BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **19**, No 4
Sofia • 2019 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2019-0032

An Algorithm for Mining High Utility Sequential Patterns with Time Interval

Tran Huy Duong[1], *Demetrovics Janos*[2], *Vu Duc Thi*[3], *Nguyen Truong Thang*[1], *Tran The Anh*[1]

[1]*Institute of Information Technology, Vietnam Academy of Science and Technology, Viet Nam*
[2]*Computer and Automation Institute Hungarian Academy of Sciences, Hungary*
[3]*Thanglong University, Hanoi, Viet Nam*
E-mails: huyduong@ioit.ac.vn demetrovics@sztaki.mta.hu vdthi@vnu.edu.vn ntthang@ioit.ac.vn theanh@ioit.ac.vn

Abstract: *Mining High Utility Sequential Patterns* (*HUSP*) *is an emerging topic in data mining which attracts many researchers. The HUSP mining algorithms can extract sequential patterns having high utility* (*importance*) *in a quantitative sequence database. In real world applications, the time intervals between elements are also very important. However, recent HUSP mining algorithms cannot extract sequential patterns with time intervals between elements. Thus, in this paper, we propose an algorithm for mining high utility sequential patterns with the time interval problem. We consider not only sequential patterns' utilities, but also their time intervals. The sequence weight utility value is used to ensure the important downward closure property. Besides that, we use four time constraints for dealing with time interval in the sequence to extract more meaningful patterns. Experimental results show that our proposed method is efficient and effective in mining high utility sequential pattern with time intervals.*

Keywords: *Datamining, sequential pattern, time interval, high utility.*

1. Introduction

Data mining is a process for extracting knowledge from data. Data can be represented in many formats of structured data such as tables [15-18], graphs [19], sequences [1-5, 14], etc. Mining Sequential Patterns is one of the most important topics in data mining with many applications. Sequential data are very popular in real life data, like customer purchase sequence, medical treatment sequence, DNA sequence, weblogs sequence, and so on. The main purpose of sequential pattern mining is to extract all patterns frequently occurring in a sequence database. There are many works on sequential pattern mining with different approaches; some of them can be mentioned like AprioriAll [1], GSP [2] (Apriori approach), PrefixSpan [3] (PrefixSpan approach), SPADE [4], SPAM [5] (vertical database format approach). These traditional approaches have some drawbacks, such as that all items in a sequential pattern have the same importance and there is no quantitative information associated

with each item. But in real datasets, different items have different importance and each item can have quantitative values. To address this problem, Ahmed et al [6] proposed a new research problem named high utility sequential pattern mining, which considers not only quantities of items but also their importance.

The goal of mining High Utility Sequential Patterns (HUSP) is to find all sequences having a high utility in a quantitative sequence database. Each item in an item set of a sequence is assigned with a quantity and each item in the database is assigned with a profit indicating its importance. The utility of an item is the product of its quantity by its unit profit. The utility of a sequential pattern can be calculated in two ways: the summation utility and the maximum utility. Admed [6] proposed the summation utility, which means the utility of a sequential pattern is the summation of all distinct occurrences of that pattern in the database sequence. To simplify the calculation and to keep the meaning of the utility, later works on HUSP like [7-10] use the maximum utility measure to calculate the utility of a sequential pattern. The utility of a pattern is then calculated by the maximum value of all distinct occurrences of that pattern in the database sequence. The problem of HUSP is that it enumerates all sequential patterns having a utility no less than a predefined minimum utility threshold.

Although HUSP problem can discover all patterns having high utility, it does not include time intervals between items. For instance, suppose we have two sequences: S1: ⟨(Computer)(1 month, Printer)⟩, and S2 : ⟨(Computer)(6 month, Printer)⟩. If we do not consider time intervals, these sequences are the same. But in fact, we can say that the sequence S1 is more important than the sequence S2, since the S1 has smaller time interval than the S2. To solve this problem, some works on item interval were proposed in [11-13]. However, these works did not consider the item's significance when mining time interval sequential patterns.

Hence, in this paper, we integrate utility into time intervals sequential pattern mining and propose a new problem – high utility sequential pattern with time interval mining. We consider not only sequences' utilities, but also time intervals between items.

The remainder of this paper is organized as follows: Section 2 provides a study of related works. Section 3 describes the problems and proposes the mining method for high utility sequential pattern with time interval. Section 4 presents the experimental result. Conclusion and comments are presented in the last session.

2. Related works

2.1. Sequential pattern mining

In 1995, Agrawal and Srikant [1] developed the sequence pattern mining problem and proposed three Apriori based algorithms: AprioriAll, AprioriSome, and DynamicSome. Like Apriori, these algorithms scan database multiple times and as they are based on the level-wise technique, it takes much time for mining. Later, Agrawal and Srikant proposed a new method called GSP [2] to speed up execution efficiency in finding sequential patterns. However, GSP is still based on Apriori, so it still needed generating and testing execution.

In 2001 Pei et al. [3] introduced PrefixSpan algorithm which is based on pattern growth approach. It does not require multiple times database scanning, so it takes considerably less time of mining than other Apriori based algorithms. Zaki [4] devised SPADE algorithm, which is a sequential pattern mining using equivalent classes. SPADE used vertical database format and level-wise technique to generate and test if a pattern is frequent. The SPAM [5] algorithm uses a depth-first search strategy using an efficient vertical bitmap representation.

