USpan: An Efficient Algorithm for Mining High Utility Sequential Patterns

Junfu Yin
Advanced Analytics Institute
University of Technology,
Sydney, Australia
junfu.yin@student.uts.edu.au

Zhigang Zheng
Advanced Analytics Institute
University of Technology,
Sydney, Australia
zgzheng@it.uts.edu.au

Longbing Cao*
Advanced Analytics Institute
University of Technology,
Sydney, Australia
longbing.cao@uts.edu.au

ABSTRACT

Sequential pattern mining plays an important role in many applications, such as bioinformatics and consumer behavior analysis. However, the classic frequency-based framework often leads to many patterns being identified, most of which are not informative enough for business decision-making. In frequent pattern mining, a recent effort has been to incorporate *utility* into the pattern selection framework, so that high utility (frequent or infrequent) patterns are mined which address typical business concerns such as dollar value associated with each pattern. In this paper, we incorporate utility into sequential pattern mining, and a generic framework for high utility sequence mining is defined. An efficient algorithm, USpan, is presented to mine for high utility sequential patterns. In USpan, we introduce the lexicographic quantitative sequence tree to extract the complete set of high utility sequences and design concatenation mechanisms for calculating the utility of a node and its children with two effective pruning strategies. Substantial experiments on both synthetic and real datasets show that USpan efficiently identifies high utility sequences from large scale data with very low minimum utility.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data mining

General Terms

Algorithms

Keywords

High utility sequential pattern mining, Sequential pattern mining

*Author for correspondence.

KDD'12, August 12–16, 2012, Beijing, China.

1. INTRODUCTION

Sequential pattern mining has emerged as an important topic in data mining. It has proven to be very essential for handling order-based critical business problems, such as behavior analysis, gene analysis in bioinformatics and weblog mining. For example, sequence analysis is widely employed in DNA and protein to discover interesting structures and functions of molecular or DNA sequences. The selection of interesting sequences is generally based on the frequency/support framework: sequences of high frequency are treated as significant. Under this framework, the *downward closure property* (also known as *Apriori property*) [1] plays a fundamental role for varieties of algorithms designed to search for frequent sequential patterns [10, 14, 5].

In practice, many patterns are identified by frequent sequential pattern mining algorithms. Most of them may not be informative to business decision-making, since they do not show the business value and impact. In some cases, such as fraud detection, some truly interesting sequences may be filtered because of their low frequencies. For example, in retail business, selling a car generally leads to much higher profit than selling a bottle of milk, while the frequency of cars sold is much lower than that of milk. In online banking fraud detection, the transfer of a large amount of money to an unauthorized overseas account may appear once in over one million transactions, yet it has a substantial business impact. Such problems cannot be tackled by the frequency/support framework.

This brings about an interesting question: how to mine sequential patterns of business interest? In the related area, *utility* is introduced into frequent pattern mining to mine for patterns of high utility by considering the quality (such as profit) of itemsets. This has led to *high utility pattern mining* [13], which selects interesting patterns based on minimum utility rather than minimum support. Let us use a toy example to illustrate. Table 1 shows the items and their respective weights or profit (quality) appearing in an online retail store. Table 2 collects several shopping sequences with quantities; each transaction in the sequence consists of one to multiple items, and each item is associated with a quantity showing how many of this item were purchased. For instance, the first sequence $(e,5)[(c,2)(f,1)](b,2)$ shows three itemsets $(e,5)$, $[(c,2)\ (f,1)]$ and $(b,2)$, and the quantity purchased of item, e.g. the quantity of e is 5. Following the high utility pattern mining concept, a possible calculation of utility of an itemset is to consider its total profit.

Table 1: Quality Table

item	a	b	c	d	e	f
weight/quality	2	5	4	3	1	1

Table 2: Quantitative Sequence Database

sid	q-sequence
1	$\langle(e,5)[(c,2)(f,1)](b,2)\rangle$
2	$\langle[(a,2)(e,6)][(a,1)(b,1)(c,2)][(a,2)(d,3)(e,3)]\rangle$
3	$\langle(c,1)[(a,6)(d,3)(e,2)]\rangle$
4	$\langle[(b,2)(e,2)][(a,7)(d,3)][(a,4)(b,1)(e,2)]\rangle$
5	$\langle[(b,2)(e,3)][(a,6)(e,3)][(a,2)(b,1)]\rangle$

Accordingly, the utility of a single item can be defined as its purchased quantity times its profit. The utility of an itemset is the sum of the utilities of all its items. Since each item in a sequence may have multiple utility values, the utility of a sequence may have multiple values. For instance, the utility of $\langle ea\rangle$ in sequence 2 is $\{(6\times1+1\times2), (6\times1+2\times2)\} = \{8, 10\}$. The utility of $\langle ea\rangle$ in the database is $\{\{\}, \{8, 10\}, \{\}, \{16, 10\}, \{15, 7\}\}$. We select the highest utility in each sequence and add them together to represent the maximum utility of the sequence in a given sequence database. The maximum utility of $\langle ea\rangle$ is $10+16+15 = 41$. A sequence is of high utility only if its utility is no less than a user-specified *minimum utility*. Following the high utility pattern mining approach, our goal is to mine for highly profitable sequential purchasing; the identified shopping patterns are more informative for retailers in determining their marketing strategy.

High utility sequential pattern mining is substantially different and much more challenging than high utility itemset mining. If the order between itemsets is considered, e.g. $(e, 5)$, $[(c, 2)(f, 1)]$ and $(b, 2)$ in record $sid = 1$ occurring sequentially, it becomes the problem of mining high utility sequential patterns. This is substantially different and much more challenging than mining frequent sequences and high utility itemsets. First, as with high utility itemset mining, the downward closure property does not hold in utility-based sequence mining. This means that most of the existing algorithms cannot be directly transferred, e.g. from frequent sequential pattern mining to high utility sequential pattern mining. Second, compared to high utility itemset mining, utility-based sequence analysis faces the critical combinational explosion and computational complexity caused by sequencing between sequential elements (itemsets).

So far, only very preliminary work has been proposed to mine for high utility sequential patterns [2, 4, 11]. It is in a very early stage since there is no systematic problem statement available. The proposed algorithms are rather specific and focus on simple situations, and still need substantial effective scanning and pruning strategies to improve performance. Basically, we can see that this is a new and promising area expecting much substantial exploration from problem definition to algorithm development and applications.

In this paper, we formalize the problem of high utility sequential pattern mining, and propose a generic framework and an efficient algorithm, USpan, to identify high utility sequences.

