Mining High Utility Sequential Patterns Using Maximal Remaining Utility

Wei Song(✉) and Keke Rong

College of Computer Science and Technology,
North China University of Technology, Beijing 100144, China
songwei@ncut.edu.cn

Abstract. Mining high utility sequential pattern is an interesting problem in data mining. In this paper, we propose a new algorithm called high utility sequential pattern mining based on maximal remaining utility (HUSP-MRU). In HUSP-MRU, the maximal remaining utility (MRU) is defined as tighter upper bound of candidates. Representing the search space with lexicographic sequential pattern tree, the matrix structures are used for MRU storage, and branch as well as node pruning based on MRU are used for improving mining efficiency. Extensive tests conducted on publicly available datasets show that the proposed algorithm outperforms USpan algorithm in terms of mining efficiency.

Keywords: Data mining · High utility sequential pattern
Maximal remaining utility · Item-utility matrix · Pruning strategy

1 Introduction

High utility sequential pattern (HUSP) mining [1, 7], which is to discover sequential patterns with high utility in a sequence database, is an important topic in the field of utility pattern mining [3, 6], and has wide real-world applications [5, 8].

Different from traditional frequent sequential pattern [4], HUSP considers both occurrence frequency and profit of itemsets in a sequence. Thus, the infrequent but high profit sequential patterns can be discovered through mining HUSPs.

USpan [7] is a typical HUSP mining algorithm. For width pruning, USpan use the sequence-weighted utilization as the upper bound, which may include unnecessary operations on utility computation. To improve the efficiency, a new HUSP mining algorithm named HUSP-MRU is proposed. In this algorithm, the maximal remaining utility is defined as a tighter upper bound. To store maximal remaining utility, the item-utility matrix and maximal utility matrix structures are used. The search space is represented by lexicographic sequential pattern tree, and the branch as well as node pruning strategies are used to avoid unnecessary traversal of unpromising nodes. Experimental results show that the HUSP-MRU algorithm is efficient.

Y. Tan et al. (Eds.): DMBD 2018, LNCS 10943, pp. 466–477, 2018.
https://doi.org/10.1007/978-3-319-93803-5_44

2 Problem of HUSP Mining

Let $\boldsymbol{I} = \{i_1, i_2, \ldots, i_m\}$ be a finite set of items, set $X \subseteq \boldsymbol{I}$ is called an *itemset*. An itemset sequence S_r is an ordered list of itemsets $< X_1, X_2, \ldots, X_n >$. Each itemset X_d in sequence S_r is denoted as S_r^d. A sequence database ***SDB*** is a set of tuples of the form (*sid*, *S*), where *sid* is the ID of sequence *S*.

The *internal utility* $q(i, S_r^d)$ represents the quantity of item i in itemset S_r^d of sequence S_r. The *external utility* $p(i)$ is the unit profit value of item i. The *utility of item* i in itemset S_r^d is defined as $u(i, S_r^d) = p(i) \times q(i, S_r^d)$. The *utility of itemset* X in S_r^d of sequence S_r is defined as $u(X, S_r^d) = \sum_{i \in X} u(i, S_r^d)$.

A sequential pattern $\alpha = < X_1, X_2, \ldots, X_p >$ is called a *subsequence* of a sequence $S_r = < S_r^1, S_r^2, \ldots, S_r^n >$, denoted as $\alpha \sqsubseteq S_r$, if there exist integers $1 \leq j_1 < j_2 < \ldots < j_p \leq n$ such that $X_1 \subseteq S_r^{j_1}, X_2 \subseteq S_r^{j_2}, \ldots, X_p \subseteq S_r^{j_p}$. We say the ordered list of itemsets $< S_r^{j_1}, S_r^{j_2}, \ldots, S_r^{j_p} >$ an *occurrence* of α in S_r. The set of all occurrences of α in S_r is denoted as $OccSet(\alpha, S_r)$.

In this paper, the term *sequence* refers to the sequence database transactions, while the term *sequential pattern* refers to the extracted subsequences from these sequences.

