A Novel Approach for Mining High-Utility Sequential Patterns in Sequence Databases

Chowdhury Farhan Ahmed, Syed Khairuzzaman Tanbeer, and Byeong-Soo Jeong

Mining sequential patterns is an important research issue in data mining and knowledge discovery with broad applications. However, the existing sequential pattern mining approaches consider only binary frequency values of items in sequences and equal importance/significance values of distinct items. Therefore, they are not applicable to actually represent many real-world scenarios. In this paper, we propose a novel framework for mining high-utility sequential patterns for more real-life applicable information extraction from sequence databases with non-binary frequency values of items in sequences and different importance/significance values for distinct items. Moreover, for mining high-utility sequential patterns, we propose two new algorithms: UtilityLevel is a high-utility sequential pattern mining with a level-wise candidate generation approach, and UtilitySpan is a high-utility sequential pattern mining with a pattern growth approach. Extensive performance analyses show that our algorithms are very efficient and scalable for mining high-utility sequential patterns.

Keywords: Data mining, sequential patterns, high-utility patterns, knowledge discovery.

Manuscript received Mar. 10, 2010; revised July 1, 2010; accepted Aug. 2, 2010.

Chowdhury Farhan Ahmed (phone: +82 31 201 2951, email: farhan@khu.ac.kr), Syed Khairuzzaman Tanbeer (email: tanbeer@khu.ac.kr), and Byeong-Soo Jeong (corresponding author, email: jeong@khu.ac.kr) are with the Database Lab, Department of Computer Engineering, Kyung Hee University, Youngin, Rep. of Korea.

doi:10.4218/etrij.10.1510.0066

I. Introduction

Sequential pattern mining [1]-[6] discovers frequent sequences from a sequence database (SDB). By maintaining the order of elements in a sequence, it can discover crucial knowledge from SDBs. For example, after buying a TV, user X has bought a DVD player within one week. After traversing web page W_1, user Y has traversed web page W_2. Therefore, sequential pattern mining becomes important in many real-life application domains such as market basket data analysis, web usage mining, biomedical gene data analysis for detecting a disease and producing a drug, telecommunication data analysis, stock market, and weather trend prediction.

Even though sequential pattern mining plays an important role in data mining applications, the existing sequential pattern mining algorithms [1]-[7] consider only binary frequency values of items in sequences and equal importance/significance values of distinct items. Moreover, they use support measures to detect whether a sequence is frequent or not. The support/frequency of a sequence is the number of transaction sequences (TSs) containing the sequence in the SDB. The problem of sequential pattern mining is to find the complete set of sequences satisfying a user-given minimum support threshold in the SDB. However, this assumption cannot truly represent many real-life scenarios. For example, in a retail market, each item has a different price/profit value, and a user may buy multiple copies of a same item. In a web traversal sequence, a user may browse different time units in different web pages, and each web page may have a different importance/significance. This gives the motivation to design a high-utility sequential pattern mining framework for SDBs.

The existing high-utility pattern (HUP) mining model [8]-[13] is designed for non-sequential databases, that is, ordinary transaction databases, where the order of elements in a pattern is not maintained. The existing model considers non-binary frequency values of items in transactions and different profit values of items in contrast to the assumption of binary frequency values of items in transactions and equal importance/significance values of items in the traditional frequent pattern mining model [14]-[16]. It uses a measure called “utility” to overcome the limitations of the support measure. This utility measure calculates the actual profit value of a pattern in a transaction database. The problem of HUP mining refers to find out those patterns having utility value greater than or equal to a user-given minimum threshold with respect to the transaction database. By using this measure, very important and useful patterns can be discovered which may not be possible with the support measure. For example, the support measure may not be able to detect a pattern such as *gold ring*, *gold necklace* because this pattern may have very small support value. On the other hand, a utility measure can easily discover this pattern because it considers the profit value as well as the non-binary frequency value. By utility mining, several important business area decisions like maximizing revenue, minimizing marketing, and/or inventory costs can be considered, and knowledge about itemsets/customers contributing to the majority of the profit can be discovered. In addition to the real-world retail market, other application areas, such as stock tickers, network traffic measurements, web-server logs, data feeds from sensor networks, and telecommunications call records can have similar solutions.

Motivated by the above real-life scenarios, in this paper, we propose a novel framework for mining high-utility sequential patterns. Our framework considers both internal and external utilities of a sequence and introduces a new measure, sequence utility (SeqUtility), to calculate the utility value of a sequence. It also defines the high-utility sequential patterns and the problem of mining high-utility sequential patterns. Moreover, we propose two new algorithms for mining high-utility sequential patterns: UtilityLevel (UL) is a high-utility sequential pattern mining with a level-wise candidate generation approach, and UtilitySpan (US) is a high-utility sequential pattern mining with a pattern growth approach. The first algorithm, UL, is simple and straightforward compared to the second algorithm. However, it adopts a level-wise candidate generation-and-test mechanism [1], [2]. Hence, it generates a large number of candidate sequences and needs several database scans. On the other hand, the second algorithm, US, exploits a sequential pattern growth mining approach [5], [6] and successfully avoids the problems of the UL algorithm. It needs a maximum of three database scans. Accordingly, it significantly reduces the number of candidate sequences as well as the overall runtime for mining high-utility sequential patterns. Extensive performance analyses show that our algorithms are very efficient for mining high-utility sequential patterns.

The remainder of this paper is organized as follows. In section II, we describe related work. In section III, we propose our framework. In section IV, we develop our proposed algorithms for mining high-utility sequential patterns. In section V, our experimental results are presented and analyzed. Finally, in section VI, conclusions are drawn.

II. Related Work

1. HUP Mining

The theoretical model and definitions of HUP mining were given in [8]. This approach is called mining with expected utility. Later, the same authors proposed two new algorithms, UMining and UMining_H, to calculate HUPs [9]. However, these methods do not satisfy the “downward closure” property of Apriori [14] and overestimate too many patterns. This property says that if a pattern is infrequent, then all of its super-patterns must be infrequent. The Two-Phase [10] algorithm was developed based on the definitions of [8] for HUP mining using the downward closure property with a measure called “transaction-weighted utilization.” The isolated items discarding strategy (IIDS) [12] for discovering HUPs was proposed to reduce some candidates in every pass of databases. Applying IIDS, the authors developed two efficient HUP mining algorithms: FUM and DCG+. However, these algorithms suffer from the problem of level-wise candidate generation-and-test methodology and need several database scans. An efficient candidate pruning technique, HUC-Prune [11], has been proposed to avoid the level-wise candidate generation-and-test problem in HUP mining. In [13], efficient tree structures have been proposed for incremental HUP mining. However, these approaches are not applicable for mining high-utility sequential patterns.