- We build the concept of sequence utility by considering the quality and quantity associated with each item in a sequence, and define the problem of mining high utility sequential patterns;
- A complete lexicographic quantitative sequence tree (LQS-tree) is used to construct utility-based sequences; two concatenation mechanisms I-Concatenation and S-Concatenation generate newly concatenated sequences;
- Two pruning methods, width and depth pruning, substantially reduce the search space in the LQS-tree;
- USpan traverses LQS-tree and outputs all the high utility sequential patterns.

Substantial experiments on both synthetic and real datasets show that the proposed framework and the USpan algorithm can efficiently identify high utility sequences from large scale data with very low minimum utility.

The paper is organized as follows. Section 2 reviews the related work. Section 3 proposes a sequence utility framework and defines the problem of mining high utility sequential patterns. Section 4 details the USpan algorithm. Experimental results and evaluation are presented in Section 5. Section 6 concludes the work.

2. RELATED WORK

2.1 Utility Itemset/Pattern Mining

Utility itemset mining, also generally called utility pattern mining, was first introduced in [13]. Every item in the itemsets is associated with an additional value, called internal utility which is the quantity (i.e. count) of the item. An external utility is attached to an item, showing its quality (e.g. price). With such a utility-based database, high utility itemsets (patterns) are mined, including those satisfying the minimum utility. Mining high utility itemsets is much more challenging than discovering frequent itemsets, because the fundamental downward closure property in frequent itemset mining does not hold in utility itemsets.

Several algorithms are available. UMining was proposed in 2004 for mining high utility patterns, but it cannot extract the complete set of them. A transaction-weighted downward closure property was introduced in [8], in which a two-phase algorithm was proposed with a pruning strategy, which makes it faster and more efficient than UMining. IHUP [3] maintains the high utility patterns in an incremental environment; since it avoids multiple scans of the database, its efficiency is far better than [8]. UP-Growth [12] also uses a tree structure, UP-Tree, to mine high utility patterns. Compared to IHUP, UP-Growth is more efficient, since it further reduces the number of promising patterns which cannot be pruned in IHUP.

The above algorithms can only handle utility itemsets, and do not involve the ordering relationships between items. The addition of ordering information in sequences makes it fundamentally different and much more challenging than mining utility itemsets.

2.2 Utility-based Sequential Pattern Mining

Frequent sequential pattern mining is a very popular topic [1, 9], with quite a few algorithms, such as SPADE [14], Prefixspan [10] and SPAM [5], proposed on the support/frequency framework. Algorithms for mining frequent sequences often result in many patterns being mined; most of them may not

make sense to business, and those with frequencies lower than the given minimum support are filtered. This limits the actionability [6] of discovered frequent patterns.

The integration of utility into sequential pattern mining aims to solve the above problem and has only taken place very recently. In total, we found only three papers in the literature. UMSP [11] was designed for mining high utility mobile sequential patterns. Each itemset in a sequence is associated with a location identifier. With this feature, the utility of a mobile sequential pattern is also a single value. UMSP searches for patterns within a structure called MTS-Tree, which is efficient. However, due to the specific constraint on the sequences, this algorithm can only handle specific sequences with simple structures (single item in each sequence element, and a single utility per item).

In [2], an algorithm is specifically designed for utility web log sequences. The utility of a pattern can have multiple values, and the authors choose the utility with maximal values to represent a pattern's utility with two tree structures, i.e. UWAS-tree and IUWAS-tree. However, sequence elements with multiple items such as $[(c,2)(b,1)]$ cannot be supported, and the scenarios considered are rather simple, which limit the algorithm's applicability for complex sequences.

UI and US [4] extends traditional sequential pattern mining. A pattern utility is calculated in two ways. The utilities of sequences having only distinct occurrences are added together, while the highest occurrences are selected from sequences with multiple occurrences and used to calculate the utilities. However, the problem defintion in [4] is rather specific. No generic framework is proposed which has a clear process to transfer from sequential pattern mining to high utility sequence analysis.

It is obvious that mining high utility sequences is a very open and challenging area. Substantial research topics ranging from problem definition to algorithm development and applications are worthwhile to explore.

3. PROBLEM STATEMENT

3.1 Sequence Utility Framework

Let $\mathcal{I} = \{i_1, i_2, ..., i_n\}$ be a set of distinct *items*. Each item $i_k \in \mathcal{I} (1 \leqslant k \leqslant n)$ is associated with a *quality* (or *external utility*), denoted as $p(i_k)$, which may be the unit profit or price of i_k. A *quantitative item*, or *q-item*, is an ordered pair (i, q), where $i \in \mathcal{I}$ represents an item and q is a positive number representing the *quantity* or *internal utility* of i, e.g. the purchased number of i. A *quantitative itemset*, or *q-itemset*, consists of more than one q-item, which is denoted and defined as $l = [(i_{j_1}, q_1)(i_{j_2}, q_2)...(i_{j_{n'}}, q_{n'})]$, where (i_{j_k}, q_k) is a q-item for $1 \leqslant k \leqslant n'$. For brevity, the brackets are omitted if a q-itemset has only one q-item. Since the items in a set can be listed in any order, without loss of generality, we assume that q-items are listed in alphabetical order. A *quantitative sequence*, or *q-sequence*, is an ordered list of q-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 q-itemset. A *q-sequence database* $\mathcal{S}$ consists of sets of tuples $\langle sid, s \rangle$, where sid is a unique identifier of s, which is a q-sequence.

We use the examples in Table 1 and Table 2 to illustrate the concepts, to show items and corresponding qualities and q-sequences respectively. In $sid = 1$ q-sequence, $(e,5)$, $(c,2)$, $(f,1)$ and $(b,2)$ are q-items; $[(c,2)\ (f,1)]$ is a q-itemset with two q-items. For convenience, in this paper, "sequence" refers to ordered itemsets without quantities, i.e. the same meaning in sequence analysis; similarly, "item" and "itemset" do not involve quantity either. We use "q-" to name the object associated with quantity. We denote the $sid = 1$ q-sequence in Table 2 as s_1; other q-sequences are numbered accordingly. We use the following definitions to construct the sequence utility framework.

Definition 1. (Q-itemset Containing) Given two q-itemsets $l_a = [(i_{a_1}, q_{a_1})(i_{a_2}, q_{a_2})...(i_{a_n}, q_{a_n})]$ and $l_b = [(i_{b_1}, q_{b_1})(i_{b_2}, q_{b_2})...(i_{b_m}, q_{b_m})]$, l_b *contains* l_a iff there exist integers $1 \leq j_1 \leq j_2 \leq ... \leq j_n \leq m$ such that $i_{a_k} = i_{b_{j_k}} \wedge q_{a_k} = q_{b_{j_k}}$ for $1 \leq k \leq n$, denoted as $l_a \subseteq l_b$.

For example, q-itemset $[(a,4)(b,1)(e,2)]$ contains q-itemsets $(a,4)$, $[(a,4)(e,2)]$ and $[(a,4)(b,1)(e,2)]$, but does not contain $[(a,2)(e,2)]$ or $[(a,4)(c,1)]$.

