Efficiently Mining Top-K High Utility Sequential Patterns

Junfu Yin, Zhigang Zheng, Longbing Cao, Yin Song, Wei Wei
Advanced Analytics Institute
University of Technology, Sydney, Australia
{*Junfu.Yin, Yin.Song, Wei.Wei-7*}*@student.uts.edu.au*
{*Longbing.Cao, Zhigang.Zheng*}*@uts.edu.au*

***Abstract*—High utility sequential pattern mining is an emerging topic in the data mining community. Compared to the classic frequent sequence mining, the utility framework provides more informative and actionable knowledge since the utility of a sequence indicates business value and impact. However, the introduction of "utility" makes the problem fundamentally different from the frequency-based pattern mining framework and brings about dramatic challenges. Although the existing high utility sequential pattern mining algorithms can discover all the patterns satisfying a given *minimum utility*, it is often difficult for users to set a proper minimum utility. A too small value may produce thousands of patterns, whereas a too big one may lead to no findings. In this paper, we propose a novel framework called *top-k high utility sequential pattern mining* to tackle this critical problem. Accordingly, an efficient algorithm, Top-k high Utility Sequence (TUS for short) mining, is designed to identify top-k high utility sequential patterns without minimum utility. In addition, three effective features are introduced to handle the efficiency problem, including two strategies for raising the threshold and one pruning for filtering unpromising items. Our experiments are conducted on both synthetic and real datasets. The results show that TUS incorporating the efficiency-enhanced strategies demonstrates impressive performance without missing any high utility sequential patterns.**

***Keywords*-High utility sequential pattern mining; Top-K sequential pattern mining**

I. Introduction

Frequent sequential pattern mining [**?**], as one of the fundamental research topics in data mining, discovers frequent subsequences in sequence databases. It is very useful for handling order-based business problems, and has been successfully adopted to various domains and applications such as complex behavior analysis [**?**] and gene sequence analysis [**?**], [**?**], [**?**]. In the frequency-based framework for typical sequence analysis, the *downward closure property* (also known as *Apriori property*) [**?**] plays a fundamental role in identifying frequent sequential patterns.

However, taking the frequency to measure pattern interestingness may be insufficient for selecting actionable sequences associated with expected quality and business impact, because the patterns identified under the frequency (support) framework do not disclose the business value and impact. To solve the above problems, the concept *utility* is introduced into sequential pattern mining to select sequences of high utility by considering the quality and value (such as profit) of itemsets. This leads to an emerging area, *high utility pattern mining* [**?**], [**?**], [**?**], [**?**], [**?**] and *high utility sequential pattern mining* [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], which selects interesting patterns / sequential patterns based on minimum utility rather than minimum support. The utility-based patterns are proven to be more informative and actionable for decision-making than the frequency-based ones [**?**]. For instance, in [**?**], [**?**], the authors discuss the extraction of profitable behaviors from the mobile commerce environments. [**?**] proposes methods to mine high utility sequences from web logs by assigning each page an impact/significance. In [**?**], a USpan algorithm is built for utility-based sequential pattern mining satisfying a predefined minimum utility.

Although algorithms such as USpan can obtain high utility sequences based on a given minimum utility, it is very difficult for users to specify an appropriate minimum utility threshold and to directly obtain the most valuable patterns. This is because the complexity of utility-based sequence databases (which may be different from the classic itemsets), determining multiple factors including the distribution of the items and utilities, density of the database, lengths of the sequences, and so on. Consequently, it is not surprising that, with a same minimum utility threshold, some datasets may produce millions of patterns while others may contribute nothing. The challenge here is that it may not be doable to tune the threshold to capture the expected number of patterns. This is because the sensitivity of the threshold makes it hard to tune for a variety of databases. It may be very costly and time consuming to achieve the proper threshold for the desired patterns.

In fact, the classic frequency/support based pattern mining also faces the same challenge. Accordingly, the concept of extracting top-k patterns has been proposed in [**?**], [**?**], [**?**], [**?**] to select the patterns with the highest frequency. In the top-k frequent pattern mining, instead of letting a user specify a threshold, the top-k pattern selection algorithms allow a user to set the number of top-k high frequency patterns to be discovered. This makes it much easier and more intuitive and practical than determining a minimum support; also the determination of k by a user is more straightforward than considering data characteristics, which are often invisible to users, for choosing a proper threshold.