2.2. Time-interval sequential pattern mining

The difference between sequential pattern mining and time-interval sequential pattern mining is that latter takes into account the time interval between items. In 2003, Chen and Huang [11] proposed time-interval problem and two algorithms: I-Apriori and I-PrefixSpan which are based on Apriori [1] and PrefixSpan [3], respectively. In 2005, Chen, Chiang and Ko [12] extended previous work [11] by applying fuzzy theory to partition the time intervals using FTI-Apriori, an Apriori based algorithm that employs a distinct fuzzy membership function. Both of their works used extended sequence approach to represent time interval.

In 2006, Yu and Hayato [13] proposed a framework to generalized sequential pattern mining with item intervals. This work used four time constraints and the extended sequence approach to handle with item interval.

2.3. High Utility Sequential pattern mining

In a real-world dataset, not only occurrence frequency of patterns, but also their quantity and significance (like profit or price) have important roles. For example, the pattern ⟨iPhone X, MacBook Air⟩ may not be a high frequency pattern in a sequence database but it may contribute high profit to the shop due to its high profit. Thus, the low frequency pattern may contribute to high profit but they may not be found in the sequence database by using traditional sequence pattern mining approach. To solve this problem, a high utility sequential pattern was proposed in 2010 by work of Ahmed, Tanbeer and Jeong [6]. Admed proposed a new framework called high utility sequential pattern with two types of item's utility: internal utility (represent item's quantity) and external utility (represent item's importance like profit). Moreover, two new algorithms were introduced using level-wise technique (UL Algorithm) and pattern-growth technique (US Algorithm). In Ahmed's work, the utility of a pattern is calculated as the summation of utilities of all distinct occurrences in a sequence. This way of calculation may find some personal buying behaviors repeatedly, rather than common behaviors. To avoid such cases and simplify the utility calculation, later works on HUSP mining used maximum utility measure.

In 2012, Yin, Zheng and Cao [7] proposed a general framework for mining HUSP and represents USpan algorithm, which uses Sequence Weight Utility (SWU) for pruning candidates and two data constructions: LQS-tree and Utility Matrix for data representation. In 2014, Lan et al. [8] proposed PHUS, an algorithm based on a projection approach of PrefixSpan [3]. Their work used SWU as an upper bound for pruning candidates and a temporal sequence table for data representation.

Alkan and Karagoz [10] proposed a new upper bound called CRoM (Cummulated Rest of Match) used for pruning candidates before generation. They also represent the HuspExt algorithm with a Prefix tree structure for data representation. In 2019 (see [9]) is published a survey of High utility sequential pattern mining. This survey provided a concise overview of recent works in the HUSP mining field, presenting related problems and research opportunities. They also provided a formal theoretical framework for comparing upper bounds used by HUSP mining algorithms.

3. Problem statement and definitions

We use a Quantitative Sequence DataBase with time interval (QiSDB) given in Table 1 as an example. Each appearance of an item in the sequence is assigned with a positive quantity value. Each distinct item in QiSDB is assigned with a profit value as shows in Table 2.

Table 1. Quantitative sequence database with time interval (QiSDB)

iSID	Data sequence
iS_1	⟨0, *a*[3] ⟩⟨1, *a*[2] *b*[4] *d*[2] ⟩⟨2, *f*[1] ⟩⟨3, *a*[4] ⟩⟨4, *d*[1] ⟩
iS_2	⟨0, *e*[3] ⟩⟨1, *a*[2] *b*[6] ⟩⟨2, *d*[1] ⟩⟨3, *c*[2]⟩
iS_3	⟨0, *c*[1] *f*[3] ⟩⟨1, *b*[3] ⟩⟨2, *d*[1] *e*[3] ⟩
iS_4	⟨0, *a*[2] ⟩⟨1, *b*[6] *d*[4] ⟩⟨2, *a*[5] *b*[4] ⟩⟨3, *e*[5] ⟩
iS_5	⟨0, *d*[1] *f*[5] ⟩⟨1, *c*[1] ⟩⟨2, *g*[4] ⟩
iS_6	⟨0, *d*[2] ⟩⟨1, *e*[3] ⟩⟨2, *a*[5] *b*[7] ⟩⟨3, *d*[4] ⟩ ⟨4, *b*[2] ⟩⟨5, *e*[4]⟩
iS_7	⟨0, *a*[3] *b*[2] ⟩⟨1, *c*[2] ⟩⟨2, *e*[2]⟩⟨3, *f*[3]⟩
iS_8	⟨0, *a*[3]⟩⟨2, *d*[1] *f*[1]⟩
iS_9	⟨0, *a*[2] *c*[4]⟩⟨2, *e*[2]⟩

Table 2. Profit table

Item	Profit
a	3
b	2
c	1
d	6
e	5
f	2
g	8

The problem of mining high utility sequential patterns with time interval is then defined as follows.

3.1. Definitions

Definition 1. An *itemset* $X \subseteq I$ is a set of items in lexicographic order. If $|X| = r$ then itemset X is called r-itemset. $I = \{i_1, i_2, \ldots, i_n\}$ is a set of all items occur in QiSDB.