2. Sequential Pattern Mining

The sequential pattern mining problem was first introduced by Agrawal and Srikant in [1]. They have designed Apriori-based algorithms to mine all the sequential patterns according to a user-given minimum threshold. Later, an improved algorithm, generalized sequential pattern [2], was proposed for sequential pattern mining. Zaki [3] devised an algorithm which is a sequential pattern discovery using equivalent classes (SPADE). SPADE was developed for sequential pattern mining using vertical data format. These algorithms are based on the level-wise candidate generation-and-testing mechanism and generate too many candidate patterns. The SPAM algorithm [4] uses a depth-first search strategy using an efficient vertical bitmap representation. An efficient algorithm, PrefixSpan [5], [6], was proposed by using a sequential pattern growth mining approach. The BIDE algorithm [7] can efficiently discover closed sequential patterns without candidate maintenance.

Research has been done for sequential pattern mining with constraints. The SPIRIT algorithm [17] has been developed for mining sequential patterns with user-specified regular expression constraints. Pei and others [18] developed a new framework, Prefix-growth, for different types of constraint-based sequential pattern mining including item, length, super-pattern, aggregate, regular expression, duration, and gap constraints. Some algorithms have also been developed to handle weight and quantity constraints in sequential pattern mining. The WSpan [19] and WIS [20] algorithms use different weight values for different items, but cannot consider non-binary frequency values of items. Kim and others [21] devised sequential pattern mining with quantities (SQUIRE). SQUIRE can consider non-binary frequency values of items. Kim and others proposed two algorithms: Apriori-QSP and PrefixSpan-QSP. These were created by extending the existing Apriori-style and PrefixSpan algorithms of traditional sequential pattern mining, respectively. They have also proposed two improved versions of these algorithms, Apriori-All and PrefixSpan-All [21]. However, the SQUIRE approach cannot consider items with different weights/profits. Moreover, it is based on the support measure. For example, if a sequence is $\{a{:}\ 3, b{:}\ 2\}$, then it just finds its support in the SDB, that is, in how many TSs it is present. As a consequence, none of the existing approaches are capable of mining high-utility sequences. Therefore, in this paper, we propose a novel framework and two new algorithms for mining high-utility sequential patterns in SDBs.

III. Proposed Framework

Let $I=\{i_1, i_2,\dots, i_n\}$ be a set of items and $P=\{p_1, p_2,\dots, p_l\}$ be a pattern (itemset), where $P\subseteq I$ and $l\in[1, n]$. A sequence S, denoted by $\{s_1, s_2,\dots, s_r\}$, is an ordered list of patterns, that is, each s_q $(1\le q\le r)$ is a pattern P, and each pattern appearing in a sequence is called an element of the sequence. The length of a sequence is the number of instances of items inside it. Consider there are two sequences, for example, $\alpha=\{a_1 a_2\dots a_x\}$ and $\beta=\{b_1 b_2\dots b_y\}$ $(x\le y)$. If there exists $j_1<j_2<\dots<j_x\le y$, such that $a_1\subseteq b_{j_1}$, $a_2\subseteq b_{j_2}$,…, and $a_x\subseteq b_{j_x}$, then α is a subsequence of β, and β is a super-sequence of α. An SDB contains a number of TSs: $\{TS_1, TS_2,\dots, TS_m\}$. TS_k $(1\le k\le m)$ contains a

Sequence ID	Sequence with internal utility	Sequence utility ($)
S_1	$a(3)\ \{a(2)\ b(6)\ d(2)\}\ f(1)\ a(5)\ d(1)$	130
S_2	$e(3)\ \{a(2)\ b(5)\}\ d(1)\ c(4)$	85
S_3	$\{c(1)\ f(2)\}\ b(3)\ \{d(1)\ e(4)\}$	74
S_4	$a(2)\ \{b(7)\ d(4)\}\ \{a(6)\ b(3)\}e(5)$	180
S_5	$\{d(1)\ f(3)\}\ c(5)\ g(2)$	67
S_6	$d(2)\ e(1)\ \{a(7)\ b(8)\}\ d(3)\ b(6)\ e(3)$	207

(a)

Item	Profit ($) per unit
a	5
b	7
c	3
d	10
e	6
f	8
g	9

(b)

Fig. 1. Examples of (a) sequence database with internal utility and (b) external utility table.

tuple $<SID_k, S_k>$, where SID_k is the sequence ID, and S_k is the sequence of the TS_k. TS_k is said to contain a sequence, X, if X is a subsequence of S_k.

Definition 1. The internal utility value of an item, i_j, in TS_k is represented by $iu(i_j, S_k)$. External utility $eu(i_j)$ is the impact/significance value of item i_j. Figure 1 shows an example SDB with internal and external utility values. Here, the internal utility values represent the quantities of items in sequences, and the external utility value of each item represents profit ($) per unit of that item. For example, in Fig. 1, $iu(b, S_1)=6$, and $eu(b)=7$. However, an item may appear multiple times in a TS. In that case, $iu(i_j, S_k)$ is the addition of all the quantities of i_j in sequence S_k. For example, in Fig. 1, $iu(a, S_1)=10$.

Definition 2. Sequence utility, $su(i_j, S_k)$, is the quantitative measure of utility for item i_j in TS_k, defined by

$$su(i_j, S_k)=iu(i_j, S_k)\times eu(i_j). \qquad (1)$$

For example, $su(b, S_1)=6\times 7=42$ in Fig. 1.

Definition 3. A sequence, for example, $X=\{x_1, x_2, \dots, x_m\}$, is called an m-sequence, where $X\subseteq S_k$, $x_p\subseteq I$, and $1\le p\le m$. To calculate the internal utility of an item, i_j, in a sequence X $(X\subseteq S_k)$, we have to take only the internal utility of i_j in X. For example, $iu(d, de(ab), S_6)=2$ (where $X=de(ab)$). Hence, as with an item, a sequence X may have multiple distinct occurrences in TS_k. Accordingly, for sequence utility of X in S_k, $su(X, S_k)$ is defined by

$$su(X, S_k)=\sum_{\forall X\in Sk}\ \sum_{i j\in X} su(i_j, X, S_k)\,. \qquad (2)$$

However, in the above equation, we refer to only all distinct occurrences of X. For example, sequence de has two distinct occurrences in S_6. Hence, $su(de, S_6)=(2\times 10+1\times 6)+(3\times 10+3\times 6)=26+48=74$ in Fig. 1. If multiple distinct occurrences cannot be formed, then a sequence is formed by taking the maximum su of a sequence. For example, in the S_6 sequence, eb can be formed by taking the fourth b or the sixth b. Here, the fourth item b is taken as it gives the maximum su of eb in S_6.