The easiness for users to determine k does not indicate the simplicity of developing an efficient algorithm for selecting top-k high utility sequential patterns. In the utility framework, TKU [**?**] is the only method for mining top-k high utility itemsets, to the best of our knowledge. No work is reported on mining top-k high utility sequences. There is significant difference between top-k utility itemset mining and top-k utility sequence mining in which the order between itemsets is considered. In fact, the problem of top-k high utility sequence mining is much more challenging than mining top-k high utility itemsets. First, as with high utility itemset mining, the downward closure property does not hold in the utility-based sequence mining. This means that the existing top-k frequent sequential pattern mining algorithms [**?**] cannot be directly applied. Second, compared to top-k high utility itemset mining [**?**], utility-based sequence analysis faces the critical combinational explosion and computational complexity caused by sequencing between itemsets. This means that the techniques in [**?**] cannot be directly transferred to top-k high utility sequential pattern mining either. Third, since the minimum utility is not given in advance, the algorithm essentially starts the searching from 0 minimum support. This not only incurs very high computational costs, but also the challenge of how to raise the minimum threshold without missing any top-k high utility sequences.

To address the above challenges, this paper proposes an efficient algorithm to identify Top-k Utility Sequences (TUS). The contributions of this work are as follows.

- We propose a novel framework for extracting the top-k high utility sequential patterns. A baseline algorithm TUSNaive is provided accordingly.
- Three strategies are proposed for effectively raising the thresholds at different stages of the mining process.
- Substantial experiments on both synthetic and real datasets show that the TUS algorithm can efficiently identify top-k high utility sequences from large scale data with large k.

The remainder of the paper is organized as follows. Section 2 defines the problem of mining top-k high utility sequential patterns. Section 3 details the TUS algorithm. Experimental results and evaluation are presented in Section 4. Section 5 concludes the work.

II. Problem Statement

Let $\mathcal{I} = \{i_1, i_2, ..., i_n\}$ be a set of distinct *items*. A *utility item*, or *u-item*, is an ordered pair (i, u), where $i \in \mathcal{I}$ represents an item and u is a positive number representing the *utility* of i, e.g. the profit of i. A *utility itemset*, or *u-itemset*, consists of no less than one u-item, which is denoted and defined as $l = [(i_{j_1}, u_1)(i_{j_2}, u_2)...(i_{j_{n'}}, u_{n'})]$, where (i_{j_k}, u_k) is a u-item for $1 \leqslant k \leqslant n'$, and $\forall k_1, k_2$, where $1 \leqslant k_1, k_2 \leqslant n'$ and $k_1 \neq k_2$, $i_{j_{k_1}} \neq i_{j_{k_2}}$. For brevity, the brackets are omitted if a u-itemset has only one u-item. Since the items in an itemset can be listed in any order. Without loss of generality, we assume that u-items are listed in the alphabetical order. A *utility sequence*, or *u-sequence*, is an ordered list of u-itemsets, which is denoted and defined as $s = \langle l_1 l_2 \ ... \ l_m \rangle$, where $l_k (1 \leqslant k \leqslant m)$ is a u-itemset. A *u-sequence database* $\mathcal{S}$ consists of sets of tuples $\langle sid, s \rangle$, where sid is a unique identifier of the u-sequence s. Readers can refer to [**?**] for more details.

Table I
U-Sequence Database

SID	TID	Transactions	TU	SU
1	1	(a,6)(d,8)(e,1)	15	112
1	2	(b,10)(c,16)(f,2)	28	
1	3	(a,12)(d,4)	16	
1	4	(a,6)(b,5)(f,3)	14	
1	5	(a,21)(d,12)(f,6)	39	
2	1	(c,20)(d,4)	24	117
2	2	(a,3)(b,5)(c,16)(f,5)	29	
2	3	(c,8)(d,10)(e,3)	21	
2	4	(f,6)	6	
2	5	(b,20)(e,1)(f,1)	22	
2	6	(a,6)(d,8)(f,1)	15	
3	1	(a,18)(c,16)(d,10)	44	105
3	2	(a,6)(b,5)(f,6)	17	
3	3	(d,4)	4	
3	4	(b,10)(c,4)(e,5)	19	
3	5	(c,4)(d,6)(e,4)	14	
3	6	(b,5)(d,2)	7	

Definition 1: (Sequence maximum utility) Because a sequence may have multiple utility values in the u-sequence context, we choose the *maximum utility* [**?**] as the sequence's utility. The *maximum utility* of a sequence t is denoted and defined as $u_{max}(t)$:

$$u_{max}(t) = \sum \max\{u(s') | s' \sim t \wedge s' \subseteq s \wedge s \in \mathcal{S}\} \quad (1)$$

For example, the utility of $\langle (ad)a \rangle$ is $\{\{26, 20, 35, 22, 37\}, \{34\}\}$. The maximum utility is $u_{max}(\langle (ad)a \rangle) = 37 + 34 = 71$. For brevity, we use *sequence utility* to represent the maximum utility of a sequence, i.e., u_{max}, for the rest of the paper.

