips t without further processing according to Strategy 1. Otherwise, IncUSP-Miner$^+$ scans $LDB|_t$ to calculate

Procedure HandleCase-1(t')

```
1 UpdateNode($t'$);
2 if $u(t') \geq min\_util \wedge \neg t'.IsHUSP$ then
3     $t'$.IsHUSP$\leftarrow True$ and insert $t'$ into $HUSP$;
4 IncUSP-Miner$^+$($t'$);
```

Procedure HandleCase-2(t')

```
1 BuildNode($t'$) and add $t'$ to $T$;
2 if $u(t') \geq min\_util$ then
3     $t'$.IsHUSP$\leftarrow True$ and insert $t'$ into $HUSP$;
4 USP-Miner($t'$);
```

the TSU of each extension sequence of t in LDB. If t has no low TSU extension sequence in D (i.e., $LM_{TSU}(\text{t}) = 0$), or all t's low TSU extension sequences in D cannot become high TSU in D' (i.e., $PEU_{LDB}(t) + LM_{TSU}(t) < min\_util$), then according to Strategies 2 or 3, IncUSP-Miner$^+$ skips all the t's extension sequences in T but not in LDB and only processes those extension sequences in LDB, say t', by the following procedure. If t' is also in T (Case 1), then IncUSP-Miner$^+$ updates the auxiliary information of t' and calls itself recursively with input t'. If t' is a new high TSU sequence (Case 2), then IncUSP-Miner$^+$ builds the node of t' and calls algorithm USP-Miner with input t' to build the subtree rooted at t'. IncUSP-Miner$^+$ then updates $LM_{TSU}(t)$ by $LM_{TSU}(t) + emltsu$ according to Strategies 4 or 5.

Otherwise, if some t's low TSU extension sequences in D are likely to become high TSU in D', IncUSP-Miner$^+$ still needs to scan the t-projected DB of NDB (i.e., $NDB|_t$) to obtain the TSU of each t's extension sequence in D'. IncUSP-Miner$^+$ skips all the t's extension sequence in T but not in LDB. For each t's high TSU extension sequence, say t', having instances in LDB, if t' is also in T (Case 1), then IncUSP-Miner$^+$ processes t' by the Case 1 sequence processing method mentioned above. Otherwise (Case 2), IncUSP-Miner$^+$ processes t' by the Case 2 sequence processing method mentioned above. According to Strategy 6, $LM_{TSU}(t)$ in D' is set to $emltsu$ to support multiple database updates. Finally, the algorithmic form of IncUSP-Miner$^+$ is shown in Algorithm 2 for interested readers.

5 PERFORMANCE EVALUATION

In this section, the performance of algorithm IncUSP-Miner$^+$ is evaluated. The algorithm proposed in our preliminary study [16], algorithm IncUSP-Miner, and two state-of-the-art algorithms for mining HUSPs from scratch, algorithms USpan [19] and HUS-Span [17], are also implemented for comparison purposes. All algorithms are implemented in C++ using Mingw-w64 compiler with O3 optimization. Experiments were performed on a computer with a 3.4GHz Intel Core i7 CPU, 12GB memory, and running on Windows 7 x64 operating system. One synthetic and three real datasets are used in experiments. Real datasets Foodmart[1] and Tafeng[2] are grocery shopping datasets, where each sequence represents a list of transactions conducted by a unique customer. The item utility within a transaction represents the sales amount of the item (product). Real dataset Music[3] is acquired from the user ratings of 100 most popular music products in Amazon, where each

[1]http://www.informit.com/store/microsoft-sql-server-2008-analysis-services-unleashed-9780672330018.
[2]http://aiia.iis.sinica.edu.tw/index.php?option=com_frontpage&Itemid=1.
[3]http://snap.stanford.edu/data/web-Amazon.html.