Definition 4. The sequence utility of TS_k is defined by

$$su(TS_k) = \sum_{i_j \in S_k} su(i_j, S_k). \quad (3)$$

For example, $su(TS_1)=su(a, S_1)+su(b, S_1)+su(d, S_1)+su(f, S_1)$ $=50+42+30+8=130$.

Definition 5. The sequence utility of a sequence X in an SDB is defined by

$$su(X, SDB) = \sum_{TS_k \in SDB} \sum_{X \subseteq S_k} su(X, S_k). \quad (4)$$

For example, $su(a(bd)a, SDB)=su(a(bd)a, TS_1)+su(a(bd)a, TS_4)=102+129=231$ in Fig. 1

Definition 6. The sequence utility value of an SDB is defined by

$$su(SDB) = \sum_{TS_k \in SDB} su(TS_k). \quad (5)$$

For example, $su(SDB)=743$ in Fig. 1.

Definition 7. The minimum sequence utility threshold, δ, is given by the percentage of sequence utility value of the database. In Fig. 1, if δ is 30% or can be expressed as 0.3, then the minimum sequence utility value can be defined as

$$minSeqUtil=\delta \times su(SDB). \quad (6)$$

Hence, in this example, $minSeqUtil=0.3\times 743=223$ in Fig. 1.

Definition 8. A sequence X is a high-utility sequential pattern if $su(X) \geq minSeqUtil$. Mining high-utility sequential pattern means discovering all the sequences X having criteria $su(X) \geq minSeqUtil$. For $minSeqUtil=223$, $a(bd)a$ is a high-utility sequential patten as $su(a(bd)a)=231$.

The most challenging problem for high-utility sequential pattern mining is that sequence utilities do not have the downward closure property. Consider $minSeqUtil=223$ in Fig. 1: a is a low-utility sequential pattern as $su(a)=135$, but its super-sequence $a(bd)a$ is a high-utility sequential pattern as $su(a(bd)a)=231$. Therefore, the downward closure property is not satisfied. To maintain the downward closure property in high-utility sequential pattern mining, we define a new measure called *sequence-weighted utility* (*swu*).

Definition 9. The *swu* value of a sequence X is defined by

$$swu(X) = \sum_{X \subseteq S_k \wedge TS_k \in SDB} su(TS_k). \quad (7)$$

Definition 10. X is a high-*swu* sequence if $swu(X) \geq minSeqUtil$.

As it is the maximum possible sequence utility value of a sequence, the downward closure property can be maintained by using this value. For example, $swu(g)=su(TS_5)=67$ in Fig. 1. Here, for $minSeqUtil=223$ in Fig. 1, as $swu(g) < minSeqUtil$, any super-sequence of g cannot be a high-*swu* sequence and obviously cannot be a high-utility sequential pattern. In our approach, after finding all the high-*swu* sequence maintaining the downward closure property, we calculate all the high-utility sequential patterns form high-*swu* sequences.

Lemma 1. The *swu* value of a sequence X maintains the downward closure property.

Proof. Let X be a high-*swu* sequence and SDB_X is the set of sequences containing X. Let Y be a superset of X, then Y cannot be present in any sequence where X is absent. Therefore, according to definition 9, the maximum *swu* value of Y is $swu(X)$. Accordingly, if $swu(X)$ is less than $minSeqUtil$, Y cannot be a high-*swu* sequence. □

Lemma 2. For an SDB and minimum sequence utility threshold, the set of high-utility sequential patterns (S) is a subset of the set of high-*swu* sequences (WS).

Proof. Let X be a high-utility sequential pattern. According to definitions 5 and 9, $su(X)$ must be less than or equal to $swu(X)$. So, if X is a high-utility sequential pattern, it must be a high-*swu* sequence. Hence, X is a member of the set WS and $S \subseteq WS$. □

IV. Mining High-Utility Sequential Patterns

1. UL Algorithm

Our first proposed algorithm, UL, is simple and straightforward. It is used for generating high-utility sequential patterns. It adopts a level-wise candidate generation approach for mining high-utility sequential patterns. At first, it generates the candidates for high-*swu* sequences, and subsequently generates high-*swu* sequences (actual candidates for high-utility sequential patterns). Finally, it detects the high-utility sequential patterns from high-*swu* sequences.

The level-by-level candidate generation-and-testing mechanism is shown in Table 1 for the SDB presented in Fig. 1 and $minSeqUtil=30\%$. In level-1, all the distinct items are candidate length-1 high-*swu* sequences. The UL algorithm scans the SDB once to generate all the length-1 high-*swu* sequences. In level-2, 51 candidate high-*swu* sequences are generated as shown in Table 1. For one length-1 high-*swu* sequence, UL generates length-2 candidate sequences by joining it with other length-1 high-*swu* sequences which include it. For example, for length-1 candidate sequence a, length-2 candidate sequences aa, ab, ac, ad, ae, and af are generated. As there are 6 length-1 candidate sequences, a total of $6\times 6=36$ length-2 candidate sequences are generated, that is, for N distinct items, N^2 length-2 sequences containing two elements will be generated. On the other hand, a length-1 candidate sequence joins with others to form length-2 single element candidate sequences, such as when sequence a joins b, c, d, e, and f to form (ab), (ac), (ad), (ae), and (af). Similarly, sequence b joins with c, d, e, and f to form (bc), (bd), (be), and (bf). In this way, 5 candidate sequences for a, 4 candidate

Table 1. Candidate generation process for UL algorithm.

Level	Candidate high-*swu* sequences	Candidate high-utility seq. pattern (high-*swu* sequence) with *swu* values
1	**7 candidates** *a*, *b*, *c*, *d*, *e*, *f*, *g*	**6 high-*swu* sequences** *a*: 602, *b*: 676, *c*: 226, *d*: 743, *e*: 546, *f*: 271
2	**51 candidates (14 candidates not appear in SDB at all)** *aa*, *ab*, *ac*, *ad*, *ae*, *af* *ba*, *bb*, *bc*, *bd*, *be*, *bf* …………………… (*ab*), (*ac*), (*ad*), (*ae*), (*af*) (*bc*), (*bd*), (*be*), (*bf*) ……………………	**17 high-*swu* sequences** *aa*: 310, *ab*: 517, *ad*: 602, *ae*: 387, *ba*: 310, *bb*: 387, *bd*: 496, *be*: 461, *da*: 517, *db*: 387, *dd*: 337, *de*: 387, *ea*: 292, *eb*: 292, *ed*: 292, (*ab*): 602, (*bd*): 310
3	**70 candidates (25 candidates not appear in SDB at all)** *aaa*, *aab*, *aad*, *aae*, *a*(*ab*), *aba*, *abb* …………………… (*ab*)*a*, (*ab*)*b*, (*ab*)*d*, (*ab*)*e* ……………………	**18 high-*swu* sequences** *aba*: 310, *abe*: 387, *ada*: 310, *adb*: 387, *ade*: 387, *a*(*ab*): 310, *a*(*bd*): 310, *bbe*: 387, *dad*: 337, *dae*: 387, *dbe*: 387, *d*(*ab*): 387, *ead*: 292, *ebd*: 292, *e*(*ab*): 292, (*ab*)*d*: 422, (*ab*)*e*: 387, (*bd*)*a*: 310
4	**4 candidates** *adbe*, *a*(*bd*)*a*, *d*(*ab*)*e*, *e*(*ab*)*d*	**4 high-*swu* sequences** *adbe*: 387, *a*(*bd*)*a*: 310, *d*(*ab*)*e*: 387, *e*(*ab*)*d*: 292