Definition 2: (Top-k high utility sequential patterns) A sequence t is called a *top-k high utility sequence* if there are less than k sequences whose utilities are no less than $u_{max}(t)$. The *optimal minimum utility* is denoted and defined as $\xi^* = min\{u_{max}(t) | t \in \mathcal{T}\}$, where $\mathcal{T}$ means the set of top-k high utility sequences. Given a u-sequence database $\mathcal{S}$ and a number k, the problem of finding the complete set of top-k high utility sequential patterns in $\mathcal{S}$ is to discover all the itemsets whose utilities are no less than ξ^* in $\mathcal{S}$.

Example 1: Suppose the desired k number of high utility sequences is set to 7, the top 7 high utility sequences in Table I are shown in Table **??**. The optimal minimum utility threshold $\xi^* = min\{151, 152, 152, 156, 156, 159, 163\} = 151$. If k is set to 3, then only sequences $\langle d(bcf)d(bf)(adf) \rangle$ and $\langle d(bcf)db(adf) \rangle$ are obtained, and $\xi^* = 159$. The reason of excluding $\langle d(bc)d(bf)(adf) \rangle$ and $\langle d(bcf)d(bf)(ad) \rangle$ is to control the number of the candidates no more than k = 3.

Table II
TOP 7 HIGH UTILITY SEQUENCES IN TABLE **??**

ID	Top-k high utility	SU
1	$\langle d(bcf)d(bf)(adf)\rangle$	163
2	$\langle d(bcf)db(adf)\rangle$	159
3	$\langle d(bc)d(bf)(adf)\rangle$	156
4	$\langle d(bcf)d(bf)(ad)\rangle$	156
5	$\langle d(bc)db(adf)\rangle$	152
6	$\langle d(bcf)db(ad)\rangle$	152
7	$\langle (bcf)d(bf)(adf)\rangle$	151

III. THE TUS ALGORITHM

In the previous section, we define the top-k high utility sequential pattern mining framework. In this section, we specify and present an efficient algorithm, TUS, for mining top-k high utility sequential patterns. Firstly, we present a baseline approach named TUSNaive. Then we present a tight utility boundary for sequences, which substantially reduces the search space. In the end, we provide a very efficient pre-insertion strategy, which effectively raises the minimum utility threshold.

A. TUSNaive: The Baseline Algorithm

Here we present a baseline algorithm called TUSNaive to extract the top-k sequences with the highest utilities. Instead of using a user specified minimum utility, TUSNaive engages a structure named *TUSList* to maintain the top-k high utility sequences on-the-fly.

TUSList is a fixed-size sorted list which is used to maintain the top-k high utility sequential patterns dynamically, and a minimum utility ξ of it is set to prune the unpromising candidates in the mining process. The mechanism can be briefed as follows. Initially, the TUSList is empty and ξ is set to 0. In this stage, whenever a candidate sequence comes, it will be inserted into TUSList, and ξ stays on 0. Once the k candidates are found, ξ is raised to the utility of the last candidate (i.e. the least utility canidate) in TUSList. After that, a candidate satisfying ξ is inserted into TUSList, then the least utility candidate(s) will be eliminated. ξ is thereafter raised to utility of the updated last candidate. The process continues until no candidate matches ξ, and those remain in the TUSList are the target patterns. The pseudo code of TUSNaive is shown in Algorithm **??**.

Algorithm 1: TUSNaive($p, \mathcal{S}$)

```
1 Scan S for items to be concatenated to p;
2 for Each of the items do
3     Let i be the item, p' = p + i and S' = S(p');
4     if u(p') > TUSList.ξ then
5         p' → TUSList;
6     TUSNaive(p', S');
7 return TUSList;
```

B. Pre-insertion

Although TUSNaive correctly extracts the top-k high utility sequences, it traverses too many invalid sequence candidates since the minimum threshold starts from 0. This directly degrades the performance of the mining task. To overcome this problem, we further propose three effective strategies, i.e., two for raising the minimum utility threshold and one for reducing the search space, to improve the performance. We start from the pre-insertion strategy.