Let $occ = < S_r^{j_1}, S_r^{j_2}, \ldots, S_r^{j_p} >$ be an occurrence of $\alpha = < X_1, X_2, \ldots, X_p >$ in sequence S_r. The utility of α with respect to *occ* is defined as $u(\alpha, occ) = \sum_{i=1}^{p} u(X_i, S_r^{j_i})$, and utility of α in S_r is defined as $u(\alpha, S_r) = max\{u(\alpha, occ) \,|occ \in OccSet(\alpha, S_r)\}$. The *sequence utility* (SU) of a sequence S_r is defined as $SU(S_r) = u(S_r, S_r)$. The utility of a sequential pattern α in ***SDB*** is defined as $u(\alpha) = \sum_{\alpha \sqsubseteq S_r \wedge S_r \in SDB} u(\alpha, S_r)$.

The *minimum utility threshold* δ, specified by the user, is defined as a percentage of the total SU values of the database, whereas the *minimum utility value* is defined as $min\_util = \delta \times \sum_{Sr \in SDB} SU(S_r)$. An sequential pattern α is called an HUSP if $u(\alpha) \geq min\_util$. Given a sequence database ***SDB***, the task of HUSP mining is to determine all sequential patterns that have utilities no less than *min_util*. Similar to the algorithms in [1, 7], the concept of remaining utility is also used in our algorithm to speed up the mining process.

Let $occ = < S_r^{j_1}, S_r^{j_2}, \ldots, S_r^{j_p} >$ be an occurrence of $\alpha = < X_1, X_2, \ldots, X_p >$ in sequence S_r, the last item of X_p, say item i, is called *extension item* of α. The *remaining sequential pattern* of S_r w.r.t. position j_p, denoted as $S_r/(\alpha, j_p)$, is defined as the subsequence of S_r from the item after the i to the end of S_r. The *remaining utility* of α w.r.t. the sequential pattern $S_r/(\alpha, j_p)$, denoted as $ru(\alpha, S_r/(\alpha, j_p))$, is the sum of the utilities of all items in $S_r/(\alpha, j_p)$. Among all values of $ru(\alpha, S_r/(\alpha, j_p))$, the one with the highest value is called the *maximal remaining utility* of α in S_r, denoted as $mru(\alpha, S_r)$.

Consider the sequence database in Table 1 and the profit table in Table 2. In the example database, the utility of item a in second itemset of S_2 (i.e., S_2^2) is $u(a, S_2^2) = 6 \times 3 = 18$, the utility of itemset $\{ae\}$ in S_2^2 is $u(\{ae\}, S_2^2) = 18 + 2 = 20$. Both $occ_1 = < S_4^1, S_4^2 >$ and $occ_2 = < S_4^1, S_4^3 >$ are occurrences of α = <*ba*> in S_4. So

$u(\alpha, S_4) = max\{u(\alpha, occ_1),\ u(\alpha, occ_2)\} = max\{31,\ 22\} = 31$. In the example database, $u(\alpha) = u(\alpha, S_3) + u(\alpha, S_4) + u(\alpha, S_5) = 11 + 31 + 28 = 70$. Given $min\_util = 60$, as $u(\alpha) > min\_util$, α is an HUSP. Consider the occurrence $occ_2 = < S_4^1, S_4^3 >$ of α in S_4, the remaining sequential pattern $S_4/(\alpha,\ 3) = <\{be\}>$, and $ru(\alpha, S_4/(\alpha,\ 3)) = 5 + 2 = 7$. Since $ru(\alpha, S_4/(\alpha,\ 2)) = 6 + 12 + 5 + 2 = 25$, which is higher than $ru(\alpha, S_4/(\alpha,\ 3))$, so $mru(\alpha, S_4) = 25$.

Table 1. Example sequence database

SID	Sequence
S_1	<(*e*,5) {(*c*,2), (*f*,1)} (*b*,2)>
S_2	<(*c*,1) {(*a*,6), (*d*,3), (*e*,2)}>
S_3	<{(*a*,2), (*e*,6)} {(*a*,1), (*b*,1), (*c*,2)} {(*a*,2), (*d*,3), (*e*,3)}>
S_4	<{(*b*,2), (*e*,2)} {(*a*,7), (*d*,3)} {(*a*,4), (*b*,1), (*e*,2)}>
S_5	<{(*b*,2), (*e*,3)} {(*a*,6), (*e*,3)} {(*a*,2), (*b*,1)}>

Table 2. Profit table

Item	*a*	*b*	*c*	*d*	*e*	*f*
Profit	3	5	4	2	1	2

3 HUSP-MRU Algorithm

3.1 The Matrix Structure

For HUSP-MRU, an item-utility matrix structure is defined for each sequence.