sequences for *b*, 3 candidate sequences for *c*, 2 candidate sequences for *d*, and 1 candidate sequence for *e* are generated. Accordingly, a total of 5+4+3+2+1=15 ((6×5)/2)) candidate sequences are generated, that is, for *N* distinct items, (*N*×(*N*–1))/2 length-2 single-element candidate sequences will be generated. Hence, a total of 36+15=51 length-2 candidate sequences are generated. The original SDB must be scanned once again with all these candidate sequences to detect the high-*swu* sequences. It is noticeable that among the 51 candidates, 14 candidates, for example, *ca*, *cc*, *ef*, and (*ce*), do not appear in the SDB at all. However, Table 1 also shows that 17 high-*swu* sequences (candidate high-utility sequential patterns) are generated from these 51 candidates.

From level-3 to the last level, the UL algorithm generates high-*swu* candidate sequences for *k*-th level by joining L_{k-1} (high-*swu* sequences for (*k*–1)th level) with L_{k-1}. A sequence *X* joins with another sequence *Y* if the subsequence obtained by dropping the first item of *X* is the same as the subsequence obtained by dropping the last item of *Y*. The new candidate sequence is formed by joining sequence *X* with the last item of *Y*. The added item becomes a separate element if it was a separate element of *Y*, and otherwise, part of the last element of *X*. For example, length-2 high-*swu* sequences *ab* and *bd* form length-3 candidate high-*swu* sequence *abd*. On the other hand, length-2 high-*swu* sequences *ab* and (*bd*) form length-3 candidate high-*swu* sequence *a*(*bd*). After generating all the candidate high-*swu* sequences by joining, UL algorithm prunes those candidate sequences having at least one subsequence which is not a length-(*k*–1) high-*swu* sequence. For example, after generating length-3 candidate high-*swu* sequence *eae* from length-2 high-*swu* sequences *ea* and *ae*, it prunes *eae* as its subsequence *ee* is not a high-*swu* sequence. However, for level-3 of this example, this algorithm generates a total of 70 candidate sequences after joining and pruning and scans SDB again to detect 18 high-*swu* sequences as shown in Table 1. Similarly, it generates length-4 candidate high-*swu* sequences and calculates 4 high-*swu* sequences. No candidate high-*swu* sequence is generated for level-5. Finally, it scans the SDB with these high-utility-*swu* sequences to discover the high-utility sequential patterns. As a consequence, the UL algorithm discovers a total of 6 high-utility sequential patterns <*b*: 266, (*ab*): 239, (*ab*)*d*: 238, *adbe*: 226, *a*(*bd*)*a*: 231, *d*(*ab*)*e*: 250> by generating a total of 132 candidates and with five database scans.

2. US Algorithm

Even though our first proposed algorithm UL can discover the final resultant high-utility sequential patterns successfully, it suffers from the level-wise candidate generation-and-testing problem and hence generates a large number candidate sequences. Moreover, its number of database scans is directly dependent on the maximum length of candidate sequences. Consider the example presented in the last section. It needs a total of 5 database scans as the maximum candidate length is 4 (last scan is needed for detecting high-utility sequential patterns from the high-*swu* sequences). Hence, it needs a total of *N*+1 database scans for the maximum candidate length of *N*. Accordingly, it needs several database scans to find the resultant sequences.

To solve the problems of our first approach, in this subsection, we propose a second algorithm US. It exploits a

Table 2. Candidate generation process for US algorithm.

Prefix	Projected SDB	Candidate high-utility seq. pattern (high-*swu* sequence) with *swu* values
a	(*abd*)*fad*: 130, (_*b*)*dc*: 85, (*bd*)(*ab*)*e*: 180, (_*b*)*dbe*: 207	**17 high-*swu* sequences** *a*: 602, *aa*: 310, *ab*: 517, (*ab*): 602, *ad*: 602, *ae*: 387, *a*(*ab*): 310, *aba*: 310, *a*(*bd*): 310, *abe*: 387, *a*(*bd*)*a*: 310, (*ab*)*d*: 422, (*ab*)*e*: 387, *ada*: 410, *adb*: 387, *ade*: 387, *adbe*: 387
b	(_*d*)*fad*: 130, *dc*: 85, (*de*): 74, (_*d*)(*ab*)*e*: 180, *dbe*: 207	**8 high-*swu* sequences** *b*: 676, *ba*: 310, *bb*: 387, *bd*: 496, (*bd*): 310, *be*: 461, *bbe*: 387, (*bd*)*a*: 310
c	(_*f*)*b*(*de*): 74	**1 high-*swu* sequence** *c*: 226
d	*fad*: 130, *c*: 85, (_*e*): 74, (*ab*)*e*: 180, (_*f*)*c*: 67, *e*(*ab*)*dbe*: 207	**10 high-*swu* sequences** *d*: 743, *da*: 517, *db*: 387, *de*: 387, *dd*: 337, *dad*: 337, *d*(*ab*): 387, *dae*: 387, *d*(*ab*)*e*: 387, *dbe*: 387
e	(*ab*)*dc*: 85, (*ab*)*dbe*: 207	**8 high-*swu* sequences** *e*: 546, *ea*: 292, *eb*: 292, *ed*: 292, *e*(*ab*): 292, *ead*: 292, *e*(*ab*)*d*: 292, *ebd*: 292
f	*ad*: 130, *b*(*de*): 74, *c*: 67	**1 high-*swu* sequence** *f*: 271

sequential pattern growth mining approach to avoid the level-wise candidate generation-and-testing approach. Furthermore, its number of database scan is independent on the maximum candidate length. It always needs a maximum of three database scans. Therefore, it significantly reduces the overall runtime for mining high-utility sequential patterns.