***Strategy** 1:* (Pre-insertion) The pre-insertion strategy inserts the utilities of both the 1-sequences and u-sequences to the TUSList before the mining process.

The pre-insertion is an effective strategy for rasing the minimum utility in TUSList. After the raw sequences are successfully stored in the memory, it needs to calculate the utility of each sequence. In this phase, we use a hash table to record the maximum utility of every distinct item in the sequences. For example, in Table **??**, the maximum utility of a in s_1 is the utility of a_5, i.e. 21. The other maximum utilities are $\{b : 10, c : 16, d : 12, e : 1, f : 6\}$. After s_1 is finished, s_1 itself will be inserted into the TUSList, and labeled as a u-sequence. The purpose is to prevent the sequence from being double-inserted, otherwise it will miss truly top-k high utility sequences. Similarly, after s_2 and s_3 are scanned, the 1-sequences will be calculated and added to the hash table, and both s_2 and s_3 will be inserted into the TUSList. After the sequences are scanned, the hash table is {a:45, b:40, c:52, d:32, e:9, f:18}. All of the items are actually 1-sequences, they are all inserted into the TUSList. With the three u-sequences and their utilities, the utilities in the TUSList is $\{117, 112, 105, 52, 45, 40, 32\}$, and the minimum utility $\xi = 32$ after pre-insertion.

As seen from the example, the pre-insertion strategy effectively raises the minimum threshold to a reasonable level before mining, and prevents from generating unpromising candidates.

C. Sorting concatenation order

The *sorting concatenation order* strategy is applied in the main mining process. It effectively identifies potential high utility sequences, and the utilities can be calculated and inserted into TUSList prior to those low utility sequences. As a result, the minimum utility ξ quickly raises to ξ^* without traversing too many invalid candidates, and the efficiency is therefore substantially improved. Now we discuss the method in details.

The term *concatenation* means an item is appended to a k-sequence to obtain a (k+1)-sequence. There are two types of concatenations: I-Concatenation and S-Concatenation. S-Concatenation means an item is appended as a 1-itemset to the sequence, while I-Concatenation means an item is added to the last itemset of the sequence. For example, c S-Concatenates to $\langle b\rangle$ result in $\langle bc\rangle$, and the I-Concatenation leads to $\langle (bc)\rangle$. More examples are in Figure **??**.

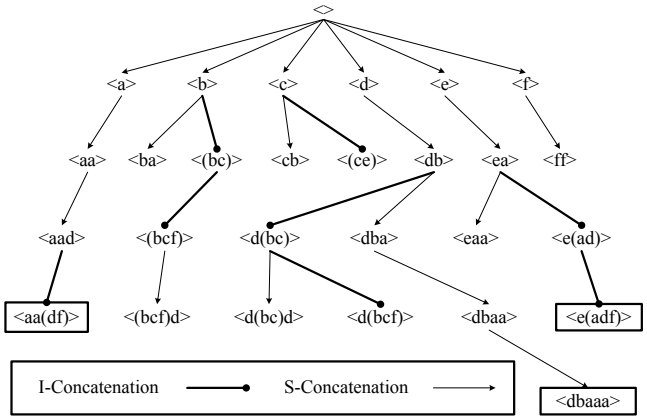


Figure 1. The concatenations for the examples in Table **??**

A sequential pattern mining algorithm usually follows a pattern-growth method to mine the expected sequences. The proposed TUS and TUSNaive algorithm, as mentioned above, are in this class. For example, in Figure **??**, one of the searching paths is $\langle\rangle \rightarrow \langle a\rangle \rightarrow \langle aa\rangle \rightarrow \langle aad\rangle \rightarrow ...$ Once this path is over, it will recursively search the other branches until no more candidates left.

Suppose we are standing on the root $\langle\rangle$. We have 6 different ways, that is a to f, to choose to continue the mining process. Which one should we concatenate to the root first? Once the first candidate and its offsprings are finished checking, what order should be applied to the others? Does it make difference? In threshold-based high utility sequence mining, there is no such concerns. Because the minimum utility ξ is a fixed number from the beginning to the end. It means that whatever order is used, the candidates they checked are always the same. However, in the top-k framework, the order of concatenating items does matter. Since ξ is dependent on the candidate inside the TUSList, we should put the high utility candidates to TUSList as soon as possible so that ξ increases to ξ^* shortly. Now we present a few definitions to illustrate the concepts.