Definition 1. Let $S_r = < S_r^1, S_r^2, \ldots, S_r^n >$ be a sequence, the *item-utility matrix* of sequence S_r is an $m \times (2 \times n)$ matrix denoted as $\boldsymbol{M_r}$, where m is the number of items in S_r. Each row of $\boldsymbol{M_r}$, representing item i, is composed of n pairs, and each pair is with the form of $p_k = < u_k(i),\ iru_k(i) > (1 \le k \le n)$, where $u_k(i)$ is calculated by Eq. 1.

$$u_k(i) = \begin{cases} u(i, S_r^k), i \in S_r^k \\ 0, i \notin S_r^k \end{cases} \tag{1}$$

Let "$\prec$" be the lexicographic order of items, if item $j \prec i$, and there exists no item q such that $j \prec q \prec i$, $iru_k(i)$ is calculated by Eq. 2.

$$iru_k(i) = \begin{cases} ru(i, S_r/(<i>, k)), i \in S_r^k \\ iru_k(j), i \notin S_r^k \wedge k \neq 1 \\ SU(S_r), i \notin S_r^k \wedge k = 1 \end{cases} \tag{2}$$

According to Definition 1, for two items i and j, if $j \prec i$ and there exists no item q such that $j \prec q \prec i$, we have $iru_k(j) = u_k(i) + iru_k(i)$, which can facilitate the calculation of item-utility matrix. Given item-utility matrix, the maximal utility as well as maximal remaining utility are also recorded in the maximal utility matrix.

Definition 2. Let $S_r = <S_r^1, S_r^2, \ldots, S_r^n>$ be a sequence, the *maximal utility matrix* of sequence S_r is an $m \times 3$ matrix, where m is the number of items in S_r. For each row, the first element (denoted as i) is the item this row represents, the second element is $u(i, S_r)$, the third element is $mru(i, S_r)$.

As the utility of sequential pattern is based on the maximal utility among all occurrences, the maximal utility matrix structure is convenient for HUSP mining.

Consider S_3 in Table 1. The item-utility matrix of S_3 is shown in Table 3, and the maximal utility matrix of S_3 is shown in Table 4.

Table 3. The item-utility matrix of S_3

Item	S_3^1		S_3^2		S_3^3	
	u	iru	u	iru	u	iru
a	6	37	3	28	6	9
b	0	37	5	23	0	9
c	0	37	8	15	0	9
d	0	37	0	15	6	3
e	6	31	0	15	3	0

Table 4. The maximal utility matrix of S_3

Item	mu	mru
a	6	37
b	5	23
c	8	15
d	6	3
e	6	31

3.2 Lexicographic Sequential Pattern Tree

Given a sequential pattern $\alpha = <X_1, X_2, \ldots, X_p>$ and an item x, $\alpha \Diamond x$ means α concatenates with x. It can be *I-extension*, $\alpha \Diamond_I x = <X_1, X_2, \ldots, X_p \cup x>$; or *S-extension*, $\alpha \Diamond_S x = <X_1, X_2, \ldots, X_p, x>$. For example, <{$ae$}> is an I-extension of <a>, while <ae> is a S-extension of <a>.

The search space of the HUSP-MRU can be represented as a lexicographic tree. A lexicographic sequential pattern tree (LSP-Tree) T is a tree structure composed as follows: (1) One empty root labeled as "< >". (2) Each node N except root consists of four fields: *N.seq*, *N.ulist*, *N.u*, and *N.mru*, where *N.seq* records the sequential pattern represented by N, *N.ulist* is composed of information for search space pruning (see Definition 3), *N.u* is $u(N.seq, S_r)$, and *N.mru* is $mru(N.seq, S_r)$, where $S_r \in \boldsymbol{SDB}$.