First, the US algorithm scans the SDB once to detect length-1 *swu* sequences. Subsequently, it generates projected databases by considering length-1 *swu* sequences as prefixes with a second database scan. Then, using a pattern growth approach, it divides the search spaces (projected databases) recursively and applies the same technique into them. By utilizing this divide-and-conquer method, it generates very few candidates compared to the UL algorithm. Note that the US algorithm only generates the high-*swu* sequences without generating a large number of intermediate candidate high-*swu* sequences needed by the UL algorithm as shown in Table 2.

For prefix *a*, the UL algorithm generates a projected database (shown in Table 2). Here, (_*b*) means that the last item in the prefix, which is *a*, forms one element (*ab*). However, in the *a*-projected database, we get <*a*: 310, *b*: 517, _*b*: 602, *d*: 602, *e*: 387, *f*: 130, and *c*: 85>. Items *c* and *f* cannot form a candidate sequence with item *a* as they have low-*swu* values (130 and 85, respectively) in the *a*-projected database with respect to the *minSeqUtil*. As a consequence, 6 candidate sequences *a*: 602, *aa*: 602, *ab*: 517, (*ab*): 602, *ad*: 602, and *ae*: 387 are generated here. Now, according to the divide-and-conquer rule, we apply the same technique on the projected databases of *aa*, *ab*, (*ab*), *ad*, and *ae*. The *aa*-projected database contains {(_*bd*) *fad*: 130, (_*b*)*e*: 180}, and we get <*a*: 130, _*b*: 310, *d*: 130, *f*: 130, *e*: 180>. So, only one candidate sequence, *a*(*ab*): 310, is generated. The *ab*-projected database contains {(_*d*)*fad*: 130, (_*d*)(*ab*)*e*: 180, *e*: 207}, and we get <*a*: 310, *b*: 180, *d*: 130, _*d*: 310, *e*: 387, *f*: 130>. Candidate sequences *aba*: 310, *a*(*bd*): 310, and *abe*: 387 are generated here. Similarly, the other candidate sequences are generated. Table 2 shows that a total of 17 candidate sequences are generated by prefix-*a*. It also shows the other candidate sequences generated by other prefix items. However, as mentioned earlier, the US algorithm only generates the high-*swu* sequences as candidates. A third database scan is needed to discover the high-utility sequential patterns from the high-*swu* sequences. The US algorithm discovers the same set of resultant high-utility sequential patterns <*b*: 266, (*ab*): 239, (*ab*)*d*: 238, *adbe*: 226, *a*(*bd*)*a*: 231, *d*(*ab*)*e*: 250> as discovered by the UL algorithm. However, in this example, it generates only 45 candidates and scans the database three times in contrast to the 132 candidates and five database scans of the UL algorithm.

3. Algorithm Description and Analysis

The UL algorithm first generates all the length-1 candidates for high-*swu* sequences and calculates the set of length-1 high-*swu* sequences (lines 1 and 2). Subsequently, it uses the *for loop* described in lines 4 to 7 to determine the high-*swu* sequences for other levels. In each level, it first generates the candidates (line 5) and then scans the SDB for high-*swu* sequences (line 6). Finally, it determines the high-utility sequential patterns from all the sets of high-*swu* sequences (line 8).

The US algorithm first declares a set, *C*, and initializes to *NULL* in order to contain all the high-*swu* sequences (lines 1 and 2). Then, it invokes the recursive US procedure (line 3). This procedure is described in lines 4 to 14. Its three parameters are described in line 5. At the beginning, it determines all the high-*swu* items in the α-projected database SDB_α (line 7). Subsequently, it appends each high-*swu* item β with α in

UL Algorithm.

```
Begin
Input: A sequence database $SDB$ with utility values, $minSeqUtil$
Output: The complete set of high-utility sequential patterns
  1. Let $C_1$ be the set of length-1 candidates for high-$swu$
     sequences
  2. Scan $SDB$ once to find length-1 high-$swu$ sequences
  3. Let $F_1$ be the set of length-1 high-$swu$ sequences
  4. for ($k$=2, $F_{k-1}$≠$NULL$, $k$++)
  5.         $C_k$ = generate length-$k$ candidate $swu$ sequences
  6.         Scan $SDB$ once to find $F_k$
  7. End for
  8. Scan $SDB$ once to find high-utility sequential patterns
     from $U_k F_k$
End
```

US Algorithm.

```
Begin
Input: An SDB with utility values, $minSeqUtil$
Output: The complete set of high-utility sequential patterns
  1. Let $C$ be the set of all high $swu$ sequences
  2. Initialize $C$ to $NULL$
  3. Call US($NULL$, 0, $SDB$)
  4. Procedure US($\alpha$, $len$, $SDB_\alpha$)
  5. Let $\alpha$ be a high-$swu$ sequence, $len$ is the length of $\alpha$,
     $SDB_\alpha$ is the $\alpha$-projected sequence database if $\alpha$≠$NULL$,
     otherwise it is the $SDB$
  6. Begin
  7.   Scan $SDB_\alpha$ once and find each high-$swu$ item $\beta$, such that
       (a) $\beta$ can be appended to $\alpha$ to form a high-$swu$ sequence
       (b) $\beta$ can be assembled to the last element of $\alpha$ to form
           a high-$swu$ sequence
  8.   For each high-$swu$ item $\beta$
  9.     Append $\beta$ with $\alpha$ in appropriate form to generate a
         high-$swu$ sequence $\alpha'$
  10.    $C = C \cup \alpha'$
  11.    Construct $SDB_{\alpha'}$
  12.    Call US($\alpha'$, $len$+1, $SDB_{\alpha'}$)
  13.  End For
  14. End Procedure
  15. Scan SDB once to discover all high-utility sequential
      patterns from $C$.
End
```

appropriate form. Consider α=ab and β=c. If the first condition (line 7(a)) is true for β, then a new sequence abc is formed, otherwise a new sequence $a(bc)$ is formed. However, this new sequence α' is added in C (line 10). In lines 11 and 12, α'-projected database $SDB_{\alpha'}$ is constructed, and a recursive call is made to the US procedure, respectively. Finally, the SDB needs to be scanned to discover all the high-utility sequential patterns from the set of candidates C (line 15).

Observation 1 and lemma 3 show that the UL algorithm needs several database scans and generates a large number of candidates. On the other hand, the US algorithm significantly reduces the number of database scans and candidates compared to the UL algorithm.

Observation 1. The number of database scans needed by the UL algorithm is directly dependent on the maximum length of the candidate sequences as it adopts the level-wise Apriori mechanism. If the maximum length of candidate sequences is N, this algorithm requires N+1 database scans. On the other hand, the number of database scans required for the US algorithm is totally independent of the maximum length of candidate sequences. A maximum of three database scans are always required. As the minimum sequence utility threshold decreases, the number of candidate sequences and their maximum length also increases. Hence, as the $minSeqUtil$ decreases, running time increases very sharply in the UL algorithm.