Definition 2. (Q-sequence Containing) Given two q-sequences $s = \langle l_1, l_2, ..., l_n \rangle$ and $s' = \langle l'_1, l'_2, ..., l'_{n'} \rangle$, we say s' *contains* s or s is a *q-subsequence* of s' iff there exist integers $1 \leq j_1 \leq j_2 \leq ... \leq j_n \leq n'$ such that $l_k \subseteq l'_{j_k}$ for $1 \leq k \leq n$, denoted as $s \subseteq s'$.

For example, $\langle (b,2) \rangle$, $\langle [(b,2)(e,3)] \rangle$, $\langle [(b,2)][(e,3)](a,2) \rangle$ are the q-subsequences of q-sequence s_5 ($sid = 5$), while $\langle [(b,4)(e,3)] \rangle$ and $\langle (b,2)(b,6) \rangle$ are not.

Definition 3. (Length and Size) A (q-)sequence is called k-(q)sequence i.e. its length is k iff there are k (q-)items in the (q-)sequence; the size of a (q-)sequence is the number of (q-)itemsets in the (q-)sequence.

For example, $\langle (e,5)[(c,2)(f,1)](b,2) \rangle$ is a 4-q-sequence with size 3. $\langle ea \rangle$ is a 2-sequence with size 2.

Definition 4. (Matching) Given a q-sequence $s = \langle (s_1, q_1)\ (s_2, q_2)...(s_n, q_n) \rangle$ and a sequence $t = \langle t_1 t_2 ... t_m \rangle$. s matches t iff $n = m$ and $s_k = t_k$ for $1 \leq k \leq n$, denoted as $t \sim s$.

Due to the variety of quantities, two q-items can be different even though their items are the same. That is, there could be multiple q-subsequences of a q-sequence matching a given sequence. For example, if we want to find the q-subsequences in q-sequence s_4 ($sid = 4$) in Table 2 which matches the sequence $\langle b \rangle$, we obtain $\langle (b,2) \rangle$ in the first q-itemset and $\langle (b,1) \rangle$ in the third q-itemset. Sometimes, two q-items can be exactly the same and appear in one q-sequence. For example, q-item $(e,2)$ appears in both the first and third q-itemsets in q-sequence s_4.

Definition 5. (Q-item Utility) The *q-item utility* is the utility of a single q-item (i, q), denoted and defined as $u(i, q)$:

$$u(i, q) = f_{u_i}(p(i), q) \tag{1}$$

where f_{u_i} is the function for calculating q-item utility.

Definition 6. (Q-itemset Utility) Q-itemset utility is the utility of an q-itemset $l = [(i_1, q_1)\ (i_2, q_2)\ ...\ (i_n, q_n)]$, denoted and defined as $u(l)$:

$$u(l) = f_{u_{is}}(\bigcup_{j=1}^{n} u(i_j, q_j)) \tag{2}$$

$f_{u_{is}}$ is the function for calculating q-itemset utility.

Definition 7. (Q-sequence Utility) For a q-sequence $s = \langle l_1 l_2 ... l_m \rangle$, the q-sequence utility is $u(s)$:

$$u(s) = f_{u_s}(\bigcup_{j=1}^{m} u(l_j)) \quad (3)$$

where f_{u_s} is the utility function for q-sequences.

Definition 8. (Q-sequence Database Utility) For a utility-oriented sequence database $\mathcal{S} = \{\langle sid_1, s_1 \rangle, \langle sid_2, s_2 \rangle, ..., \langle sid_r, s_r \rangle\}$, the q-sequence database utility is $u(\mathcal{S})$:

$$u(\mathcal{S}) = f_{u_{db}}(\bigcup_{j=1}^{r} u(s_j)) \quad (4)$$

$f_{u_{db}}$ is the function for aggregating utilities in the database.

In the above, utility functions f_{u_i}, $f_{u_{is}}$, f_{u_s} and $f_{u_{db}}$ are all application-dependent, which may be determined through collaboration with domain experts.

Definition 9. (Sequence Utility) Given a utility-oriented database $\mathcal{S}$ and a sequence $t = \langle t_1 t_2 ... t_n \rangle$, t's utility in q-sequence $s = \langle l_1 l_2 ... l_m \rangle$ from $\mathcal{S}$ is denoted and defined as $v(t, s)$, which is a utility set:

$$v(t, s) = \bigcup_{s' \sim t \wedge s' \subseteq s} u(s') \quad (5)$$

The utility of t in $\mathcal{S}$ is denoted and defined as $v(t)$, which is also a utility set:

$$v(t) = \bigcup_{s \in \mathcal{S}} u(t, s) \quad (6)$$

For example, let sequence $t = \langle ea \rangle$, t's utility in the s_4 sequence in Table 2 is $v(t, s_4) = \{u(\langle (e, 2)(a, 7) \rangle), u(\langle (e, 2)(a, 4) \rangle)\} = \{16, 10\}$. t's utility in $\mathcal{S}$ is $v(t) = \{u(t, s_2), u(t, s_4), u(t, s_5)\} = \{\{8, 10\}, \{16, 10\}, \{15, 7\}\}$. This shows that there may be multiple utility values for a sequence within the utility sequence framework. For instance, $t = \langle ea \rangle$ has 2 utility values 16 and 10 for s_4. This is very different from frequent sequential pattern mining, in which there is only one support associated with a sequence.

3.2 High Utility Sequential Pattern Mining

In the utility Definitions 5-8, we did not provide the utility functions f_{u_i}, $f_{u_{is}}$, f_{u_s} and $f_{u_{db}}$. Here, we first specify them, and then state the problem of high utility sequential pattern mining. Although there may be various ways, we here define the above functions as

$$f_{u_i}(p(i), q) = p(i) \times q, \quad (7)$$

$$f_{u_{is}}(\bigcup_{j=1}^{n} u(i_j)) = \sum_{j=1}^{n} u(i_j), \quad (8)$$

$$f_{u_s}(\bigcup_{j=1}^{m} u(t_j)) = \sum_{j=1}^{m} u(l_j), \quad (9)$$

$$f_{u_{db}}(\bigcup_{j=1}^{r} u(s_i)) = \sum_{j=1}^{r} u(s_j) \quad (10)$$

Definition 10. (High Utility Sequential Pattern) Because a sequence may have multiple utility values in the q-sequence context, we choose the *maximum utility* as the sequence's utility. The *maximum utility* of a sequence t is denoted and defined as $u_{max}(t)$:

$$u_{max}(t) = \sum \max\{u(s') | s' \sim t \wedge s' \subseteq s \wedge s \in \mathcal{S}\} \quad (11)$$

Sequence t is a high utility sequential pattern if and only if

$$u_{max}(t) \geq \xi \quad (12)$$

where ξ is a user-specified *minimum utility*. Therefore, given a utility sequence database $\mathcal{S}$ and the minimum utility ξ, the problem of mining high utility sequential patterns is to extract all high utility sequences in $\mathcal{S}$ satisfying ξ.