Given a node N in an LSP-Tree T, the sequential pattern represented by a child node N' of N (i.e., $N'.seq$) is either an I-extension sequential pattern or an S-extension sequential pattern of $N.seq$.

Figure 1 shows a part of the LSP-Tree of the sequence database in Table 1. All the children nodes of a parent node are ordered lexicographically with I-extension sequential patterns before S-extension sequential patterns.

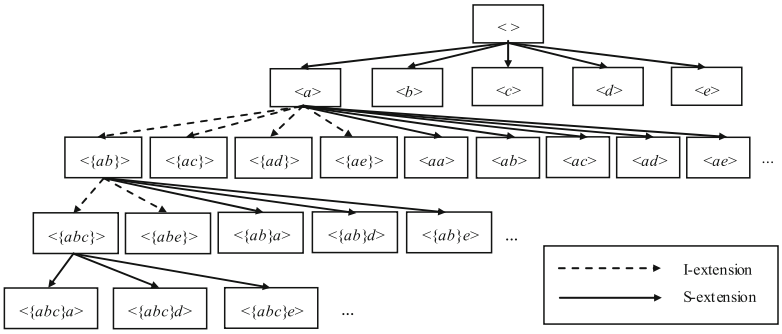

Fig. 1. Partial LSP-Tree for the example sequence database in Table 1

Definition 3. Let ***SDB*** be a sequence database, N be a node of LSP-Tree T representing sequential pattern $\alpha = <X_1, X_2, \ldots, X_p>$. Each entry of $N.ulist$ corresponds to an element of $g(\alpha) = \{S_r \in \boldsymbol{SDB} | \alpha \sqsubseteq S_r\}$, and consists of five fields: *SID*, *TID*, u, ru and *pointer*, where *SID* is the sequence ID, *TID* is the itemset ID of extension item of α occurs, u and ru are utility and remaining utility w.r.t. the enumerating occurrence, *pointer* links to the next occurrence of α in *SID*, or "null" if there is none.

For the node N representing sequential pattern <*ba*>, the $N.ulist$ is illustrated in Fig. 2, while $N.u$ and $N.mru$ are shown in Table 5.

SID	*TID*	*u*	*ru*	*pointer*
S_3	S_3^3	11	9	null
S_4	S_4^2	31	25	
S_5	S_5^2	28	14	

SID	*TID*	*u*	*ru*	*pointer*
S_4	S_4^3	22	7	null
S_5	S_5^3	16	5	null

Fig. 2. The *ulist* structure of <*ba*>

Table 5. *N.u* and *N.mru* of <*ba*>

SID	*u*	*mru*
S_3	11	9
S_4	31	25
S_5	28	14

3.3 Searching Strategies

After constructing the LSP-Tree, our algorithm traverses the search space from the root using a depth-first strategy. For a sequential pattern $\alpha = <X_1, X_2, \ldots, X_p>$, the HUSP-MRU algorithm checks its extension item to perform I-extension and S-extension, respectively. To facilitate the operation of sequential pattern extension, the item-utility matrix can be used. For I-extension, only items in the entries below the extension item are checked, and the set of all potential I-extension items is denoted as $PIS(\alpha)$. For S-extension, only items in the entries after the itemset containing extension item are checked, and the set of all potential S-extension items is denoted as $PSS(\alpha)$.

Here, we consider the extension case of sequential pattern <{*ab*}> in S_3. Here item *b* is the extension item. As shown in Table 3, items *c*, *d* and *e* are larger than *b* according to the lexicographic order, so entries of the last three rows of the matrix could be concatenated after *b* for I-extension. Specifically, only S_3^2 includes {*ab*}, so only item *c* can be used to form the I-extension, and the result is <{*abc*}>. The utility of <{*abc*}> is the utility of $u(\{ab\}, S_3)$ plus the newly added item's utility $u(c, S_3)$, i.e., $u(\{abc\}, S_3) = 8 + 8 = 16$.

We still use the sequential pattern <{*ab*}> to show the idea of S-extension. As we can see from Table 3, items that can be used for S-extension to <{*ab*}> are located in the third column of the item-utility matrix of S_3. Thus, sequential patterns <{*ab*}*a*>, <{*ab*}*d* > and <{*ab*}*e*> are the candidates. Take <{*ab*}*a*> for example, $u(<\{ab\}a>, S_3) = u(\{ab\}, S_3) + u(a, S_3^3) = 8 + 6 = 14$.