Table 2. Characteristics of Synthetic and Real Datasets

(a) Statistics and Parameters of Synthetic and Real Datasets

Dataset	**D′**	**L**	**N**	$\frac{\lvert\Delta db\rvert}{\lvert D\rvert}$
Foodmart	8842	30.5	1559	0.05
Tafeng	32266	25.34	23812	0.05
Music	32778	4.55	100	0.05
D50C10T5N0.5S4I2.5	50K	43.13	500	0.05

(b) Statistics and Parameter Description

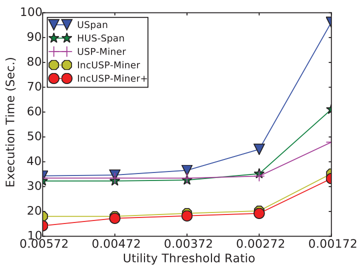
Statistics (Parameter)	**Description**
D′	Number of sequences in the updated database
L	Average length per sequence
N	Number of distinct items
$\frac{\lvert\Delta db\rvert}{\lvert D\rvert}$	Update Ratio

sequence represents a list of products rated by a unique user. The item utility represents the rating of the item (product) made by the user at a particular time. The synthetic dataset D50C10T5N0.5 is generated by the IBM data generator [14]. Due to the lack of utility information, the internal utility of each item is randomly generated from 1 to 10, while the external utility of each item is randomly generated using a log-normal distribution similar to existing studies [17, 19]. The parameter settings of the synthetic dataset and the statistics of the real datasets are summarized in Table 2(a), while the meaning of the parameters and statistics of datasets are expressed in Table 2(b).

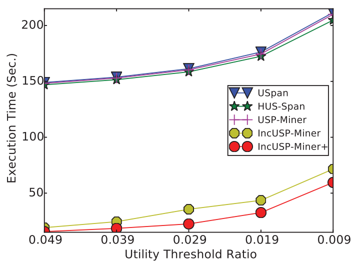
5.1 Effect of Utility Threshold

This subsection shows the experimental results of all the above algorithms on each dataset under varied utility thresholds. The minimum utility threshold *min_util* is set as a threshold ratio (ξ) times the utility of the original database (i.e., $min\_util = \xi \times \sum_{\forall s \in D} u_D(s)$).

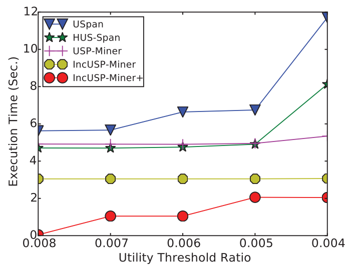
Figure 4 shows the execution time on each dataset under varied utility threshold ratio ξ. Generally, for each dataset, IncUSP-Miner$^+$ and IncUSP-Miner outperform all algorithms mining HUSPs from scratch. The reason is that the candidate pattern tree is helpful for efficiently calculating utilities and the proposed node skipping strategy is able to skip some nodes not necessary to be visited. One can see that IncUSP-Miner$^+$ is more efficient than IncUSP-Miner, and the difference of the performance between them is significant especially on dataset D50C10T5N0.5S4I2.5. The reason is that the value of $\frac{N}{D}$ on dataset D50C10T5N0.5S4I2.5 is much smaller than those on other datasets. When $\frac{N}{D}$ is small, the average fan-out of the candidate pattern tree is small, making the performance of the proposed database scan reduction strategies (used in IncUSP-Miner$^+$) more significant than other strategies. Therefore, IncUSP-Miner$^+$ outperforms IncUSP-Miner especially when $\frac{N}{D}$ is small. It is interesting that although adopting an upper bound of sequence utility (i.e., TSU) tighter than HUS-Span, USP-Miner is not as efficient as HUS-Span on most cases. It is because in addition to mining HUSPs, USP-Miner has an extra job for building the candidate pattern tree, resulting in more computation overhead. Fortunately, the performances of USP-Miner and HUS-Span are very close. In some cases, USP-Miner outperforms HUS-Span and USpan, showing the advantage of TSU.