Here we illustrate the utility definitions in Section 3.1 and the above utility functions through their use in the retail business. In Tables 1 and 2, the utility of a shopped item (q-item) is its profit, equal to the unit profit (weight or quality) of the item times the quantity of the item shopped. The profit (q-itemset utility) of a series of purchased items (q-itemset) is the sum of the profits of all items. Similarly, we can calculate the profit (utility) for a shopping sequence and for a shopping database. For example, for s_1, the utility of q-item $(e, 5)$ is $u(e, 5) = 5 \times 1 = 5$, which is also the utility of the first itemset's utility. Similarly, the utility of s_1 and $\mathcal{S}$ are $u(s_1) = u(e, 5) + u(c, 2) + u(f, 1) + u(b, 2) = 5 \times 1 + 2 \times 4 + 1 \times 1 + 2 \times 5 = 24$ and $u(\mathcal{S}) = u(s_1) + u(s_2) + u(s_3) + u(s_4) + u(s_5) = 24 + 41 + 27 + 50 + 37 = 179$ respectively. The utility of sequence ea is $u_{max}(\langle ea \rangle) = 10 + 16 + 15 = 41$. If the minimum utility is $\xi = 40$, then the shopping sequence $s = \langle ea \rangle$ is a high utility sequential pattern since $u_{max}(s) = 41 \geq \xi$.

The utility Definitions 5-9 and the utility functions defined in Equations (7)-(10) define the problem of utility sequence mining. The high utility sequential pattern mining specification defined in Equations (11) and (12) is a special case of utility sequence mining. Based on different definitions of sequence utility calculation, other metrics can be defined for selecting high utility sequences. In fact, the traditional frequent sequence mining problem can also be viewed as a special case of the above utility-based framework. Suppose we set the quantity of all items as 1, and define the utility functions in Equations (7)-(10) as

$$f_{u_i}(i, q) = p(i) \times q, \quad (13)$$

$$f_{u_{is}}(\bigcup_{j=1}^{n} u(i_j)) = \prod_{j=1}^{n} u(i_j), \quad (14)$$

$$f_{u_s}(\bigcup_{j=1}^{m} u(t_j)) = \prod_{j=1}^{m} u(l_j), \quad (15)$$

$$f_{u_{db}}(\bigcup_{j=1}^{r} u(s_i)) = \sum_{j=1}^{r} u(s_j) \quad (16)$$

then the sequence utility is equal to its support. We can also prove that the specific algorithms proposed in the related work [4, 11, 2] are special cases of our proposed utility sequence mining framework.

4. USPAN ALGORITHM

Here we specify and present an efficient algorithm, USpan, for mining high utility sequential patterns. USpan is composed of a lexicographic q-sequence tree, two concatenation mechanisms, and two pruning strategies.

4.1 Lexicographic Q-sequence Tree

For utility-based sequences, we adapt the concept of the Lexicographic Sequence Tree in [5] to the characteristics of q-sequences, and come up with the Lexicographic Q-sequence Tree (LQS-Tree) to construct and organize utility-based q-sequences.

Suppose we have a k-sequence t, we call the operation of appending a new item to the end of t to form (k+1)-sequence *concatenation*. If the size of t does not change, we call the operation *I-Concatenation*. Otherwise, if the size increases by one, we call it *S-Concatenation*. For example, $\langle ea\rangle$'s I-Concatenate and S-Concatenate with b result in $\langle e(ab)\rangle$ and $\langle eab\rangle$, respectively. Assume two k-sequences t_a and t_b are concatenated from sequence t, then $t_a < t_b$ if

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 the concatenated item in t_a is alphabetically smaller than that of t_b.

For example, $\langle(ab)\rangle < \langle(ab)b\rangle$, $\langle(abc)\rangle < \langle(ab)b\rangle$, $\langle(ab)c\rangle < \langle(ab)d\rangle$ and $\langle(ab)(de)\rangle < \langle(ab)(df)\rangle$.

Definition 11. (Lexicographic Q-sequence Tree) An lexicographic q-sequence tree (LQS-Tree) T is a tree structure satisfying the following rules:

- Each node in T is a sequence along with the utility of the sequence, while the root is empty
- Any node's child is either an I-Concatenated or S-Concatenated sequence node of the node itself
- All the children of any node in T are listed in an incremental and alphabetical order

Additionally, if we set $\xi = 0$, then the complete set of the identified high utility sequential patterns forms a *complete-LQS-Tree*, which covers the complete search space.

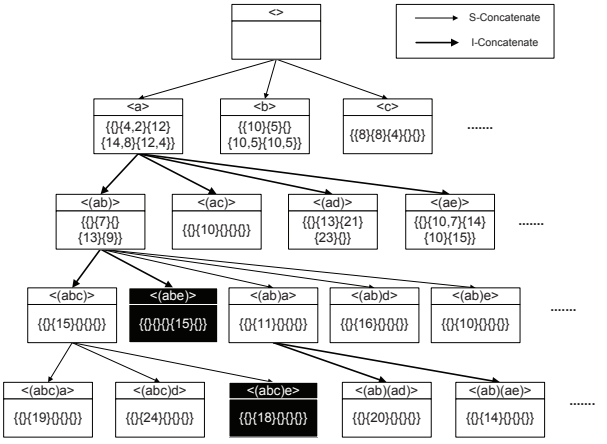


Figure 1: The Complete-LQS-Tree for the Example in Table 2

Figure 1 is an example of LQS-Tree. The root is an empty q-sequence, while the nodes in the black boxes such as $\langle(abe)\rangle$ are leaves in the LQS-Tree. The bold lines and the light lines represent I-Concatenation and S-Concatenation, respectively. Nodes within the same parent are arranged in increasing order. The utilities of the sequences are in the bottom of the respective boxes.

Given a sequence t and a sequence database $\mathcal{S}$, calculating $v(t)$ in $\mathcal{S}$ is easy without any prior knowledge. For example, if we want to calculate $v(\langle ea\rangle)$, we simply find all the q-subsequences in each q-sequence that match $\langle ea\rangle$, and calculate and aggregate the utilities of those q-subsequences. We obtain $v(\langle ea\rangle) = \{\{8, 10\}, \{16, 10\}, \{15, 7\}\}$ and $u_{max}(\langle ea\rangle) = 41$. Once we have $u_{max}(\langle ea\rangle)$, a very natural question is, "Can any $\langle ea\rangle$'s child's maximum utility be calculated by simply adding the highest utility of the q-items after $\langle ea\rangle$ to $u_{max}(\langle ea\rangle)$?". Unfortunately, the answer is no.