3.4 Pruning Strategies

Without pruning, an LSP-Tree search will consider every node. The key to an efficient LSP-Tree search is to remove entire branches or unpromising nodes from consideration.

In the search space formed by LSP-Tree, when a new node N_α representing sequential pattern $\alpha = <X_1, X_2, \ldots, X_p>$ is constructed, we compute

$$bu(\alpha) = \sum_{\alpha \sqsubseteq S_r \wedge S_r \in SDB} (u(\alpha) + mru(\alpha, S_r)) \tag{3}$$

If $bu(\alpha) < min\_util$, then we stop traversing all children nodes of N_α, we call this pruning strategy *branch pruning*.

When the node N_α passes the branch pruning, and can be used to form potential sequential patterns, then the items in $PIS(\alpha)$ or $PSS(\alpha)$ will be checked one by one. Specifically, for each item $i \in PIS(\alpha)$ or $i \in PSS(\alpha)$, we compute

$$nu(\alpha) = \sum_{\alpha \sqsubseteq S_r \wedge S_r \in SDB} (u(\alpha') + mru(i, S_r)) \tag{4}$$

where α' is the result of I-extension or S-extension of α with item i. If $nu(\alpha) < min\_util$, then we do not need to extend α with i, and we call this pruning strategy *node pruning*.

3.5 Algorithm Description

The proposed HUSP-MRU algorithm for mining HUSPs is described in Algorithm 1.

Algorithm 1	**HUSP-MRU(α)**
Input:	A sequential pattern α, sequence database ***SDB***, minimum utility value *min_util*
Output:	All high utility sequential patterns
1	**if** $bu(\alpha)<min\_util$ **then**
2	**return**;
3	**end if**
4	**for** each item $i \in PIS(\alpha)$ **do**
5	**if** $nu(\alpha)<min\_util$ **then**
6	Delete i from $PIS(\alpha)$;
7	**continue**;
8	**end if**
9	$\alpha_I \leftarrow$I-extension(α, i);
10	**if** $u(\alpha_I) \geq min\_util$ **then**
11	Output α_I;
12	**end if**
13	HUSP-MRU(α_I)
14	**end for**
15	**for** each item $i \in PSS(\alpha)$ **do**
16	**if** $nu(\alpha)<min\_util$ **then**
17	Delete i from $PSS(\alpha)$;
18	**continue**;
19	**end if**
20	$\alpha_S \leftarrow$S-extension(α, i);
21	**if** $u(\alpha_S) \geq min\_util$ **then**
22	Output α_S;
23	**end if**
24	HUSP-MRU(α_S)
25	**end for**
26	Output all HUSPs.

In Algorithm 1, Steps 1–3 perform the branch pruning strategy. If the current sequence pattern can not be used to generate HUSP, it returns to its parent sequential pattern. The loop (Steps 4–14) checks each item in $PIS(\alpha)$ for I-extension of α. Steps 5–8 perform the node pruning strategy. Then, new sequential pattern α_I concatenated by I-extension of α is generated in Step 9. If the utility of the extension result is no lower

than the minimum utility value, it is output as an HUSP in Step 11. Step 13 recursively invokes itself to go deeper in the LSP-Tree. Similarly, the loop (Steps 15–25) checks each item in $PSS(\alpha)$ for S-extension of α. Finally, Step 26 outputs all of the discovered high utility sequential patterns.

4 Performance Evaluation

We evaluate the performance of our HUSP-MRU algorithm and compare it with the USpan [7] algorithm. The source code of USpan was downloaded from the SPMF data mining library [2].

Table 6. Characteristics of the datasets

Dataset	#Sequences	#Items	AvgLength	Type of data
Bible	36369	13905	21.64	Book
BMS	59601	497	6.02	Click-stream
Kosarak10k	638811	39998	11.64	Click-stream
SIGN	730	267	93.00	Sign language

4.1 Experimental Environment and Datasets

The experiments were performed on a computer with 3.40 GHz CPU, 8 GB memory, and running on 64-bit Microsoft Windows 7. Our program was written in Java. Four real datasets, downloaded from the SPMF data mining library [2], were used for evaluation, and their characteristics are presented in Table 6.