Lemma 3. If N_1 is the number of candidates generated by the US algorithm, and N_2 is the number of candidates generated by the UL algorithm, then $N_1 \leq N_2$.

Proof. A sequence X $\{x_1, x_2, \ldots, x_n\}$ is a candidate sequence if all of its subsets of length-(n–1) are candidate sequences in the UL algorithm. This is because it is based on the level-wise Apriori mechanism. As a consequence, X may not be present in the database, or its utility value could be too low for it to become a candidate. In the US approach, if X is not present in the database, then it cannot appear in any projected database. Therefore, it cannot appear as a candidate. Moreover, after determining X is a low-swu sequence, it is pruned. Accordingly, a candidate set of the US algorithm contains only the actual high-swu sequences; hence, N_1 cannot be greater than N_2. □

V. Experimental Results

To evaluate the performance of our approach, we performed several experiments on synthetic datasets generated based on the principle introduced in [1] and using the source code available at [22]. These types of datasets have been used by most of the previous sequential data mining studies [1]-[5], [7], [19]-[21]. The parameters shown below are used to generate the datasets.

|D|: Number of customers
|C|: Average number of transactions per customer
|T|: Average number of items per transaction
|S|: Average length of maximal sequences
|I|: Average length of itemsets in maximal sequences
|N|: Number of distinct items

We generated two datasets for our experiments: D100K.C8.T6.S6.I5.N10K and D200K.C10.T8.S8.I7.N10K. Moreover, we used two real-life datasets: BMS-WebView-1 and BMS-WebView-2 [23], [24] contain web click-stream data. However, these datasets do not provide the internal and external utility values of the sequences. Most of the existing HUP mining algorithms [10]-[13] have generated random

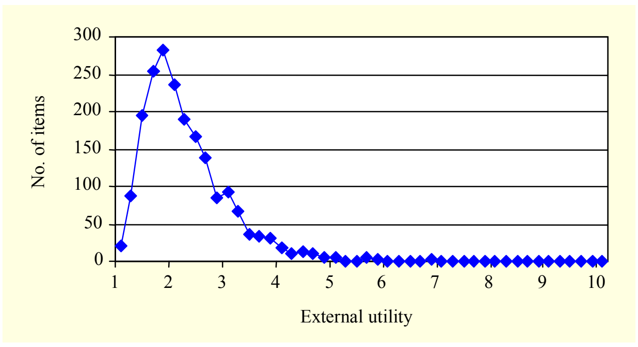

Fig. 2. External utility distribution for 2,000 distinct items using log-normal distribution.

numbers for internal and external utilities. We have generated random numbers for internal and external utilities ranging from 1 to 10 and 1.0 to 10.0, respectively. Moreover, based on our observation in real-world databases that most items carry low profit, we generated the profit values using a log-normal distribution. Most of the HUP mining research [10]-[13] has adopted the same technique. Figure 2 shows the external utility distribution of 2,000 distinct items using a log-normal distribution. Our programs were written in Microsoft Visual C++ 6.0 and run with the Windows XP operating system on a Pentium dual core 2.13 GHz CPU with 2 GB main memory.

At first, we compared the performance of UL and US algorithms on the D100K.C8.T6.S6.I5.N10K dataset. Figure 3(a) shows the comparison with respect to number of candidate sequences. The minimum sequence utility threshold range of 0.1% to 0.9% was used here. The *x*-axis of Fig. 3(a) shows the different minimum sequence utility thresholds, and the *y*-axis shows the number of candidate sequences. As discussed in section IV, the UL algorithm generates a large number of candidates due to the level-wise candidate generation-and-testing methodology. On the other hand, the US algorithm generates fewer candidates compared to the UL algorithm by exploiting a pattern growth approach. The result of Fig. 3(a) reflects the analysis of section IV. Figure 3(b) shows the comparison with respect to runtime. The *x*-axis of Fig. 3(b) shows the different minimum sequence utility thresholds, and the *y*-axis shows the runtime. Section IV also shows that the UL algorithm needs several database scans. Furthermore, the UL algorithm is directly dependent on the maximum length of candidate sequences. In contrast, the number of database scans needed by the US algorithm is not dependent on the maximum length of candidate sequences. It always needs a maximum of three database scans. Due to this achievement in scanning an SDB and reduced candidate set, the US algorithm significantly outperforms the UL algorithm as shown in Fig. 3(b).

Subsequently, we compared the UL and US algorithms with respect to different SDB sizes on the D200K.C10.T8.S8.I7.N10K dataset as shown in Fig. 3(c). The *x*-axis of Fig. 3(c) shows the different number of sequences, and the *y*-axis shows the runtime. For each database size shown in the *x*-axis, two mining operations are performed with two different minimum sequence utility thresholds (2% and 5%) for each algorithm. Figure 3(c) shows that the US algorithm significantly outperforms the UL algorithm in all the stages in this dataset. It also demonstrates that the runtime increases when the number of sequences increases or the minimum sequence utility threshold decreases. Furthermore, it shows the scalability of our proposed algorithms with an increasing number of sequences in this SDB. The US algorithm is also memory-efficient compared to the UL algorithm since it generates a small number of candidates. The US algorithm needs

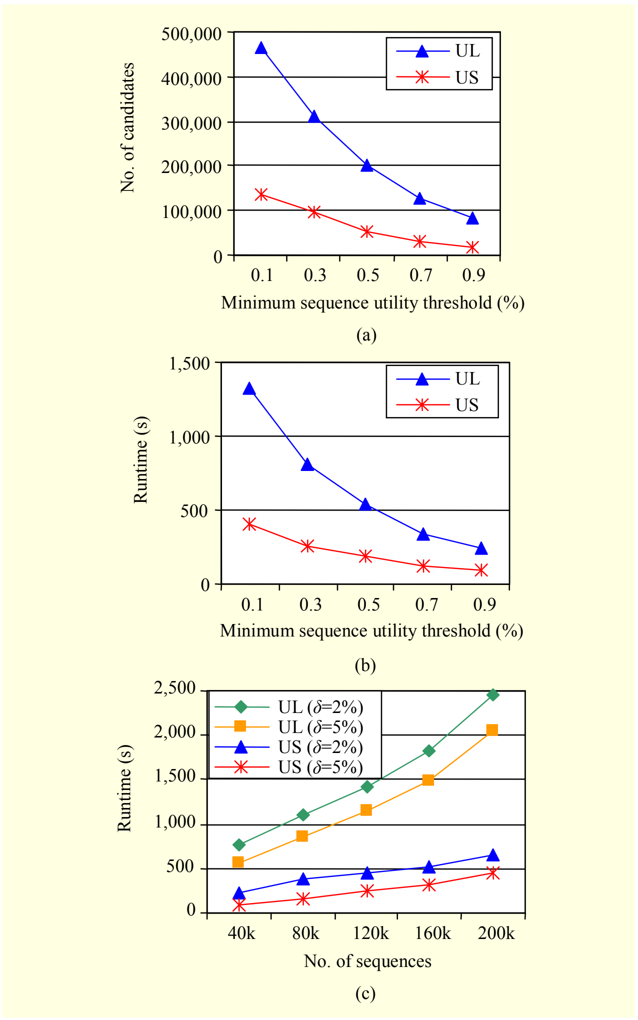

Fig. 3. Performance evaluation of proposed algorithms. (a) number of candidates compared on the D100K.C8.T6.S6.I5.N10K dataset, (b) runtime comparison on the D100K.C8.T6.S6.I5.N10K dataset, and (c) runtime comparison with different sequence database sizes on the D200K.C10.T8.S8.I7.N10K dataset.