Definition 2. *Interval extended sequence:*

$iS = \langle (t_{1,1}, X_1), (t_{1,2}, X_2), \ldots, (t_{1,m}, X_m) \rangle$ is a list of the itemsets order by their occurrence time. Here, X_i $(1 \leq i \leq m)$ is an itemset and $t_{\alpha,\beta}$ is the time interval between itemsets X_α and X_β, then $t_{\alpha,\beta} = X_\beta.\text{time} - X_\alpha.\text{time}$

Definition 3. *Internal utility and external utility:* Internal utility of an item $i_j \in I$ in a sequence iS_a denoted as $iu(i_j, iS_a)$ is quantity of item i_j in iS_a. External utility of item i_j is its significant value and denoted as $eu(i_j)$.

Table 1 is a QiSDB with internal utility values and Table 2 is an external utility values table. The internal utility value represents items' quantities and external utility value represents profit per unit of that item. Item a in iS_9 has $iu(a,iS_9) = 2$, its external utility $eu(a)=3$. An item in a sequence may appear multiple times, in that case $iu(i_j, iS_k)$ is the maximum value among all the quantities of i_j in sequence iS_k. For example, $iu(a, iS_1)= 4$.

Definition 4. The *utility of an item* i_j in a sequence iS_a denoted as $su(i_j, iS_a)$ is defined by: $u(i_j, iS_a) = iu(i_j, iS_a) \times eu(i_j)$.

For example, $u(a, iS_1)= iu(a, iS_1) \times eu(a) = 4\times3=12$.

Definition 5. The *utility of a pattern* : $\alpha = \langle(t_{1,1}, X_1), (t_{1,2}, X_2), \dots, (t_{1,n}, X_n)\rangle$ is a pattern with length n and $\alpha \subseteq iS_a$, sequence utility of the pattern α in iS_a denoted as $u(\alpha, iS_a)$ is defined by

$$su(\alpha, iS_a) = \max\{\sum_{i_j \in \alpha} su(i_j, iS_a), \forall \alpha \in iS_a\}.$$

Definition 6. The ***sequence utility of an input sequence*** iS_a is the sum of utilities of all items in iS_a, which means

$$su(iS_a) = \sum_{i_j \in iS_a} su(i_j, iS_a).$$

Definition 7. The *utility of a pattern* α in a QiSDB denoted as $su(\alpha, QiSDB)$ and is defined by

$$su(\alpha, QiSDB) = \sum_{iS_a \in QiSDB} su(\alpha, iS_a).$$

Definition 8. The *utility of a* QiSDB is defined by:

$$su(QiSDB) = \sum_{iSa \in QiSDB} su(iS_a).$$

Definition 9. *Time constraints*: Given an interval extended sequence $\alpha = \langle(t_{1,1}, X_1), (t_{1,2}, X_2), \dots, (t_{1,n}, X_n)\rangle$, the time constraints are given as follows:

- C_1 = min_time_interval is a minimum item interval between any two adjacent itemsets, which means $t_{i,i+1} \geq$ min_time_interval for all $\{i|1 \leq i \leq n-1\}$.
- C_2 = max_time_interval is a maximum item interval between any two adjacent itemsets, which means $t_{i,i+1} \leq$ max_time_interval for all $\{i|1 \leq i \leq n-1\}$.
- C_3 = min_whole_interval is a minimum item interval between the first and the last itemset of the sequence, which means $t_{i,n} \geq$ min_whole_interval.
- C_4 = max_whole_interval is a maximum item interval between the first and the last itemset of the sequence, which mean $t_{i,n} \leq$ max_whole_interval.

Definition 10. *The high utility sequential pattern with time interval*: Given a quantitative sequence database with time interval QiSDB, each item $i_j \in I$ in the input sequences iS_a is assigned with an internal utility $iu(i_j,iS_a)$ and an external utility $eu(i_j)$. Given a minimum utility threshold minSeqUtil and four time constraints C_1, C_2, C_3, C_4, a sequential pattern $\alpha = \langle(t_{1,1}, X_1), (t_{1,2}, X_2), \dots, (t_{1,n}, X_n)\rangle$ is a high utility sequential pattern with time interval if it satisfies:

$su(\alpha, QiSDB) \geq$ minSeqUtil & $t_{\alpha,\beta}$ satisfies time constraints C_1, C_2, C_3, C_4.

Then the problem of mining high utility sequential pattern with time interval is defined as follows:

- Given a quantitative sequence database with time interval QiSDB, each item $i_j \in I$ in the input sequences iS_a is assigned with an internal utility iu(i_j,iS_a) and an external utility eu(i_j). Given a minimum utility threshold minSeqUtil and four time constraints C_1, C_2, C_3, C_4, find all high utility sequential patterns with time interval in QiSDB which means finding the set L such that:

$L = \{\alpha \subseteq \text{QiSDB} \mid \text{su}(\alpha, \text{QiSDB}) \geq \text{minSeqUtil} \ \& \ t_{\alpha,\beta}$ satisfies time constraints C_1, C_2, C_3, $C_4\}$

- The high utility sequential pattern with time interval does not satisfy the downward closure property, which means a subsequence of a high utility sequential pattern with time interval may not be a high utility sequential pattern with time interval.

3.2. The proposed solution

In this subsection, we propose an algorithm for mining high utility sequential patterns with time interval (UIPrefixSpan). Our main approach is to push time constraints and the utility threshold while still maintaining the downward closure property.

a. Projected database

To avoid checking every possible combination of a potential candidate sequences, we first fix the order of items within each element. Since items within an element of a sequence can be listed in any order, without loss of generality, one can assume that they are always listed alphabetically.