Bible is moderately dense and contains many medium length sequences. BMS contains many short sequences. Kosarak10k is a sparse dataset that contains short sequences and a few very long sequences. SIGN is a dense dataset having very long sequences.

4.2 Running Time

We demonstrate the efficiency performance of the two algorithms. When measuring the runtime, we varied the minimum utility threshold for each dataset.

Figures 3, 4, 5 and 6 show the execution time comparisons for the four datasets. We can see that the HUSP-MRU algorithm is always faster than USpan algorithm. On average, HUSP-MRU is faster than USpan by 25.16%, 12.40%, 14.81% and 28.71% on Bible, BMS, Kosarak10k and SIGN, respectively. The reason for the results is that the branch pruning and node pruning strategies can avoid unnecessary search space traversal operations. Furthermore, with item-utility matrix and maximal utility matrix, computation of utilities of candidate sequential patterns can be accelerated.

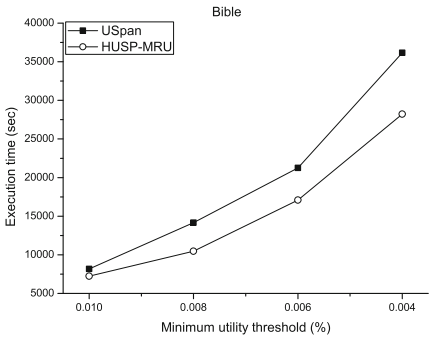

Fig. 3. Execution times on Bible

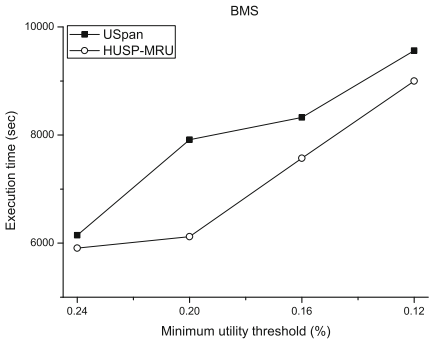

Fig. 4. Execution times on BMS

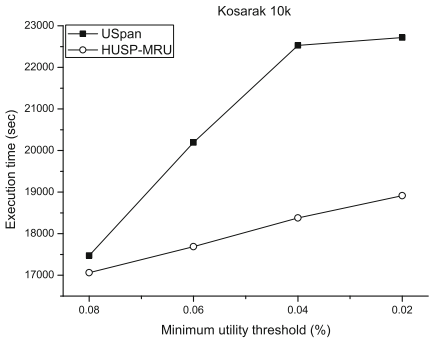

Fig. 5. Execution times on Kosarak10k

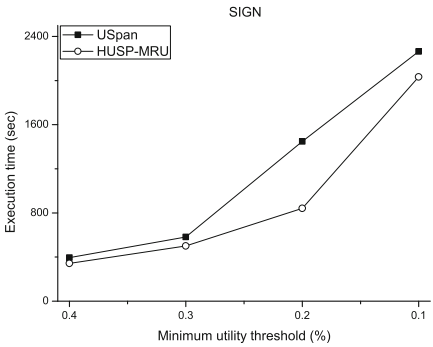

Fig. 6. Execution times on SIGN

4.3 Effect of Pruning Strategies

To demonstrate the benefit of branch pruning and node pruning, we compare the performance of HUSP-MRU with and without pruning strategies. In Figs. 7, 8, 9 and 10, we use HUSP-MRU-B to denote the baseline implementation of HUSP-MRU without pruning strategies.

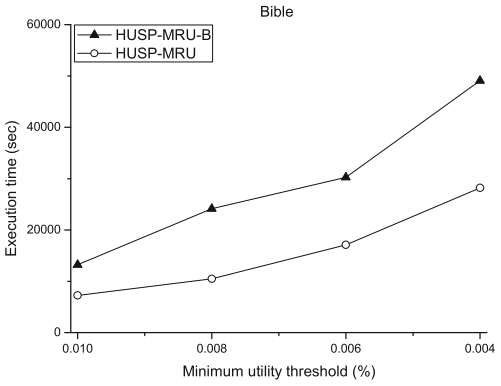

Fig. 7. Execution times of HUSP-MRU with and without pruning strategies on Bible

As shown in Figs. 7, 8, 9 and 10, the pruning strategies are beneficial in improving the efficiency. On Bible and BMS, the implementation with pruning strategies spends about half of the total time of the baseline algorithm. The most obvious case appears on the dataset of SIGN that HUSP-MRU is 5.08 times faster than HUSP-MRU-B on average.