3.481 MB memory in the D100K.C8.T6.S6.I5.N10K dataset (δ=0.1%) and 7.235 MB memory in the D200K.C10.T8.S8.I7.N10K dataset (δ=2%). In contrast, the UL algorithm needs 8.179 MB memory in the D100K.C8.T6.S6.I5.N10K dataset (δ=0.1%) and 22.693 MB memory in the D200K.C10.T8.S8.I7.N10K dataset (δ=2%).

Finally, we compared our proposed algorithms with the existing algorithms. As discussed in section II, the existing approaches are based on the support measure. However, our framework is based on the utility measure. To show the significance and efficiency of our proposed two algorithms, we compared them with the SQUIRE approach. Even though it is based on the support measure, SQUIRE considers non-binary frequency values of items. Hence, it is more related to our approach than the other methods. Section II also mentions that the SQUIRE approach presents two Apriori-based algorithms, Apriori-QSP and Apriori-All, as well as two pattern growth-based algorithms, PrefixSpan-QSP and PrefixSpan-All. We compared our Apriori-based algorithm, UL, with the Apriori-QSP and Apriori-All algorithms. We compared our pattern growth-based algorithm, US, with the PrefixSpan-QSP and PrefixSpan-All algorithms.

We have used two real-life datasets BMS-WebView-1 and BMS-WebView-2 [23], [24] to compare our algorithms with the existing algorithms. These datasets contain several months' worth of click-stream data from two e-commerce websites. Each transaction in these datasets is a web session consisting of all the product detail pages viewed in that session. That is, each product detail view is an item. The BMS-WebView-1 dataset has 59,602 transactions of 497 distinct items. The BMS-WebView-2 dataset has 77,512 transactions of 3,340 distinct items. The average transaction sizes are 2.5 and 5.0, respectively. In our experiments, an item is regarded as an item of a sequence, and a transaction is regarded as a sequence of items. Figure 4 shows the comparison of our Apriori-based algorithm UL with the existing Apriori-based algorithms, Apriori-QSP and Apriori-All, on the real-life BMS-WebView-1 dataset. Similarly, Fig. 5 shows the comparison of our pattern growth-based algorithm, US, with the pattern growth-based algorithms, PrefixSpan-QSP and PrefixSpan-All, on the real-life BMS-WebView-2 dataset.

The *x*-axes of Figs. 4 and 5 show the different number of sequences, and the *y*-axes shows the runtime. For each database size shown in the *x*-axes, a mining operation is performed with a minimum threshold of 0.06% in BMS-WebView-1 dataset (Fig. 4) and a minimum threshold of 0.05% in BMS-WebView-2 dataset (Fig. 5). For the existing algorithms, the minimum threshold value represents the minimum support value for a mining operation. For our

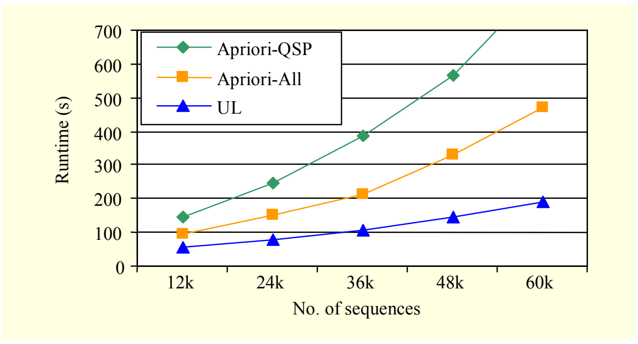

Fig. 4. Performance comparison with existing Apriori-based algorithms on BMS-WebView-1 dataset.

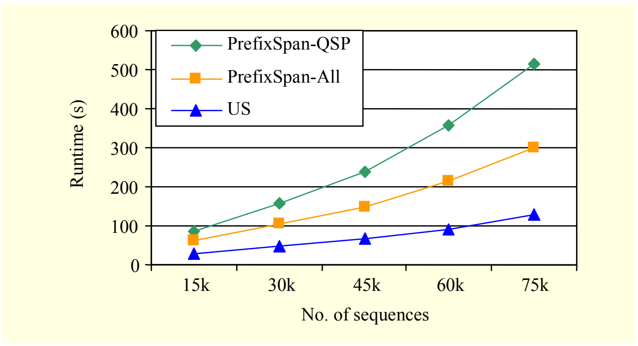

Fig. 5. Performance comparison with existing pattern growth-based algorithms on BMS-WebView-2 dataset.

algorithms, the minimum threshold value represents the minimum sequence utility value in a mining operation. Due to the support measure, the existing algorithms generate a large number of sequential patterns. In section II, it is noted that they are not applicable for high-utility sequential pattern mining. As a consequence, many unimportant sequential patterns may be generated with low-utility value. However, our algorithms use a utility measure and prune all the low-utility sequential patterns during the mining process. They avoid the generation of a huge amount of traditional sequential patterns and efficiently discover only high-utility sequential patterns. Therefore, our algorithms significantly outperform the existing algorithms as shown in Figs. 4 and 5.

Moreover, our algorithms also outperform the existing algorithms in memory usage since they avoid handling a large number of sequential patterns in the mining process. The UL algorithm needs 7.386 MB memory, the Apriori-All algorithm needs 12.109 MB memory, and the Apriori-QSP algorithm needs 15.724 MB memory on the BMS-WebView-1 dataset (minimum threshold=0.06%). The US algorithm needs 3.075 MB memory, the PrefixSpan-All algorithm needs 7.528 MB memory, and the PrefixSpan-QSP algorithm needs 10.491 MB memory on the BMS-WebView-2 dataset (minimum threshold=0.05%).