In frequent sequential pattern mining, the *downward closure property* serves as the foundation of pattern mining algorithms. However, this property does not hold in the high utility pattern mining problem. In Table 2, $u_{max}(\langle ea\rangle) = 41$, but $u_{max}(\langle e\rangle) = 5 + 6 + 2 + 2 + 3 = 18$, which is lower than its super-pattern. Thus, any frequent sequential pattern mining algorithms built on this property, such as prefixspan [10] and SPADE [14], cannot mine for high utility sequences. What is more, if we check the maximum utilities of a path in the complete-LQS-Tree, we find that the utilities of the sequential patterns $\langle(ae)\rangle$, $\langle(ae)a\rangle$, $\langle(ae)(ab)\rangle$, $\langle(ae)(abc)\rangle$ and $\langle(ae)(abc)a\rangle$ are 49, 33, 41, 25 and 29 respectively. There is no such property as anti-monotonicity in the maximum utilities. Therefore, it is not surprising that given $\xi > 0$, the high utility sequences may not form an complete-LQS-Tree. For example, for $\xi = 60$, the high utility sequential patterns are $\{(be)a(ab)\}$, $\{ba(ab)\}$, $\{(be)aa\}$ and $\{(be)ab\}$. Obviously, these four patterns cannot form an complete-LQS-Tree.

USpan consequently uses a depth-first search strategy to traverse the LQS-Tree to search for high utility patterns. As shown in Figure 1, USpan first generates the children of the root. It then takes $\langle a\rangle$ as the current node, checks whether $\langle a\rangle$ is a high utility pattern, and scans for $\langle a\rangle$'s possible children. If $\langle a\rangle$'s first children, i.e. $\langle(ab)\rangle$, are not taken as the current node, the same operations will apply to $\langle(ab)\rangle$. This procedure will be recursively invoked until there is no other node in the LQS-Tree to visit.

Three important things about USpan need to be addressed. First, knowing the utility of a node, how can we generate the node's children's utilities by concatenating the corresponding items? The answer is provided in Section 4.2. Second, how can we avoid checking unpromising children? We discuss this in Section 4.3. Finally, when should USpan stop the search of deeper nodes? This is discussed in Section 4.4.

4.2 Concatenations

At this point, we discuss how to generate the children's utility based on the utility of its parent, in other words, through I-Concatenation and S-Concatenation. We introduce a *utility matrix* to represent the utility of a q-sequence. Table 3 is the utility matrix of q-sequence s_4 in Table 2. Each element in the matrix is a tuple; the first value shows the utility of the q-item, and the second is the utility of the remaining items in the q-sequence; we call it *remaining utility*, which will be discussed in Section 4.4. The items that do not appear in the q-sequence are given zero utility value. We illustrate the concatenations with q-sequence s_4; other sequences can be conducted in the same way. Let us look at the record for item b in Table 3. Clearly, q-

Table 3: Utility Matrix of Q-sequence s_4 in Table 2

items	q-itemset 1	q-itemset 2	q-itemset 3
a	(0,50)	(14,24)	(8,7)
b	(10,40)	(0,24)	(5,2)
d	(0,40)	(9,15)	(0,2)
e	(2,38)	(0,15)	(2,0)

subsequences $\langle(b,2)\rangle$ and $\langle(b,1)\rangle$ match the sequence $\langle b\rangle$, so $v(\langle b\rangle, s_4) = \{10, 5\}$. Items can either I-Concatenate or S-Concatenate to an existing pattern.

We start from the I-Concatenation. In the example, only items larger than b can be I-Concatenated, i.e. entries in the rectangle from $d1$ (meaning d in itemset 1) to $e3$ are possible items. More precisely, only itemsets 1 and 3 have b, so items corresponding to $e1 = (2, 38)$ to $e3 = (2, 0)$ can be used to form the q-subsequences that match the sequence $\langle(be)\rangle$. The utilities of $\langle(be)\rangle$ are the utilities of $u(\langle b\rangle, s_4)$ plus the newly added q-items' utilities $e1 = (2, 38)$, $e3 = (2, 0)$, i.e. $v(\langle(be)\rangle, s_4) = \{10 + 2, 5 + 2\} = \{12, 7\}$. Similarly, we have $v(\langle a\rangle, s_4) = \{14, 8\}$, and utilities for its I-Concatenated sequences $v(\langle(ab)\rangle, s_4) = \{13\}$, $v(\langle(ad)\rangle, s_4) = \{23\}$, $v(\langle(ae)\rangle, s_4) = \{10\}$, etc.

S-Concatenation is a little more complicated. We continue with $\langle(be)\rangle$. As we can see from the utility matrix, there is no other literal that can be I-Concatenated to $\langle(be)\rangle$. Q-items that can be S-Concatenated to the q-subsequences are located in the rectangle region from $a1$ to $e3$. Thus, sequences such as $\langle(be)a\rangle$, $\langle(be)b\rangle$, $\langle(be)d\rangle$ and $\langle(be)e\rangle$ are the candidates. $\langle[(b,2)(e,2)](a,7)\rangle$ and $\langle[(b,2)(e,2)](a,4)\rangle$ match sequence $\langle(be)a\rangle$, whose utilities are $v(\langle(be)a\rangle, s_4) = \{12 + 14, 12 + 8\} = \{26, 20\}$. We also have $v(\langle(be)b\rangle, s_4) = \{17\}$, $v(\langle(be)d\rangle, s_4) = \{21\}$, $v(\langle(be)e\rangle, s_4) = \{14\}$.

From the above two examples, we conclude that a sequence's children's utilities can be calculated in terms of the utility of a sequence and the positions of the last q-items of q-subsequences that match the sequence. For example, to generate the utility of $\langle(be)a\rangle$ based on $\langle(be)\rangle$ in s_4, we only need to know the following information from $\langle(be)\rangle$: i) $e1$ and $e3$ are the two last q-items of q-subsequences which match the sequence $\langle(be)\rangle$, and ii) the utilities are 12 and 7 respectively. As for which q-items matches b, this is not important. Additionally, we call $e1$ *pivot*, because it is the first place where the q-subsequences that match $\langle(be)\rangle$ end. Items that are similar to $e3$ are called *ending q-items*.

Figure 2 presents the data representation in USpan. Every sequence is stored in the memory in the form of a utility matrix. We omit the entries in the figure for simplicity. The pivot in q-sequence 1 is the black dot; other ending q-items are the black solid boxes on the right side of the dot.