For example, the sequence is presented as ⟨(0, *a*[3]) (1, *a*[1]*b*[3]*c*[4]) (2, *c*[4]*a*[2])⟩ instead of ⟨(0, *a*[3]) (1, *a*[1]*c*[4] *b*[3]) (2, *a*[2]*c*[4])⟩. By such a convention, the expression of a sequence is unique.

If we follow the order of the prefix of a sequence and project only the postfix of a sequence, we can examine in an orderly manner all the possible subsequences and their associated projected database.

Definition 11. *Prefix and postfix of interval extended sequence*: Given an interval extended sequence $\alpha = \langle(t_{1,1}, X_1), (t_{1,2}, X_2), \ldots, (t_{1,n}, X_n)\rangle$, where X_β is an itemset, there exists an integer j ($1 \leq j \leq n$) that satisfies $X_\beta \subseteq X_j$ and $t_{1,\beta} = t_{1,j}$.

We define a prefix of interval extended sequence α with regard to (X_β, $t_{1,\beta}$) as follows:

$$\text{Prefix}\ (\alpha, X_\beta, t_{1,\beta}) = \langle(t_{1,1}, X_1), (t_{1,2}, X_2), (t_{1,3}, X_3), \ldots, (t_{1,j}, X_\beta)\rangle.$$

Then the postfix of interval extended sequence α with regard to (X_β, $t_{1,\beta}$) is defined as follows:

$$\text{Postfix}\ (\alpha, X_\beta, t_{1,\beta}) = \langle(t_{j,j}, X'_j), (t_{j,j+1}, X_{j+1}), \ldots, (t_{j,n}, X_n)\rangle$$

with X'_j being the subset of X_j after minus X_β. When $X'_j = \varnothing$, postfix of α with regard to (X_β, $t_{1,\beta}$) is defined as follows:

$$\text{Postfix}\ (\alpha, X_\beta, t_{1,\beta}) = \langle(t_{j,j+1}, X_{j+1}), (t_{j,j+2}, X_{j+2}), \ldots, (t_{j,n}, X_n)\rangle.$$

On the other hand, when there does not exist integer j, postfix of α with regard to (X_β, $t_{1,\beta}$) becomes:

$$\text{Prefix}\ (\alpha, X_\beta, t_{1,\beta}) = \varnothing,$$
$$\text{Postfix}\ (\alpha, X_\beta, t_{1,\beta}) = \varnothing.$$

Definition 12. *Projected database with interval extended sequence*: Given a sequence $\beta = \langle(t_{1,1}, X_1), (t_{1,2}, X_2), (t_{1,3}, X_3),..., (t_{1,m}, X_m)\rangle$, a projected database with regard to β denoted as $\text{QiSDB}|_\beta$ is all postfixes of all input sequence iS_a in QiSDB with regard to β.

b. Maintaining downward closure property:

In utility base framework, the Downward Closure Property (DCP) of the sequence utility is not always observed. That means a subset of a high utility sequence is not necessarily a high utility sequence. Thus, we cannot use sequencer utility for pruning the search space and we must use another value, which ensures DCP. The following definition of sequence weight utility is based on A h m e d, T a n b e e r and J e o n g [6].

Definition 13. *Sequence weight utility of sequence* α: Given a sequence α, the swu of α is defined as follows:

$$\text{swu}(\alpha) = \sum_{\alpha \subseteq \text{iS}_a \wedge \text{iS}_a \in \text{QiSDB}} \text{su}(\text{iS}_a).$$

Definition 14. *Candidate pattern*: Given a minimum threshold minSeqUtil, a sequential pattern α is called a candidate pattern if it satisfies

$\text{swu}(\alpha) \geq \text{minSeqUtil}$ and α satisfies time constraints C_1, C_2, C_3, C_4.

Lemma 1. The Sequence Weight Utility (SWU) maintains the Downward Closure Property (DCP).

P r o o f: Let α be a candidate pattern and d_α be a set of input sequences that contains α in QiSDB. Let β be a super-sequence of α then β cannot be presented in any sequence where α is absent. Therefore, the maximum sequence weight utility of β is $\text{swu}(\alpha)$. Then, if $\text{swu}(\alpha)$ is less than minimum utility threshold minSeqUtil then β is not a candidate pattern. ■

Lemma 2. Given a QiSDB and a minimum utility threshold minSeqUtil, the high utility sequential patterns with time interval is a subset of candidate patterns.

P r o o f: Let α be a high utility sequential pattern with time interval. According to Definition 5 and Definition 13, $\text{su}(\alpha, \text{QiSDB})$ must be less than or equal to $\text{swu}(\alpha)$. So, if α is a high utility sequential pattern, it must be a candidate pattern. ■

3.3. Mining high utility sequential pattern with time interval using prefix projected database algorithm (UIPrefixSpan Algorithm)

In this subsection, we propose UIPrefixSpan Algorithm for mining high utility sequential pattern with time interval. Our algorithm extends pattern growth approach with time interval and utility, which was based on PrefixSpan [3] algorithm and Y u and Y a m a n a [13] work which mines frequent sequential pattern with item interval.