Definition 3: (Ending u-item and pivot) Suppose that all the (u-)items in the (u-)sequences are listed alphabetically. Let $s = \langle l_1 l_2 ... l_n\rangle$ be a u-sequence, $t = \langle t_1 t_2 ... t_m\rangle$ be a sequence and $s \simeq t$. Assume that $s_a = \langle l_{a_1} l_{a_2} ... l_{a_m}\rangle$, where $l_{a_m} = [(i_{p_1}, u_{p_1})(i_{p_2}, u_{p_2})...(i_{p_q}, u_{p_q})]$, $s_a \subseteq s$ and $s_a \sim t$. (i_{p_q}, u_{p_q}) is called the *ending u-item* of t in s. Additionally, (i_{p_q}, u_{p_q}) is called *pivot* or *projection point* iff there is no $s_b = \langle l_{b_1} l_{b_2} ... l_{b_m}\rangle$ where $s_b \subseteq s$ and $s_b \sim t$ such that $b_m < a_m$.

For example, the ending u-items of $\langle(ad)a\rangle$ in s_1 are a_3, a_4 and a_5, where the pivot is a_3. For $\langle d(bf)\rangle$ in s_1, the ending u-items are f_2 and f_4.

Definition 4: (Ending u-item maximum utility) The ending u-item maximum utility is denoted and defined as

$$u(t, i, s) = max\{u(s')|s' \sim t \wedge s' \subseteq s \wedge i \in s'\} \quad (2)$$

where t is a sequence, s is a u-sequence, i is an ending u-item of t in s. Specifically, we use $u_p(t, s)$ to denote the pivot maximum utility, i.e.

$$u_p(t, s) = u(t, i_p, s) \quad (3)$$

where i_p is the pivot.

For example, $u(\langle(ad)a\rangle, a_3, s_1) = max(26) = 26$, $u(\langle(ad)a\rangle, a_4, s_1) = max(20, 22) = 22$ and $u(\langle(ad)a\rangle, a_5, s_1) = max(35, 37) = 37$. Obviously, the ending u-item utility of a sequence is a subset of the utility of itself. The pivot maximum utility of $\langle(ad)a\rangle$ is $u_p(\langle(ad)a\rangle, s_1) = u(\langle(ad)a\rangle, a_3, s_1) = 26$.

Definition 5: (Sequence-Projected Utilization) The Sequence-Projected Utilization (SPU) of a sequence t in $\mathcal{S}$ is denoted and defined as $SPU(t)$

$$SPU(t) = \sum_{i \in s \wedge s \in \mathcal{S}} (u_{rest}(i, s) + u_p(t, s)), \quad (4)$$

where i is the pivot of t in s, and

$$u_{rest}(i, s) = \sum_{i' \in s \wedge i \prec i'} u(i') \quad (5)$$

u_{rest} means the sum of the utilities of the u-items after the pivot (exclusive). For example, $u_{rest}(a_5, s_1) = u(d, 12) + u(f, 6) = 12 + 6 = 18$ and $u_{rest}(f_5, s_1) = 0$. The meaning of SPU is the pivot utility plus the rest u-sequence utility. For example, $SPU(\langle a\rangle, s_1) = u_p(\langle a\rangle, s_1) + u_{rest}(a_1, s_1) = 6 + 106 = 112$. Similarly, $SPU(\langle a\rangle, s_2) = 3 + 90 = 93$ and $SPU(\langle a\rangle, s_3) = 18 + 87 = 105$. Therefore, $SPU(\langle a\rangle) = 112 + 93 + 105 = 310$.

Definition 6: (Item concatenation order) Given a sequence t and two items a, b. Let t_a and t_b be the sequences of a and b concatenated to t respectively, where $t_a \neq t_b$. We say a is prior to b, denoted as $a \rhd b$, if either of following conditions is true:

- $SPU(t_a) > SPU(t_b)$, or
- $SPU(t_a) = SPU(t_b)$ and
  - i) t_a is I-Concatenated from t, and t_b is S-Concatenated from t, or
  - ii) both t_a and t_b are I-Concatenated or S-Concatenated from t, but a is alphabetically smaller than b.