4.3 Width Pruning

The above section discusses how to concatenate items to a sequence, a remaining issue is what kind of items are qualified to be concatenated. This section presents the scanning function of USpan, and proposes a width pruning strategy to further select the promising items.

As shown in Figure 2, those located at the left side of the pivot (inclusive) are called *projected q-items*. Clearly, it is not possible to concatenate these projected q items. The qualified items are at the right side of the pivot. They are I-Concatenation items right under the pivot and the ending q-items, and S-Concatenation q-items are on the right

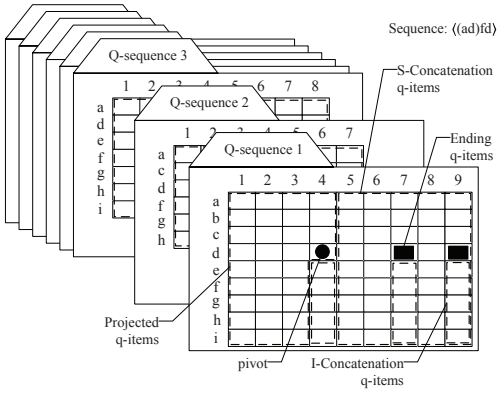


Figure 2: Data Representation in USpan

side of the pivot. For each sequence in $\mathcal{S}$, those items should be scanned and inserted into the corresponding I-Concatenation and/or S-Concatenation lists.

Not every qualified item is a promising item. For example, f is qualified to concatenate to several sequences, but it appears once only in the whole database, i.e. $(f, 1)$ in q-sequence 1. The maximum utility of any sequence containing f will be no more than the utility of q-sequence 1, so any sequence concatenating with f will, if it can, make itself a low utility pattern.

To avoid selecting the unpromising items, we propose a *width pruning strategy* for the scanning subroutine. This is based on the *sequence-weighted downward closure property* (SDCP), which is similar to the *transaction-weighted downward closure property* (TDCP) in [8]. Before introducing SDCP, we give a definition to the *sequence-weighted utilization* (SWU) of a sequence.

Definition 12. (SWU) SWU of a sequence t in $\mathcal{S}$ is denoted and defined as $SWU(t)$

$$SWU(t) = \sum_{s' \sim t \wedge s' \subseteq s \wedge s \subseteq \mathcal{S}} u(s) \tag{17}$$

For example, $SWU(\langle f\rangle) = u(s_1) = 24$ and $SWU(\langle ea\rangle) = u(s_2) + u(s_4) + u(s_5) = 41 + 50 + 37 = 128$.

THEOREM 1. *(Sequence-weighted Downward Closure Property) Given a utility-based sequence database $\mathcal{S}$, and two sequences t_1 and t_2, where t_2 contains t_1, then*

$$SWU(t_2) \leqslant SWU(t_1) \tag{18}$$

PROOF. Let $s_2 \subseteq s_j \in \mathcal{S}$ be a subsequence matching the sequence t_2. Since t_2 contains t_1, we know that there must be a subsequence $s_1 \subseteq s_2$ matching t_1. Therefore, a sequence containing subsequences such as s_2 is a subset of that containing s_1, i.e

$$\bigcup_{s_2 \sim t_2 \wedge s_2 \subseteq s_j \wedge s_j \subseteq \mathcal{S}} s_j \subseteq \bigcup_{s_1 \sim t_1 \wedge s_1 \subseteq s_i \wedge s_i \subseteq \mathcal{S}} s_i \tag{19}$$

We derive,

$$\sum_{s_2 \sim t_2 \wedge s_2 \subseteq s_j \wedge s_j \subseteq \mathcal{S}} u(s) \leqslant \sum_{s_1 \sim t_1 \wedge s_1 \subseteq s_i \wedge s_i \subseteq \mathcal{S}} u(s_i) \tag{20}$$

and obtain $SWU(t_2) \leqslant SWU(t_1)$. □

Based on Theorem 1, we define whether an item is "promising". Imagine we have a k-sequence t, a new item i concatenates to t and results in a (k+1)-sequence t'. If $SWU(t') \geqslant \xi$, we say item i is a *promising item* to t. Otherwise, i is called an *unpromising item*. In the implementation, to test whether an item is promising, we do not have to generate the new sequence to test whether an item is promising. We simply add the utilities of all the sequences; this is equal to the SWU of the new sequence.

4.4 Depth Pruning

The width pruning strategy avoids constructing unpromising patterns into the LP-Tree; a depth pruning strategy stops USpan from going deeper by identifying the leaf nodes in the tree. Imagine the following scenario: the pivots are approaching the end of q-sequences; meanwhile, the maximum utility of the sequence is much less than ξ. The gap is so large that even if all the utilities of the remaining q-items are counted into the utility of the sequence, the cumulative utility still cannot satisfy ξ. In this situation, we use the depth pruning strategy to backtrack USpan instead of waiting to go deeper and returning with nothing.

We use the notation $u_{rest}(i, s)$ to refer to the remaining utility at q-item i (exclusive) in q-sequence s. In the utility matrix, the *remaining utility* appears in the second element in each entry, as shown in Table 3, e.g. $u_{rest}(b1, s_4) = 40$, $u_{rest}(d2, s_4) = 15$.

THEOREM 2. *Given a sequence t and $\mathcal{S}$, the maximum utilities of t and t's offsprings are no more than*

$$\sum_{i \in s' \wedge s' \sim t \wedge s' \subseteq s \wedge s \in \mathcal{S}} (u_{rest}(i, s) + u(s')), \qquad (21)$$

where i is the pivot in s of t , $i \in s'$ and $s' \subseteq s$.

PROOF. Suppose we have the utility of sequence t in $\mathcal{S}$, we can divide each sequence $s \in \mathcal{S}$ into two parts from the pivots, where the pivots are in the left part. Assume $s' \subseteq s$ and pivot $i \in s'$, in other word, s' is the far left subsequence in s that matches t. t's offsprings can be only concatenated from the right side of the pivot. Correspondingly, it is easy to understand that the maximum utilities of the concatenated items are no more than $u_{rest}(i, s)$. Hence, the utility of any item concatenated from s is no more than $u_{rest}(i, s) + u(s')$. Similarly, the highest utility of other sequences in $\mathcal{S}$ can be calculated in the same way. Thus, the theorem holds. □

Based on Theorem 2, if the utility upper bound, i.e. the sum of remaining utilities and utilities of far left subsequences, is less than ξ, we can simply stop USpan from going deeper and backtrack the search procedure.

4.5 USpan Algorithm

The USpan algorithm is illustrated in Algorithm 1. The input for USpan is a database $\mathcal{S}$ and a minimum utility threshold ξ; the output includes all the high utility patterns.