Our algorithm (UIPrefixSpan) always needs maximum three QiSDB database scans. First, UIPrefixSpan scans QiSDB once to find length-1 candidate patterns. Then, in the second database scan, it generates projected databases with length-1 candidate patterns as prefixes. In the next step, it uses pattern growth approach to recursively generate candidate patterns. In the final database scan, it checks each of candidate patterns found in the previous step in its real utility and output all patterns with utility higher than minSeqUtil.

- *Input:*

(1) Quantitative sequence database with time interval QiSDB and external utility values table,

(2) Minimum threshold : minSeqUtil,

(3) Time constraints C_1, C_2, C_3, C_4,

- *Output:* Set of the high utility sequential pattern with time interval

UIPrefixSpan Algorithm

Start

1) Let $\alpha = \varnothing$.

2) Let $R = \varnothing$; $L = \varnothing$.

3) Scan QiSDB first time, calculate swu(i, QiSDB) value with each item i in QiSDB, find length-1 candidate satisfies condition swu(i, QiSDB) $\geq$ minSeqUtil

4) Loop with all i:

5) a) Let $\alpha = \langle(0, i)\rangle$.

6) - $R = \{R, \alpha\}$.

7) - Check condition su(α, QiSDB) $\geq$ minSeqUtil,

if satisfies then $L = \{L, \alpha\}$.

8) b) Excute recursive function R = subUIPrefixSpan(QiSDB$|_\alpha$, R, minSeqUtil, C_1, C_2, C_3, C_4).

9) End Loop;

10) Scan QiSDB and check condition su(α, QiSDB) $\geq$ minSeqUtil, with each $\alpha \in R$, If satisfies then $L = \{L, \alpha\}$.

11) Output L.

End.

Function subUIPrefixSpan (QiSDB$|_\alpha$, R, minSeqUtil, C_1, C_2, C_3, C_4)

Start:

1) Scan QiSDB$|_\alpha$, calculate swu(i) with all item i and find all pairs of i and its time interval, denoted as (Δt,i) that satisfy swu(i) $\geq$ minSeqUtil, C_1 and C_2.

2) Let $\alpha = \langle\alpha, (\Delta t, i)\rangle$.

3) Check if α satisfies condition C_4 or not

4) Only if α satisfies C_4,

5) a) Excute R = subUIPrefixSpan (QiSDB$|_\alpha$,R, minSeqUtil, C_1, C_2, C_3, C_4).

6) b) When α satisfies C_3,

7) $R = \{R, \alpha\}$.

8) Return R .

End.

The UIPrefixSpan algorithm declares a pattern α and two sets: R is a set of all the candidate patterns and L is a set of all the high utility sequential patterns; and it initializes them to null (line 1 and 2). Then, it scans QiSDB once and finds all length-1 candidate patterns (line 3). Next, it executes a loop with all candidate patterns found in the previous step (line 4-9). A pattern α is defined by putting zero time value to each candidate (since we cannot determine time interval with length-1 pattern, so we put zero as its time interval) and put it into the candidate set R (line 4 to 6). The real utility value of α is checked after that, if it's utility greater than or equal to minSeqUtil then α is a high utility sequential pattern with time interval, and we put it in the high utility sequential patterns L (line 7). Next, the UIPrefixSpan calls

recursive function subUIPrefiSpan to generate all candidate patterns by using a pattern-growth approach (line 8). The third database scan is then executed to check if every candidate pattern is a high utility sequential pattern or not and puts those patterns which are high utility patterns in the high utility sequential patterns set L. Finally, we output the set L.

The function subUIPrefixSpan generates candidate patterns with time interval in the projected database. First, it scans the projected database and find all pairs of items and their time interval with sequence weight utility equal or higher than minimum threshold minSeqUtil and time interval of that pair must also satisfy two time constraints C_1 and C_2 (line 1). Then it joins pairs found in the previous step with database's prefixes to create new interval extended sequence patterns (line 2). Each new pattern then is checked with time constraints C_4 (line 4), if it satisfies C_4 then recursively function subUIPrefixSpan is called with new prefix to get a new pattern (line 5). After that, the new pattern is checked with constraint C_3, if it satisfies C_3 then it is a candidate pattern, we put it in the set of candidate pattern R and output the set R (line 6-8).

4. Experimental results

For evaluating the effectiveness of our algorithm, we performed several experiments on synthetic datasets generated using an IBM data generator which was introduced in [1]. The parameters of the IBM data generator are as follows:

$|D|$ is the number of customers;

$|C|$ is the average number of transactions per customer;

$|T|$ is the average number of items per transaction;

$|S|$ is the average length of maximal sequences;

$|I|$ is the average length of itemsets of maximal sequences;

$|N|$ is the number of distinct items.

We generated four synthetic datasets: D10K.C9.T8.S7.I8.N1K (DS1), D10K.C5.T4.S5.I6.N1K (DS2), D100.C10.T5.S10.I5.N1K (DS3), D200K.C10.T9.S9.I7.N1K (DS4). Moreover, we used a real-life dataset: BMS-WebView-1 (DS5) containing 59,601 web click-stream data sequences with 497 data items, and the average length of a sequence is 2.42; there are some long sequences (more than 318 sequences with more than 20 items).

However, these datasets do not provide the internal utility and external utility of sequences. Most of the others HUP mining algorithms have generated random numbers for internal and external utilities. Thus, we have generated random numbers for internal and external utilities ranging from 1 up to 5 and from 1.0 up to 10.0, respectively. In real-life databases, most items carry low profit. So we generated external utility (which carry profit values) using a log-normal distribution. Fig. 1 shows the external utility distribution of 1000 items in synthetic datasets (DS1-DS4).