Generally, $a \rhd b$ means that the utilities of t_a's offspring candidates are likely higher than that of t_b. Taking $\langle aa\rangle$ in Figure **??** as an example, in u-sequence 1, items b, d and f can be I-Concatenated to $\langle aa\rangle$, and the SPUs are 17, 22 and 15 respectively. Obviously, $d \rhd b \rhd f$ holds in u-sequence 1. It also reveals the fact that $d_3 \prec b_4 \prec f_4$. The S-Concatenation is also in the situation. For example, since $a_4 \prec b_4 \prec f_4 \prec d_5$, we can easily tell $a \rhd b \rhd f \rhd d$ without calculating. Basically, in a single u-sequence, $a \prec b$ means that a has more remaining utility than b, so a should be concatenated earlier than b. Back to the database, although $SPU(t_a) > SPU(t_b)$ may not mean $a \prec b$ in all

u-sequences, it reflects that t_a projected less utility than t_b in the database. When $SPU(t_a) = SPU(t_b)$, we apply the normal sequential pattern growth rules. In our experience, there is nearly no chance for two concatenation items have the same SPU. In most of time, $SPU(t_a)$ is either higher or lower than $SPU(t_b)$. Based on Definition **??**, we present the strategy below.

***Strategy** 2:* (Sorting concatenation order) Given a sequence t, and the items can be concatenated to t are $a_1, a_2, ..., a_n$. Then $a_{k_1}, a_{k_2}, ..., a_{k_n}$ is the order to be concatenated to t, where $a_{k_1} \rhd a_{k_2} \rhd ... \rhd a_{k_n}$.

For example, assume $t = \langle a \rangle$. Items b, c, d, e, f are able to I-Concatenate to $\langle a \rangle$, and the SPUs of $\langle (ab) \rangle$, $\langle (ac) \rangle$, $\langle (ad) \rangle$, $\langle (ae) \rangle$ and $\langle (af) \rangle$ are 192, 172, 186, 98 and 161 respectively. Similarly, a to f can be S-Concatenated to $\langle a \rangle$, and the corresponding SPUs are 145, 189, 181, 157, 72 and 164. Therefore, the items concatenation order for $\langle a \rangle$ is $b_i \rhd b_s \rhd d_i \rhd d_s \rhd c_i \rhd f_s \rhd f_i \rhd d_s \rhd a_s \rhd e_i$. [1]

D. Sequence-Reduced Utility

In this part, we provide a tighter sequence boundary and a novel pruning strategy. The following example illustrates the problem.

Definition 7: (Sequence-Reduced Utilization) Let $\mathcal{I} = \{i_1, i_2, ..., i_n\}$ be a set of distinct items, and $\mathcal{B}, \mathcal{W} \subseteq \mathcal{I}$, where $\mathcal{B}$ is named as *blacklist* and $\mathcal{W}$ is named as *whitelist*, and $\mathcal{B} \cup \mathcal{W} = \mathcal{I}$, $\mathcal{B} \cap \mathcal{W} = \emptyset$. Given a sequence t, a u-sequence $s = \langle l_1 l_2 ... l_n \rangle$. Suppose $s \simeq t$, let i_p be the pivot of t in s and $i_p \in l_m$, where $1 \leq m \leq n$. The *SRU*(Sequence-Reduced Utilization) of a sequence t in s is denoted and defined as $SRU(t, s)$

$$SRU(t,s) = u_p(t,s) + \sum_{i_p \prec i' \wedge i' \in l_m} u(i') + \sum_{k=m+1}^{n} \sum_{i' \in l_k \wedge i' \in \mathcal{W}} u(i')$$

Given an item i, the SRU of i in t's projection database is denoted and defined as

$$SRU(i,t) = \sum \{SRU(t,s) | i' \sim i \wedge i' \in l_k \wedge s \in \mathcal{S}\} \quad (6)$$

Item $i \in \mathcal{W}$ if and only if $SRU(i,t) \geq \xi$.