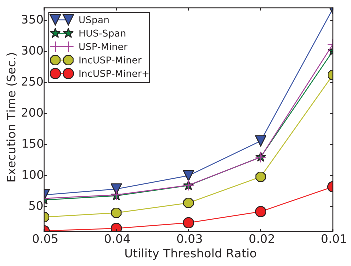
(a) Foodmart



(b) Tafeng



(c) Music



(d) D50C10T5N0.5S4I2.5

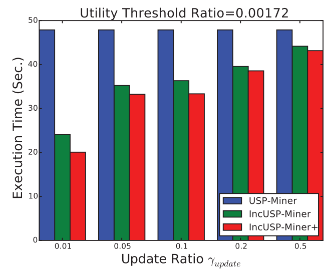
Fig. 4. Effect of utility threshold.

5.2 Effects of Update Ratio

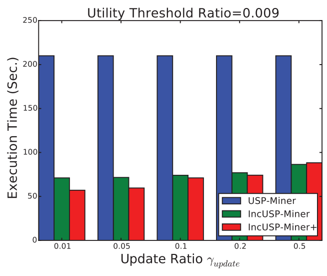
The *update ratio* is defined as the number of the updated sequences (appended sequences and inserted sequences) in Δdb to the number of sequences in the original database D, i.e., $\frac{|\Delta db|}{|D|}$. To evaluate the effect of update ratio, we set the values of the update ratio to 0.01, 0.05, 0.1, 0.2, and 0.5 and the experimental results are shown in Figure 5. It can be observed from Figure 5 that when the updated ratio gets smaller, IncUSP-Miner$^+$ and IncUSP-Miner get more efficient than USP-Miner. It is because when the updated ratio is large, the database utility of *LDB* is very likely to be large. Such result makes the node skipping strategy (used in IncUSP-Miner$^+$ and IncUSP-Miner) and the database scan reduction strategies (used in IncUSP-Miner$^+$) not perform well, and a large number of new high TSU sequences arise in D'. Therefore, IncUSP-Miner$^+$ and IncUSP-Miner have to adjust many tree nodes in the candidate pattern tree by scanning *LDB* multiple times. In addition, IncUSP-Miner$^+$ outperforms IncUSP-Miner in most cases, showing the advantage of the proposed database scan reduction strategies. The experimental results shown in Figure 5 suggest to perform IncUSP-Miner$^+$ to mine HUSPs incrementally when the updated ratio is lower than or equal to 0.2.

5.3 Scalability Test: Effects of Database Size

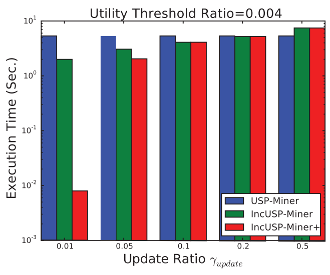
We also evaluate the scalability of all algorithms under different database sizes on a dense dataset (DxC10T5N0.5S4I2.5) and a sparse dataset (DxC10T5N10S4I2.5), where the size of the database (i.e., number of sequences) ranges from 100,000 to 300,000 in increments of 50,000. The execution time and memory usage are shown in Figure 6. As shown in Figures 6(a) and 6(c), with the aid of the candidate pattern tree and node skipping strategy, IncUSP-Miner$^+$ and IncUSP-Miner significantly outperform the algorithms mining HUSPs from scratch. By adopting the proposed database scan reduction strategies, IncUSP-Miner$^+$ is more efficient than IncUSP-Miner especially in the dense dataset. In addition, it is obvious that the execution time of all algorithms increases when the size