Datasets mentioned above do not contain occurrence time, so we generated occurrence time according to itemsets' order. Which means in each sequence, first itemset has occurrence at time 0, second itemset has occurrence at time 1, third itemset has occurrence at time 2, and so on.

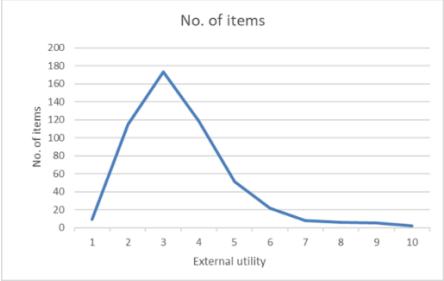
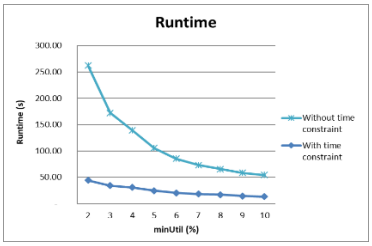


Fig 1. External utility distribution for 1000 items using log-normal distribution

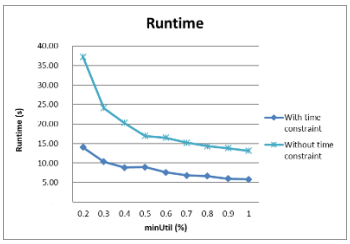
All experiments were performed on a computer which has an Intel Core i7 - 3.6 Ghz processor and 8 GB of memory, running Microsoft Windows 10. All algorithms were written in Java 1.8.

4.1. Performance test

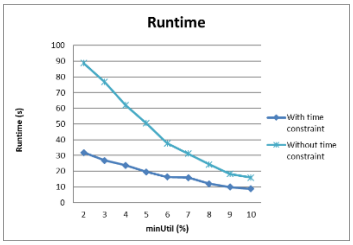
We executed performance test of UIPrefixSpan in two cases: with time constraints C_1=0; C_2=5; C_3=0; C_4=20 (UIPrefixSpan1) and without time constraint ((UIPrefixSpan2).



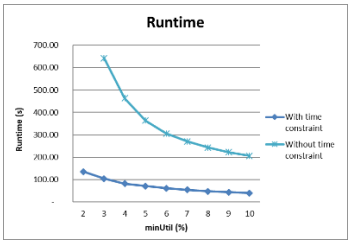
(a) D10K.C9.T8.S7.I8.N1K



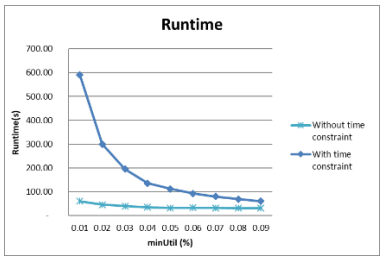
(b) D10K.C5.T4.S5.I6.N1K



(c) D100.C10.T5.S10.I5.N1K



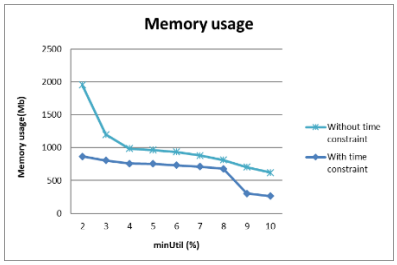
(d) 200K.C10.T9.S9.I7.N1K



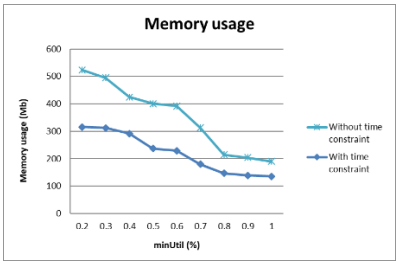
(e) BMS-WebView-1

Fig 2. Runtime

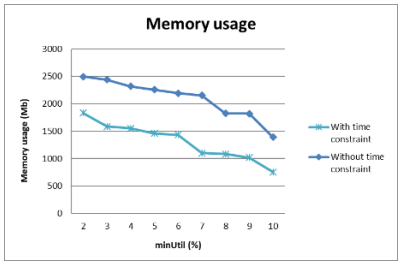
As shown in Fig. 2, UIPrefixSpan1 performs faster than UIPrefixSpan2. When the minUtil is decreased, the runtime of UIPrefxiSpan2 significantly increases and in case of DS4, UIPrefixSpan2 cannot run with low minUtil (2%). In contrast, UIPrefixSpan1 runs well with low minUtil and much faster than UIPrefixSpan2 in both synthetics (DS1-DS4) and real dataset (DS5). It is because when we used time constraints (UIPrefixSpan1), less candidates were generated, so the search space was reduced and runtime decreased.