Lines 1 describes the depth pruning strategy. A node is judged as a leaf or not based on the comparison between the value of Equation (21) and ξ; if it is lower than ξ then it returns to its parent nodes. Lines 2 to 4 are the scanning subroutine with the width pruning in Line 5. Once the concatenation items are collected, the unpromising items are omitted from the respective lists. Lines 7 and 12 construct the I-Concatenation and S-Concatenation children respectively. It invokes the concatenation to generate the utilities of sequences; the positions are also maintained. USpan then outputs the high utility sequences if qualified, and recursively invokes itself to go deeper in the LQS-Tree.

Algorithm 1 USpan$(t, v(t))$

Input: A sequence t, t's utility $v(t)$, a utility-based sequence database $\mathcal{S}$, the minimum utility threshold ξ.
Output: All high utility sequential patterns

```
 1: if p is a leaf node then return
 2: scan the projected database S(v(t)) once to:
 3:       a).put I-Concatenation items into ilist, or
 4:       b).put S-Concatenation items into slist
 5: remove unpromising items in ilist and slist
 6: for each item i in ilist do
 7:     (t', v(t')) ← I-Concatenate(p,i)
 8:     if u_max(t') ⩾ ξ then
 9:         output t'
10:     USpan(t', v(t'))
11: for each item i in slist do
12:     (t', v(t')) ← S-Concatenate(p,i)
13:     if u_max(t') ⩾ ξ then
14:         output t'
15:     USpan(t', v(t'))
    return
```

5. EXPERIMENTS

The USpan algorithm was implemented in C++ of Visual Studio 2010. All experiments were conducted on a desktop computer with Intel Core 2 CPU of 2.80GHz, 4GB memory and Windows XP Professional SP3. Both real and synthetic datasets are used to evaluate the efficiency of USpan.

5.1 Data Sets

Four source datasets are used for the experiments. They include two synthetic datasets $DS1, DS2$ generated by the IBM data generator [1].

DS1 is C10T2.5S4I2.5DB10kN1k.

DS2 is C8T2.5S6I2.5DB10kN10k.

The parameters in DS1(DS2) mean that the average number of elements in a sequence is 10(8), the average number of items in an element is 2.5(2.5), the average length of a maximal pattern consists of 4(6) elements and each element is composed of 2.5(2.5) items average. The data set contains 10k(10k) sequences, the number of items is 1000(10k).

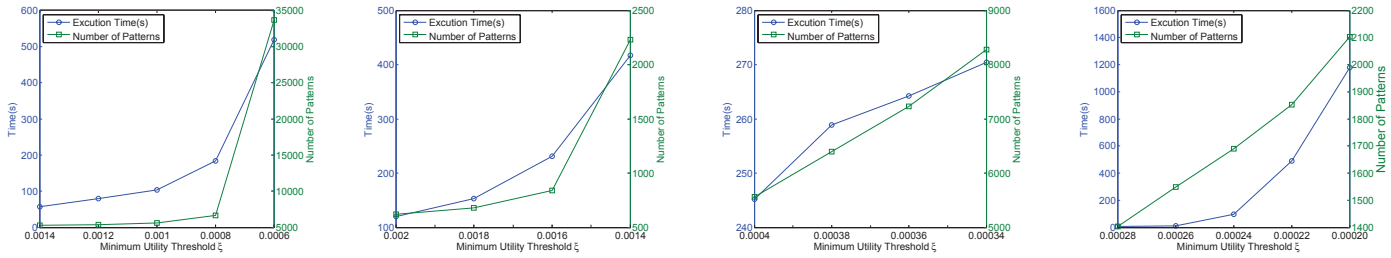
We also test two real datasets $DS3, DS4$.

DS3 is a real dataset consisting of online shopping transactions. Each customer has a sequence of records containing the information about product ID, the amount of the products and its unit price. There are 811 distinct products, 350,241 transactions and 59,477 customers in the dataset. The average number of elements in a sequence is 5. The max length of a customer's sequence is 82. The most popular product has been ordered 2176 times. We test USpan by selecting online shopping sequences with high sale turnover.

DS4 is a real dataset that includes mobile communication transactions. The dataset is a 100,000 mobile-call history from a specific day. There are 67,420 customers in the dataset. The maximum length of a sequence is 152.

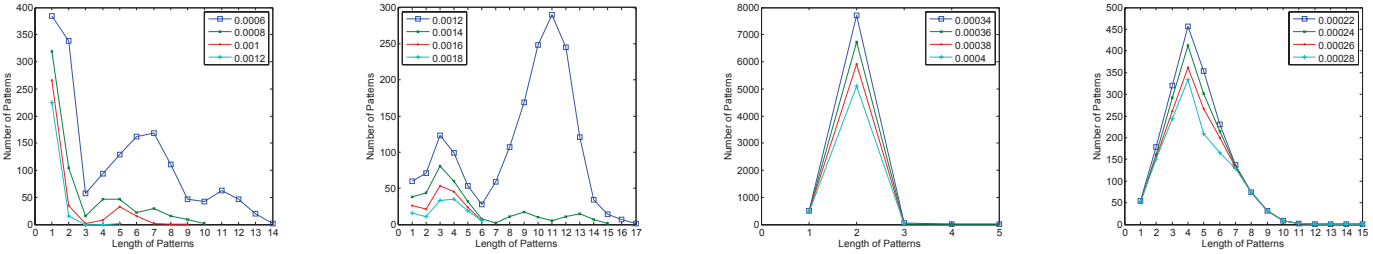
5.2 Performance Evaluation

We conduct intensive experiments to evaluate the performance of USpan in terms of computational cost, memory



(a) DS1 (b) DS2 (c) DS3 (d) DS4

Figure 3: Evaluation of Execution Time and Number of Patterns on the Four Datasets



(a) DS1 (b) DS2 (c) DS3 (d) DS4

Figure 4: Distribution of Discovered High Utility Sequential Patterns

usage, number of high utility patterns, and length of patterns on different datasets.

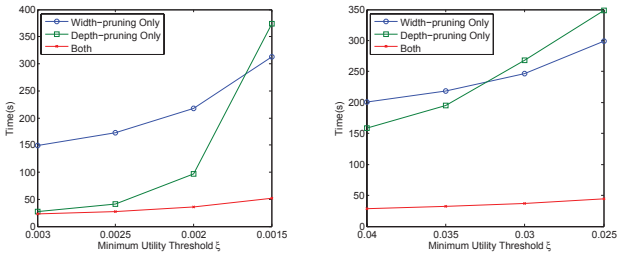
The execution times of mining high utility sequential patterns by USpan on $DS1$ to $DS4$ are shown in Figure 3; the figure also includes the number of patterns. When the minimum utility threshold decreases, more execution time is required since we can obtain many more high utility sequential patterns. The results also show that USpan can extract high utility sequences under very low minimum utility (for instance, 0.0006 for $DS1$ and 0.0002 for $DS2$).