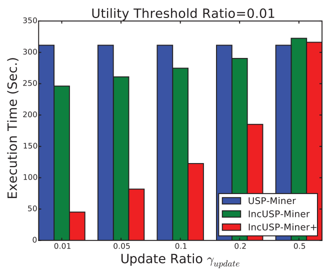
(a) Foodmart



(b) Tafeng

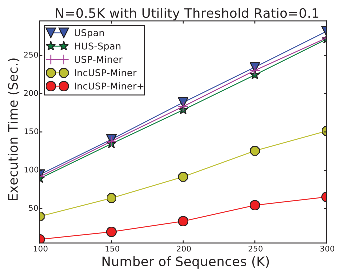


(c) Music

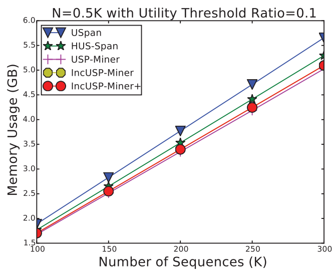


(d) D50C10T5N0.5S4I2.5

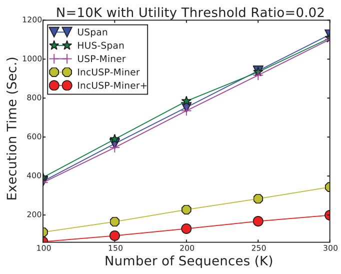
Fig. 5. Effect of the update ratio.



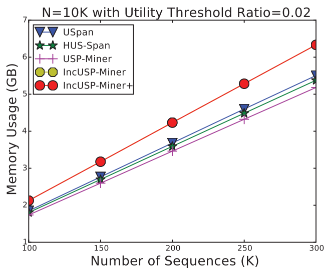
(a) Execution Time on DxC10T5N0.5S4I2.5



(b) Memory Consumption on DxC10T5N0.5S4I2.5



(c) Execution Time on DxC10T5N10S4I2.5



(d) Memory Consumption on DxC10T5N10S4I2.5

Fig. 6. Effect of database size.

of database increases. We can also observe that the increment of IncUSP-Miner$^+$ is more slighter than other algorithms, showing the advantage of the combined use of the proposed node skipping strategy and the database scan reduction strategies.

Moreover, as shown in Figures 6(b) and 6(d), the memory consumption of all algorithms also increases as the size of database increases. It is because when the number of sequences gets larger, all algorithms require more memory space to store their internal data structures. It is interesting to see from Figure 6(b) that the memory spaces of IncUSP-Miner$^+$, IncUSP-Miner, and USP-Miner are smaller than those of USpan and HUS-Span on dataset DxC10T5N0.5S4I2.5. It is intuitive that IncUSP-Miner$^+$, IncUSP-Miner, and USP-Miner require extra memory spaces to store the candidate pattern tree. Fortunately, due to employing tighter upper bound of sequence utility (i.e., TSU), IncUSP-Miner$^+$, IncUSP-Miner, and USP-Miner are able to effectively prune the candidate pattern tree, thereby resulting in acceptable extra memory overhead. Since the candidate pattern trees in IncUSP-Miner$^+$, IncUSP-Miner, and USP-Miner are used to store high TSU sequences in D', D', and D, respectively, IncUSP-Miner$^+$ and IncUSP-Miner require larger memory space than USP-Miner. In addition, IncUSP-Miner$^+$ needs to store some extra information in the candidate pattern tree to support multiple database updates, making the memory usage of IncUSP-Miner$^+$ is slightly higher than that of IncUSP-Miner. As shown in Figure 6(d), IncUSP-Miner$^+$ requires the largest memory space than other algorithms on dataset DxC10T5N0.5S4I2.5. It is because when the number of items gets larger, the size of candidate pattern tree in IncUSP-Miner$^+$ also gets larger, thereby requiring more memory space. Fortunately, due to the pruning effect of TSU, the increment of the memory usage of IncUSP-Miner$^+$ over USpan and HUS-Span on dataset DxC10T5N0.5S4I2.5 is still acceptable.