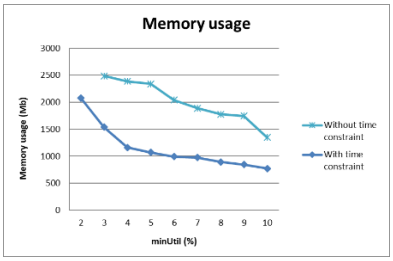
(a) D10K.C9.T8.S7.I8.N1K



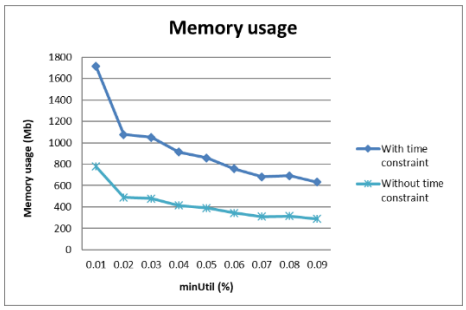
(b) D10K.C5.T4.S5.I6.N1K



(c) D100.C10.T5.S10.I5.N1K



(d) D200K.C10.T9.S9.I7.N1K

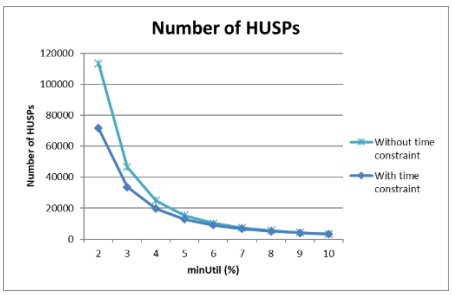


(e) BMS-WebView-1

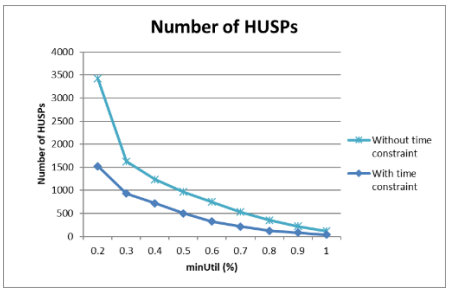
Fig 3. Memory usage

UIPrefixSpan1 also uses less memory than UIPrefixSpan2 as shown in Fig. 3. On DS1, UIPrefixSpan1 uses 1.2 times less memory than UIPrefixSpan2 and in some cases (2, and 9, and 10%), it uses 2.2 times less memory. On DS2, UIPrefixSpan1 uses 1.4 times less memory than UIPrefixSpan2 and with the low minUtils (<0.7 percent), it uses 1.6 times less memory. On other datasets, UIPrefixSpan1 also runs from 1.6 up to 1.8 times faster than UIPrefixSpan2. Generally, for all datasets, when minUtil is decreased, the memory usage increases. The memory usage of UIPrefixSpan2 also increases faster than UIPrefixSpan1's when the minUtil is

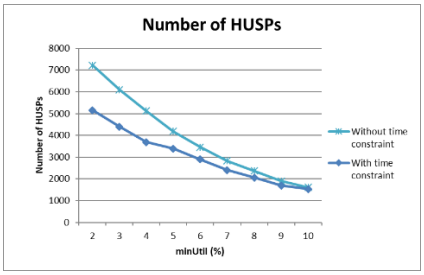
decreased. That is because when we used time constraints, the search space was reduced and that made our algorithm use less memory.



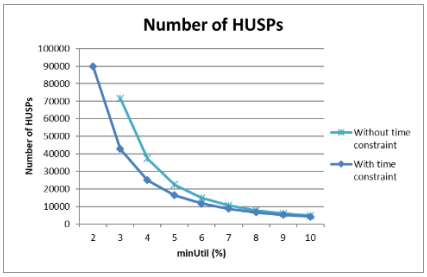
(a) D10K.C9.T8.S7.I8.N1K



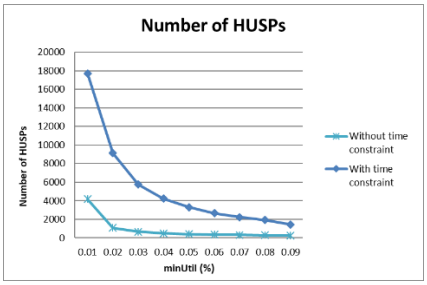
D10K.C5.T4.S5.I6.N1K



(c) D100.C10.T5.S10.I5.N1K



(d) D200K.C10.T9.S9.I7.N1K



(e) BMS-WebView-1

Fig. 4. Number of high sequential patterns with time interval

Despite that, a number of high sequential patterns found in UIPrefixSpan1 are less than UIPrefixSpan2 (Fig. 4), but those patterns are more meaningful. By using time constraints, less meaningful patterns generating can be avoided.

4.2. Scalability test

We performed scalability tests of the UIPrefixSpan algorithm on D200K.C10.T9.S9.I7.N1K (DS2) dataset with different database sizes. We set minimum utility threshold minUtil to 3%. The result shows that runtime is increased linearly as database size increased. UIPrefixSpan algorithm shows good scalability in both cases (Fig. 5).

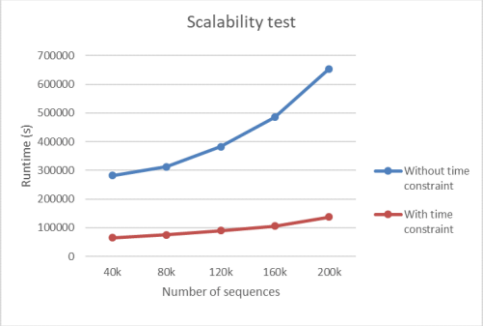


Fig 5. Scalability test

5. Conclusion

In this paper, we developed an algorithm called UIPrefixSpan which detects high utility sequential patterns with time interval based on the candidate pattern growth model. We consider not only the occurrence frequency of patterns but also their utility and time interval.