***Strategy** 3:* Keep refreshing the blacklist, until all the items in the whitelist satisfy $SRU(i,t) \geq \xi$.

IV. Experiments

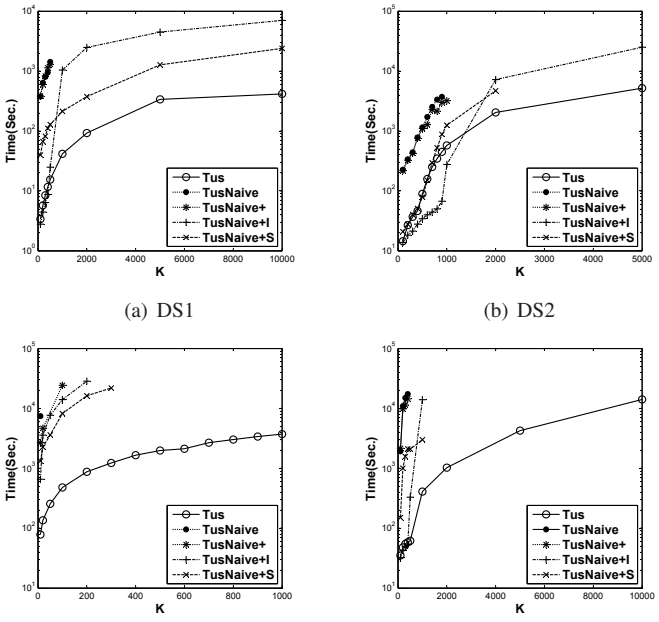
In this section, we evaluate the performance of TUS on a variety of datasets. Since there is no algorithm can solve the top-k high utility sequence mining, and it is not easy to upgrade the existing method such as [**?**] either, we thus compare TUS with TUSNaive, which is a baseline approach without pre-insertion, sorting and SRU as described in Section **??**, **??** and **??** respectively. TUSNaive with different strategies are also compared.

[1] b_i and b_s means I-Concatenation and S-Concatenation respectively, similar to the others.

Table III
Characteristics of the Synthetic Datasets

Characteristics	DS1 [?]	DS2 [?]
Average itemset per sequence C	10	8
Average items per itemset T	2.5	2.5
Average itemsets in maximum sequences S	10	6
Average items in maximum sequences I	2.5	2.5
Number of sequences D	100k	10k
Number of different items N	1k	10k

We conduct experiments on 2 synthetic (DS1 and DS2, which are generated as [**?**] using the settings in Table **??**) and 2 real datasets (DS3 [**?**] and DS4 [**?**]) to compare the efficiency of TUS with TUSNaive, in terms of computational costs on different data sizes and data characteristics. To make the top-k and threshold based approaches comparable, we run top-k approaches first. After getting the utility of the k-th pattern, that is the optimal minimum utility in Definition **??**, we use this value as the minimum threshold for running the threshold-based methods. The TUS algorithm is implemented in C++ of Microsoft Visual Studio 2010. All experiments are conducted on a virtual machine in a server with Intel Xeon CPU of 3.10GHz, 8GB memory and Windows 8 system.



(a) DS1 (b) DS2

(c) DS3 (d) DS4

Figure 2. Execution Time of Different Stategies

We conduct experiments to evaluate the performance of TUS, in terms of computational costs, on datasets *DS1* to *DS4*. Different strategies were compared to show their corresponding performance on the datasets as well. In Figure

??, TUSNaive+ refers to TUNaive with SRU; TUSNaive+I refers to TUNaive with SRU and pre-insertion; TUSNaive+S refers to TUNaive with SRU and sorting.

The results show that TUS is generally 10 - 1000+ times faster than TUSNaive. For DS4 TUSNaive cannot finish the mining with a very small k (with k = 20) in 24+ hours. Besides, the gap between TUS and TUSNaive increases with the increase of k. The results also show that TUS, TUSNaive+I and TUSNaive+S are generally faster than TUSNaive+ from Figure **??**. That indicates the proposed three optimization measures, including SRU, sorting and pre-insertion, are effective for top-k pattern mining.

Generally, TUSNaive+I is faster than TUSNaive+S when k is small. After k exceeds a certain number, TUSNaive+S outperforms TUSNaive+I. For example, k = 1000 in Figure **??**, Figure **??**, k = 2000 in Figure **??**, k = 20 in Figure **??**. This is because TUSNaive+S starts the mining from 0 while TUSNaive+I does not, pre-insertion directly prunes unpromising branches than the sorting strategy. The sorting strategy always traverses the higher estimated utility candidates first. This guarantees ξ raising to ξ^* shortly, while TUSNaive+I does not. So when k is large, sorting is better than pre-insertion.

V. Conclusions

In this paper, we have proposed an efficient algorithm named TUS for mining top-k high utility sequential patterns from utility-based sequence databases. TUS guarantees there is no sequence missed during the mining process. We have developed a new sequence boundary and a corresponding pruning strategy for effectively filtering the unpromising candidates. Moreover, a pre-insertion and a sorting strategy has been introduced to raise the minimum utility threshold. The mining performance is enhanced significantly since both the search space and the number of candidates are effectively reduced by the proposed strategies. Both synthetic and real datasets have been used to evaluate the performance of TUS, which is shown to substantially outperform the baseline algorithms, and the performance of TUS is close to the optimal case of the state-of-the-art utility sequential pattern mining algorithms.