6 CONCLUSIONS AND FUTURE WORK

In this article, we addressed the problem of incremental mining of HUSPs. Specifically, we introduced the concept of TSU, which is a tight upper bound of sequence utility, and the candidate pattern tree, which is an extension of the lexicographic tree, storing high TSU sequences to facilitate incremental HUSP mining. Accordingly, the node structure on the candidate pattern tree is designed to keep concise auxiliary information to eliminate redundant computation. We then proposed algorithm IncUSP-Miner$^+$ to incrementally mine HUSPs with the aid of the candidate pattern tree. Experimental results showed that IncUSP-Miner$^+$ outperforms the-state-of-the-art incremental HUSP mining algorithm in terms of mining efficiency. In the future, we will re-design IncUSP-Miner$^+$ by utilizing the MapReduce paradigm to support incremental HUSP mining on large -scale databases in distributed environments.

APPENDIX

A PROOF OF LEMMA 3

Proof. Let the length of t be l, the extension item of t be i, and the extension positions of t in s be $p_1, p_2, \ldots, p_k$, where $p_1 < p_2 < \cdots < p_k$. Also, let the parent sequence of t be α and the extension positions of α be $ap_1, ap_2, \ldots, ap_{ak}$, where $ap_1 < ap_2 < \cdots < ap_{ak}$. The proof of this lemma is divided into the following two parts:

Part 1: Proof of $u(t,s) \leq TSU(t,s)$

We first prove $u(t,s) \leq TSU(t,s)$ by considering the following two cases.

- *The case that l=1:*
  Since t is a 1-sequence, i is the only item of t. Suppose that the instance of t in s at extension position $jmax$, where $1 \leq jmax \leq k$, denoted as $inst_t$, is of the largest utility among all t's instances in s. If $jmax = 1$, then we have $u(t,s) = u(i, p_{jmax}, s) = u(i, p_1, s)$. Since t

has only one item, each instance of t at extension position p_j, where $p_j > p_1$, is a subsequence of $rs(i, p_1, s)$. Therefore, we have $u(t, s) \leq ru(i, p_1, s)$ when $jmax > 1$. According to Definition 15, we have $u(t, s) \leq u(i, p_1, s) + ru(i, p_1, s) = TSU(t, s)$.

- *The case that $l \geq 2$:*
  We prove this case by considering the following two conditions:
  —Condition that $u(\alpha, s) \neq u(\alpha, ap_1, s)$:
  In this condition, according to Definition 16, we have $TSU(t, s) = PEU(\alpha, s)$. According to Reference [17], we know that $u(t, s) \leq PEU(\alpha, s)$. Therefore, we have $u(t, s) \leq PEU(\alpha, s) = TSU(t, s)$ in this condition.
  —Condition that $u(\alpha, s) = u(\alpha, ap_1, s)$:
  In this condition, $u(\alpha, ap_1, s)$ is of the largest utility among all instances of α in s. The utility of the instance of item i at position p_1 is $u(i, p_1, s)$. According to Definition 13, instances of i at positions $p_2, p_3, \ldots, p_k$ are in $rs(i, p_1, s)$. Thus, the utility of an instance of i at position p_j, where $1 \leq j \leq k$, is smaller than or equal to $u(i, p_1, s) + ru(i, p_1, s)$. It is clear that $u(t, s)$ equals the utility of an instance of α in s plus the utility of an instance of i at position p_j where $1 \leq j \leq k$. Therefore, we have $u(t, s) \leq u(\alpha, ap_1, s) + u(i, p_1, s) + ru(i, p_1, s) = TSU(t, s)$.

Part 2: Proof of $u(t', s) \leq TSU(t, s)$

Suppose that t' is of length $l' = l + \Delta_l$. Let the extension positions of t' in s be $p'_1, p'_2, \ldots, p'_{k'}$ where $p'_1 < p'_2 < \cdots < p'_{k'}$. Since t' is a descendant sequence of t and α is the parent sequence of t, t' equals $\alpha \cdot i \cdot i_1 \cdot i_2 \cdots i_{\Delta_l}$. We then prove $u(t', s) \leq TSU(t, s)$ by considering the following two conditions.