We used a prefix-projected database building method, which allows to significantly reduce the search space when mining the high utility sequential patterns with time interval. By using sequence weight utility value, UIPrefixSpan maintains downward closure property in mining sequential patterns with high utility. Moreover, by adding time constraints, our proposed algorithm excludes extraction of interval extended sequences with time intervals in which the user is not interested. Extensive performances showed that UIPrefixSPan was efficient and scalable in high utility sequential pattern mining.

With above comments, we can conclude that UIPrefixSpan is an efficient algorithm for mining high utility sequential patterns with time interval.

References

1. A g r a w a l, R., R. S r i k a n t. Mining Sequential Patterns. – In: Proc. of International Conference on Data Engineering (ICDE'95), 1995.
2. A g r a w a l, R., R. S r i k a n t. Mining Sequential Patterns: Generallizations and Performance Improvements. – Lecture Notes in Computer Science, Vol. **1057**, 1996, pp. 3-17.
3. P e i, J., J. H a n, B. M. A s i, H. P i n o. PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. – In: Proc. of Seventeenth International Conference on Data Engineering, 2001.
4. Z a k i, M. SPADE: An Efficient Algorithm for Mining Frequent Sequences. – Machine Learning, Vol. **40**, 2000, pp. 31-60.
5. A y r e s, J., J. G e h r k e, T. Y i u, J. F l a n n i c k. Sequential Pattern Mining Using Bitmap Representation. –In: Proc. of ACM SIGKDD'02, 2002.
6. A h m e d, C. F., S. K. T a n b e e r, B. S. J e o n g. A Novel Approach for Mining Highutility Sequential Patterns in Sequence Databases. – ETRI Journal, 2010, pp. 676-686.
7. Y i n, J., Z. Z h e n g, L. C a o. USpan: An Efficient Algorithm for Mining High Utility Sequential. – In: Proc. of 18th ACM SIGKDD International Conference on Knowledge, 2012.

8. L a n, G. C., T. P. H o n g, V. S. T s e n g, S. L. W a n g. Applying the Maximum Utility Measure in High Utility Sequential Pattern Mining. – Expert Syst. Appl., Vol. **41**, 2014, No 11, pp. 5071-5081.
9. T r u o n g-C h i, T., P. F o u r n i e r-V i g e r. A Survey of High Utility Sequential Pattern Mining. – High-Utility Pattern Mining: Theory, Algorithms and Applications, Vol. **51**, P. Fournier-Viger, J. Lin, R. Nkambou, B. Vo, V. Tseng, Eds., Cham, Springer, 2019.
10. A l k a n, O. K., P. K a r a g o z. CRoM and HuspExt: Improving Efficiency of High Utility Sequential Pattern Extraction. – In: 32nd IEEE International Conference on Data Engineering (ICDE'16), Helsinki, 2016.
11. C h e n, Y.-L., T. C.-H. H u a n g. Discovering Time-Interval Sequential Patterns in Sequence Databases. – Expert Systems with Applications, Vol. **25**, 2003, No 3, pp. 343-354.
12. C h e n, Y.-L., M.-C. C h i a n g, M.-T. K o. Discovering Fuzzy Time-Interval Sequential Patterns in Sequence Databases. – IEEE Transactions on Systems Man and Cybernetics, Vol. **35**, 2005, No 5, pp. 959-972.
13. Y u, H., H. Y a m a n a. Generalized Sequential Pattern Mining with Item. – Journal of Computers, Vol. **1**, 2006, No 3, pp. 51-60.
14. D e m e t r o v i c s, J., V. D. T h i, T. H. D u o n g. An Algorithm to Mine Normalized Weighted Sequential Patterns Using Prefix-Projected Database. – Serdica Journal of Computing, Vol. **9**, 2015, No 2, pp. 105-122.
15. D e m e t r o v i c s, J., H. M. Q u a n g, V. D. T h i, N. V. A n h. An Efficient Method to Reduce the Size of Consistent Decision Tables. – Acta Cybern., Vol. **23**, 2018, No 4, pp. 1039-1054.
16. D e m e t r o v i c s, J., V. D. T h i, N. L. G i a n g. On Finding All Reducts of Consistent Decision Tables. –Cybernetics and Information Technology, Vol. **14**, 2014, No 4, pp. 3-10.
17. D e m e t r o v i c s, J., N. T. L. H u o n g, V. D. T h i, N. L. G i a n g. Metric Based Attribute Reduction Method in Dynamic Decision Tables. – Cybernetics and Information Technologies, Vol. **16**, 2016, No 2, pp. 3-15.
18. C a o, C. N., J. D e m e t r o v i c s, N. L. G i a n g, V. D. T h i. About a Fuzzy Distance between Two Fuzzy Partitions and Application in Attribute Reduction Problems. – Cybernetics and Information Technologies, Vol. **16**, 2016, No 4, pp. 13-28.
19. D e m e t r o v i c s, J., H. M. Q u a n g, N. V. A n h, V. D. T h i. An Optimization of Closed Frequent Subgraph Mining Algorithm. – Cybernetics and Information Technologies, Vol. **17**, 2017, No 1, pp. 3-14.

Received: 29.05.2019; Second Version: 18.11.2019; Accepted: 26.11.2019