VI. Conclusion

In this paper, we proposed a novel framework for mining high-utility sequential patterns. Our proposed framework considers both internal and external utilities of a sequence and introduces a new measure, SeqUtility, to calculate the utility value of a sequence. It also defines the high-utility sequential patterns and the problem of mining high-utility sequential patterns. Furthermore, we proposed two new algorithms, UL and US, for mining high-utility sequential patterns. Of the two algorithms, UL is simpler and more straightforward for discovering high-utility sequential patterns. However, it suffers from the level-wise candidate generation-and-test problem and needs several database scans. The second algorithm, US, solves these problems by exploiting a sequential pattern growth approach. It generates a small number of candidates and needs a maximum of three database scans. Extensive performance analyses showed that our algorithms are very efficient and scalable for mining high-utility sequential patterns in sequence databases.

References

[1] R. Agrawal and R. Srikant, "Mining Sequential Patterns," *Proc. 11th Int. Conf. Data Eng.*, 1995, pp. 3-14.

[2] R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements," *Proc. 5th Int. Conf. Extending Database Technol.,* 1996, pp. 3-17.

[3] M.J. Zaki, "SPADE: An Efficient Algorithm for Mining Frequent Sequences," *Mach. Learning*, vol. 42, no. 1-2, Jan. 2001, pp. 31-60.

[4] J. Ayres et al., "Sequential Pattern Mining Using a Bitmap Representation," *Proc. 8th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2002, pp. 429-435.

[5] J. Pei et al., "Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 11, Oct. 2004, pp. 1424-1440.

[6] J. Pei et al., "PrefixSpan: Mining Sequential Patterns by Prefix-Projected Growth," *Proc. 17th Int. Conf. Data Eng.*, 2001, pp. 215-224.

[7] J. Wang, J. Han, and C. Li, "Frequent Closed Sequence Mining without Candidate Maintenance," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 8, 2007, pp. 1042-1056.

[8] H. Yao, H.J. Hamilton, and C.J. Butz, "A Foundational Approach to Mining Itemset Utilities from Databases," *Proc. 3rd SIAM Int. Conf. Data Mining*, 2004, pp. 482-486.

[9] H. Yao and H.J. Hamilton, "Mining Itemset Utilities from Transaction Databases," *Data Knowl. Eng.*, vol. 59, no. 3, 2006, pp. 603-626.

[10] Y. Liu, W.K. Liao, and A. Choudhary, "A Two Phase Algorithm for Fast Discovery of High Utility of Itemsets," *Proc. 9th Pacific-Asia Conf. Knowl. Discovery Data Mining*, 2005, pp. 689-695.

[11] C.F. Ahmed et al., "An Efficient Candidate Pruning Technique for HUP Mining," *Proc.13th Pacific-Asia Conf. Knowl. Discovery Data Mining*, 2009, pp. 749-756.

[12] Y.C. Li, J.S. Yeh, and C.C. Chang, "Isolated Items Discarding Strategy for Discovering High Utility Itemsets," *Data Knowl. Eng.*, vol. 64, no. 1, 2008, pp. 198-217.

[13] C.F. Ahmed et al., "Efficient Tree Structures for HUP Mining in Incremental Databases," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 12, 2009, pp. 1708-1721.

[14] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," *Proc. 2nd Int. Conf. Very Large Data Bases*, 1994, pp. 487-499.

[15] J. Han et al., "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach," *Data Mining Knowl. Discovery*, vol. 8, 2004, pp. 53-87.

[16] J. Han et al., "Frequent Pattern Mining: Current Status and Future Directions," *Data Mining Knowl. Discovery*, vol. 15, no. 1, 2007, pp. 55-86.

[17] M.N. Garofalakis, R. Rastogi, and K. Shim, "SPIRIT: Sequential Pattern Mining with Regular Expression Constraints," *Proc. 25th Int. Conf. Very Large Data Bases*, 1999, pp. 223-234.

[18] J. Pei, J. Han, and W. Wang, "Mining Sequential Patterns with Constraints in Large Databases," *Proc. 11th Int. Conf. Inform. Knowl. Management*, 2002, pp. 18-25.

[19] U. Yun, "A New Framework for Detecting Weighted Sequential Patterns in Large Sequence Databases," *Knowl.-Based Syst.*, vol. 21, no. 2, 2008, pp. 110-122.

[20] U. Yun, "WIS: Weighted Interesting Sequential Pattern Mining with a Similar Level of Support and/or Weight," *ETRI J.*, vol. 29, no. 3, June 2007, pp. 336-352.

[21] C. Kim et al., "SQUIRE: Sequential Pattern Mining with Quantities," *J. Syst. Software*, vol. 80, no. 10, 2007, pp. 1726-1745.

[22] http://www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/datasets/syndata.html

[23] Frequent Itemset Mining Dataset Repository. Available at: http://fimi.cs.helsinki.fi/data/

[24] Z. Zheng, R. Kohavi, and L. Mason, "Real World Performance of Association Rule Algorithms," *Proc. 7th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2001, pp. 401-406.

Chowdhury Farhan Ahmed received his BS and MS in computer science from the University of Dhaka, Bangladesh, in 2000 and 2002, respectively. From 2003 to 2004, he worked as a faculty member at the Institute of Information Technology, University of Dhaka, Bangladesh. Since 2004, he has been working as a faculty member in the Department of Computer Science and Engineering, University of Dhaka, Bangladesh. Currently, he is a PhD candidate in the Department of Computer Engineering at Kyung Hee University, Rep. of Korea, and expecting to be awarded the PhD in August, 2010. His research interests are in the areas of data mining and knowledge discovery.

Syed Khairuzzaman Tanbeer received his BS in applied physics and electronics and MS in computer science from the University of Dhaka, Bangladesh, in 1996 and 1998, respectively. He received his PhD in computer engineering from Kyung Hee University, Rep. of Korea, in 2010. Since 1999, he has been working as a faculty member in Department of Computer Science and Information Technology, Islamic University of Technology, Dhaka, Bangladesh. Currently, he is working as a postdoctoral fellow in the Department of Computer Engineering, Kyung Hee University, Rep. of Korea. His research interests include data mining, parallel and distributed mining, and knowledge discovery.

Byeong-Soo Jeong received his BS in computer engineering from Seoul National University, Rep. of Korea, in 1983. He received his MS in computer science from KAIST, Daejeon, Rep. of Korea, in 1985, and his PhD in computer science from the Georgia Institute of Technology, Atlanta, USA, in 1995. In 1996, he joined the faculty at Kyung Hee University, Rep. of Korea, where he is now a professor at the College of Electronics and Information. From 1985 to 1989, he was on the research staff at Data Communications Corp., Rep. of Korea. From 2003 to 2004, he was a visiting scholar at the Georgia Institute of Technology, Atlanta. His research interests include database systems, data mining, and mobile computing.