Figure 4 shows the distribution of the high utility sequential patterns discovered in terms of pattern length. It shows that the maximum length of identified high utility sequences increases dramatically with the decrease of minimum utility. It is also clear that high utility sequential pattern mining shows a very different trend of pattern distribution against minimum utility from that of frequent sequential pattern mining against minimum support, e.g. when the pattern length is 11, we find the largest number of identified utility sequences in $DS2$. This shows that the Apriori property does not hold in utility sequence mining.

5.3 Evaluation of Pruning Strategies

We test the computational costs of the two proposed pruning methods on $DS1$ and $DS2$. Three type of pruning strategies are evaluated. The first only uses depth-pruning, the second only uses width-pruning, and the third uses both depth and width pruning. The results of these three strategies are shown in Figure 5.

On both datasets, the depth-pruning method is very sensitive to the minimum utility. When the threshold is high, the pruning is very effective, because it only goes deeper when there is a higher remaining utility value. It can greatly ignore invalid searches by pruning patterns whose pivots appear at the end of sequences. However, when the threshold decreases, the search space in LQS-Tree grows exponentially.



(a) DS1 (b) DS2

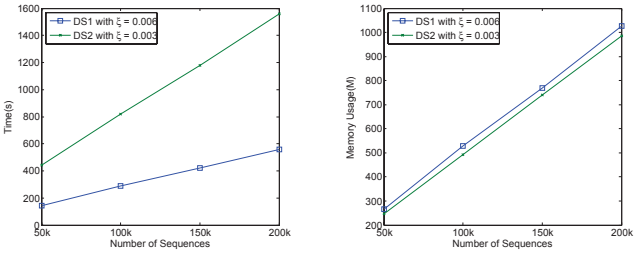
Figure 5: Comparison of Pruning Methods

In contrast, width pruning is more stable with the decrease of the threshold. The reason is that width pruning always prevents unpromising items from getting into the concatenation lists. It can control the width of the trees very well, however it cannot control whether the current sequence is promising until it reaches the very end of the LQS-Tree. The combination of both width and depth pruning strategies leads to extremely improved efficiency compared to either of them, and result in up to eight times the difference in execution time, because the two kinds of pruning strategies can compensate for the shortcomings of each other. In addition, since the high utility sequential pattern mining algorithm in [4] is essentially based on width-pruning, the experimental results indirectly show that USpan is much more efficient.

5.4 Scalability Test

The scalability test is conducted to test USpan's performance on large-scale datasets. Figure 6 shows the results on datasets $DS1$ and $DS2$ in terms of different data sizes: 50K to 200K sequences are extracted from $DS1$ and $DS2$, by setting $\xi = 0.006$ on $DS1$ and $\xi = 0.003$ on $DS2$.

On both datasets, the execution time and memory usage



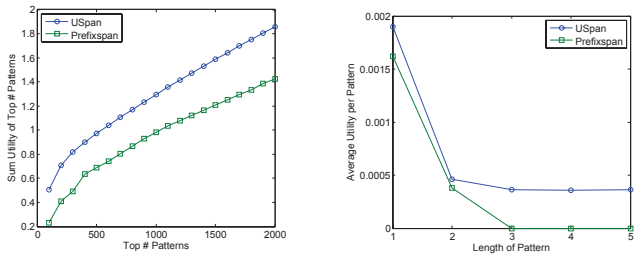
(a) Execution Time (b) Memory Usage

Figure 6: Results of Scalability Test

are exactly linear with the number of transactions, as shown in Figure 6. USpan stores the whole dataset, and the running time is directly related to the size of the LQS-Tree.

5.5 Utility Comparison with Frequent Pattern Mining

This experiment tests the utility difference between the patterns identified by USpan and that the patterns identified purely by frequent sequential pattern mining.



(a) Sum (b) Average

Figure 7: High Utility vs. Frequent Sequential Patterns

Figure 7(a) shows the utilities of two groups of identified patterns, one by prefixspan and the other by USpan. The x axis refers to the top n number of frequent vs. high-utility patterns selected from the two groups, while the y axis shows the sum of the utilities of the top n patterns. Figure 7(b) shows the average utilities of patterns with different lengths from prefixspan and USpan. The x axis refers to the lengths of patterns; while the y axis shows the average utilities per pattern. The results show that USpan can identify higher utility patterns more efficiently, and it can extract top patterns with higher average utility per pattern.

6. DISCUSSION AND CONCLUSIONS

Frequent sequential pattern mining leads to patterns which do not show business value and impact, and thus are not actionable for business decision-making. In this paper, we have provided a systematic statement of a generic framework, and an efficient algorithm, USpan, for mining high utility sequential patterns. Substantial experiments on both synthetic and real datasets have shown that USpan can efficiently identify high utility sequences in large-scale data with low minimum utility. For USpan, the metrics in Equation (11) can be changed to some other proper metrics such as minimum or average utility. The corresponding functions in lines 8 and 13 will be modified as well to fit the new frameworks, and the complexities are in the same level. In fact, USpan stores the positions and utilities of the candidates, a range of different functions can be applied on them with different purposes in such a framework. Our future work is on designing algorithms for even bigger datasets and better pruning strategies.

7. ACKNOWLEDGEMENTS

This work is sponsored in part by Australian Research Council Discovery Grant (DP1096218) and ARC Linkage Grant (LP100200774).

8. 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, J. Byeong-Soo and L. Young-Koo, *Efficient tree structures for high utility pattern mining in incremental databases*, TKDE 2009, vol. 21, pp. 1708-1721.

[4] C. F. Ahmed, S. K. Tanbeer and B. Jeong, *A Novel Approach for Mining High-Utility Sequential Patterns in Sequence Databases*, ETRI Journal, 2010, vol.32, no.5, pp.676-686.

[5] J. Ayres, J. Flannick, J. Gehrke and T. Yiu, *Sequential PAttern mining using a bitmap representation*, ICDM 2002, pp. 429-435.

[6] L. Cao, P. Yu, C. Zhang and Y Zhao. *Domain Driven Data Mining*. Springer, 2010.

[7] Y. Li, J. Yeh and C. Chang, *Isolated items discarding strategy for discovering high utility itemsets*, Data and Knowledge Engineering, Vol. 64, Issue 1, pp.198-217, Jan., 2008.

[8] 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.

[9] N. R. Mabroukeh and C. I. Ezeife, *A taxonomy of sequential pattern mining algorithms*, ACM Comput. Surv., 2010, vol. 43, pp. 1-41.

[10] 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.

[11] 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.

[12] 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.

[13] H. Yao, H. J. Hamilton and C. J. Butz, *A foundational approach to mining itemset utilities from databases*, ICDM 2004, pp. 482-486.

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