- *Case that $l = 1$:*
  It is clear that $u(t', s)$ is the sum of "the utility of an instance of t at extension position p_j where $1 \leq j \leq k$" and "the utility of an instance of each item $i_{j'}$, where $j' = 1, 2, \ldots, \Delta_l$, at positions larger than or equal to p_j." Since t is of length 1, the utility of the t's instance at extension position p_1 is $u(i, p_1, s)$. The t's instances at extension positions p_j, where $2 \leq j \leq k$ are contained in $rs(i, p_1, s)$. In addition, the instances of $i_{j'}$, where $j' = 1, 2, \ldots, \Delta_l$, at extension positions larger than or equal to p_j are also contained in $rs(i, p_1, s)$. Thus, we have $u(t', s) \leq u(i, p_1, s) + ru(i, p_1, s) = TSU(t, s)$.
- *Case that $l \geq 2$:*
  —Condition that $u(\alpha, s) \neq u(\alpha, ap_1, s)$:
  Definition 16 shows that $TSU(t, s) = PEU(\alpha, s)$ in this condition. In addition, Reference [17] shows that $u(t', s) \leq PEU(t, s)$ while Lemma 2 shows that $PEU(t, s) \leq PEU(\alpha, s)$. As a result, we have $u(t', s) \leq PEU(t, s) \leq PEU(\alpha, s) = TSU(t, s)$, thereby proving $u(t', s) \leq TSU(t, s)$ in this condition.
  —Condition that $u(\alpha, s) = u(\alpha, ap_1, s)$:
  Definition 16 shows that $TSU(t, s) = u(\alpha, ap_1, s) + u(i, p_1, s) + ru(i, p_1, s)$ in this condition. Since t' is a descendant sequence of α, we have $ap_1 \leq p'_1$. Suppose the instance of t' in s at extension position $p'_{jmax'}$, denoted as $inst_{t'}$, is of the largest utility among all instances of t' in s. Thus, $u(t', s) = u(t', p'_{jmax'}, s)$. We can get an instance of t, denoted as $inst_t$, and an instance of α, denoted as $inst_\alpha$, by removing the latest Δ_l and $\Delta_l + 1$ items, respectively, from $inst_{t'}$. Let the extension position of $inst_t$ be p_{jt} where $1 \leq jt \leq k$. Also let the extension position of $inst_\alpha$ be ap_{aj} where $1 \leq aj \leq ak$. Since $u(\alpha, s) = u(\alpha, ap_1, s)$, we have $u(\alpha, ap_{aj}, s) \leq u(\alpha, ap_1, s)$. If $jt = 1$, then the utility of the instance of i in $inst_t$ is $u(i, p_{jt}, s) = u(i, p_1, s)$. Otherwise, the instance of i in $inst_t$ is contained in $rs(i, p_1, s)$. In

addition, the latest Δ_l items of $inst_{t'}$ are also contained in $rs(i, p_1, s)$. Thus, the utility of the latest $\Delta_l + 1$ items of $inst_{t'}$ are smaller than or equal to $u(i, p_1, s) + ru(i, p_1, s)$.

It is clear that the utility of $inst_{t'}$ (i.e., $u(t', p'_{jmax'}, s)$) equals the utility of $inst_\alpha$ (i.e., $u(\alpha, ap_{aj}, s)$) plus the utility of the latest $\Delta_l + 1$ items in $inst_{t'}$. As a result, we can derive $u(t', s) = u(t', p'_{jmax'}, s) \leq u(\alpha, ap_1, s) + u(i, p_1, s) + ru(i, p_1, s) = TSU(t, s)$, thereby proving $u(t', s) \leq TSU(t, s)$ in this condition.

Finally, this lemma is proven by the above two-part proof. □

REFERENCES

[1] C. F. Ahmed, S. K. Tanbeer, and B.-S. Jeong. 2010. A novel approach for mining high-utility sequential patterns in sequence databases. *ETRI J.* 32, 5 (2010), 676–686.