References

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

[2] C. F. Ahmed, S. K. Tanbeer and B. Jeong, *Mining High Utility Web Access Sequences in Dynamic Web Log Data*, SNPD 2010, pp.76-81.

[3] C. F. Ahmed, S. K. Tanbeer and B. Jeong, *A Novel Approach for Mining High-Utility Sequential Patterns in Sequence Databases*, ETRI, 2010, vol.32, pp.676-686.

[4] C. F. Ahmed, S. K. Tanbeer, B. Jeong and T. Lee, *Efficient tree structures for high utility pattern mining in incremental databases*, TKDE, 2009, vol. 21, pp. 1708-1721.

[5] L. Cao. *Actionable Knowledge Discovery and Delivery*, WIREs Data Mining and Knowledge Discovery, 2012, Vol. 2, Issue 2, pp. 149 - 163.

[6] L. Cao and P. Yu, *Behavior Computing*, Springer, 2012.

[7] Y. L. Cheung and A. W. Fu, *Mining frequent itemsets without support threshold: with and without item constraints*, TKDE, 2004, Vol. 16, pp. 1052-1069.

[8] K. Chuang, J. Huang and M. Chen, *Mining Top-K Frequent Patterns in the Presence of the Memory Constraint*, VLDB Journal, 2008, Vol. 17, pp. 1321-1344.

[9] G. Dong and J. Pei, *Sequence Data Mining*, Springer, USA, 2007.

[10] J. Han, J. Wang, Y. Lu and P. Tzvetkov, *Mining Top-K Frequent Closed Patterns without Minimum Support*, ICDM 2012, PP. 211-218.

[11] Y. Liu, W. Liao and A. Choudhary, *A two-phase algorithm for fast discovery of high utility itemsets*, PAKDD 2005, vol. 3518, pp. 689-695.

[12] M. Liu and J. Qu, *Mining high utility itemsets without candidate generation*, CIKM 2012, pp. 55-64

[13] J. Liu, K. Wang and B. Fung, *Direct Discovery of High Utility Itemsets without Candidate Generation*, ICDM 2012, pp. 984-989

[14] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal and M.C. Hsu.,*PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth*, ICDE 2001, pp. 215-224.

[15] T. M. Quang, S. Oyanagi and K. Yamazaki, *ExMiner: An Efficient Algorithm for Mining Top-K Frequent Patterns*, ADMA 2006, pp. 436C447.

[16] L. Shen, H. Shen, P. Prithard and R. Topor,*Finding the N largest itemsets*, ICDM 1998, pp. 211-222

[17] B. Shie, H. Hsiao, V. S. Tseng and P. S. Yu, *Mining high utility mobile sequential patterns in mobile commerce environments*, DASFAA 2011, pp.224-238.

[18] V. S. Tseng, C.-W. Wu, B.-E. Shie and P. S. Yu, *UP-Growth: an efficient algorithm for high utility itemset mining*, KDD 2010, pp. 253-262.

[19] P. Tzvetkov, X. Yan and J. Han, *TSP: Mining Top-K Closed Sequential Patterns*, ICDM 2003, pp. 347-354.

[20] J. Wang and J. Han, *TFP: An Efficient Algorithm for Mining Top-K Frequent Closed Itemsets*, TKDE, 2005, Vol. 17, pp. 652-664.

[21] C. Wu, B. Shie, V. S. Tseng and P. S. Yu, *Mining top-K high utility itemsets*, KDD 2012, pp. 78-86.

[22] J. Yin, Z. Zheng and L. Cao, *USpan: an efficient algorithm for mining high utility sequential patterns*, KDD 2012, pp. 660-668.

[23] M. J. Zaki, *SPADE: An Efficient Algorithm for Mining Frequent Sequences*, Machine Learning, 2001, vol. 42, pp. 31-60.

[24] U. Yun, J.J. Leggett, *WSpan: Weighted Sequential Pattern mining in large sequence databases*, Proc. of the Third Int'l Conf. on IEEE Intelligent Systems, 2006, pp. 512 - 517.

[25] B.-E. Shie, V. S. Tseng and P. S. Yu, *Mining interesting user behavior patterns in mobile commerce environments*, Applied Intelligence, 2013, Volume 38, Issue 3, pp. 418-435.

[26] http://www.informit.com/store/microsoft-sql-server-2008-analysis-services-unleashed-9780672330018

[27] http://www.kdd.org/kdd-cup-2000-online-retailer-website-clickstream-analysis