[2] C. F. Ahmed, S. K. Tanbeer, and B.-S. Jeong. 2011. A framework for mining high utility web access sequences. *IETE Tech. Rev.* 28, 1 (2011), 3–16.

[3] C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, and Y.-K. Lee. 2009. Efficient tree structures for high utility pattern mining in incremental databases. *IEEE Trans. Knowl. Data Eng.* 21, 12 (2009), 1708–1721.

[4] J. Ayres, J. Gehrke, T. Yiu, and J. Flannick. 2002. Sequential pattern mining using a bitmap representation. In *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining.* 429–435.

[5] Y. Chen, J. Guo, Y. Wang, Y. Xiong, and Y. Zhu. 2007. Incremental mining of sequential patterns using prefix tree. In *Proceedings of the 11th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining.*

[6] H. Cheng, X. Yan, and J. Han. 2004. IncSpan: Incremental mining of sequential patterns in large database. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

[7] M. N. Garofalakis, R. Rastogi, and K. Shim. 1999. SPIRIT: Sequential pattern mining with regular expression constraints. In *Proceedings of the 25th International Conference on Very Large Data Bases.* 223–234.

[8] C. Kim, J.-H. Lim, R. T. Ng, and K. Shim. 2007. SQUIRE: Sequential pattern mining with quantities. *J. Syst. Software* 80, 10 (2007), 1726–1745.

[9] G.-C. Lan, T.-P. Hong, V. S. Tseng, and S.-L. Wang. 2014. Applying the maximum utility measure in high utility sequential pattern mining. *Expert Syst. Appl.* 41, 11 (2014), 5071–5081.

[10] S. N. Nguyen, X. Sun, and M. Orlowska. 2005. Improvements of IncSpan: Incremental mining of sequential patterns in large database. In *Proceedings of the 9th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining.*

[11] S. Parthasarathy, M. J. Zaki, M. Ogihara, and S. Dwarkadas. 1999. Incremental and interactive sequence mining. In *Proceedings of the 8th International Conference on Information and Knowledge Management.*

[12] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. 2004. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Trans. Knowl. Data Eng.* 16, 11 (2004), 1424–1440.

[13] B.-E. Shie, H.-F. Hsiao, V. S. Tseng, and P. S. Yu. 2011. Mining high utility mobile sequential patterns in mobile commerce environments. In *Proceedings of the 16th International Conference on Database Systems for Advanced Applications.* 224–238.

[14] R. Srikant and R. Agrawal. 1996. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology.* 3–17.

[15] J. Wang and J. Han. 2004. BIDE: Efficient mining of frequent closed sequences. In *Proceedings of the 20th IEEE International Conference on Data Engineering.* 79–90.

[16] J.-Z. Wang and J.-L. Huang. 2016. Incremental mining of high utility sequential patterns in incremental databases. In *Proceedings of ACM International Conference on Information and Knowledge Management.* 2341–2346.

[17] J.-Z. Wang, J.-L. Huang, and Y.-C. Chen. 2016. On efficiently mining high utility sequential patterns. *Knowl. Info. Syst.* 49, 2 (2016).

[18] X. Yan, J. Han, and R. Afshar. 2003. CloSpan: Mining closed sequential patterns in large datasets. In *Proceedings of the 3rd SIAM International Conference on Data Mining.* 166–177.

[19] J. Yin, Z. Zheng, and L. Cao. 2012. USpan: An efficient algorithm for mining high utility sequential patterns. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* 660–668.

[20] J. Yin, Z. Zheng, L. Cao, Y. Song, and W. Wei. 2013. Efficiently mining top-K high utility sequential patterns. In *Proceedings of the 13rd IEEE International Conference on Data Mining.* 1259–1264.

Received June 2017; revised October 2017; accepted January 2018