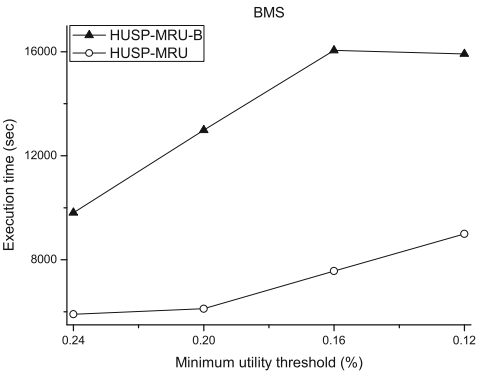

Fig. 8. Execution times of HUSP-MRU with and without pruning strategies on BMS

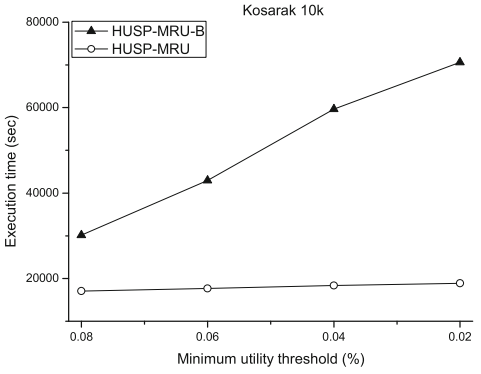

Fig. 9. Execution times of HUSP-MRU with and without pruning strategies on Kosarak10k

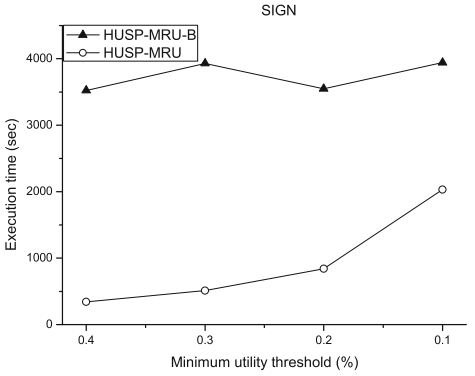

Fig. 10. Execution times of HUSP-MRU with and without pruning strategies on SIGN

5 Conclusions

In this paper, we proposed an HUSP mining algorithm called the HUSP-MRU based on the maximal remaining utility. In the HUSP-MRU algorithm, matrix structures are used to store information for utility computation. The search space is organized as the LSP-Tree and traversed using depth-first manner. To improve the mining efficiency, branch and node pruning strategies are used. Experiments on four datasets show that the HUSP-MRU algorithm outperforms USpan algorithm.

Acknowledgments. This work was supported by Beijing Natural Science Foundation (4162022), and High Innovation Program of Beijing (2015000026833ZK04).

References

1. Alkan, O.K., Karagoz, P.: CRoM and HuspExt: improving efficiency of high utility sequential pattern extraction. IEEE Trans. Knowl. Data Eng. **27**, 2645–2657 (2015)
2. Fournier-Viger, P., Lin, J.C.-W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., Lam, H.T.: The SPMF open-source data mining library version 2. In: Berendt, B., Bringmann, B., Fromont, É., Garriga, G., Miettinen, P., Tatti, N., Tresp, V. (eds.) ECML PKDD 2016. LNCS (LNAI), vol. 9853, pp. 36–40. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46131-1_8
3. Liu, Y., Liao, W.-k., Choudhary, A.: A two-phase algorithm for fast discovery of high utility itemsets. In: Ho, T.B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 689–695. Springer, Heidelberg (2005). https://doi.org/10.1007/11430919_79
4. Mooney, C.H., Roddick, J.F.: Sequential pattern mining - approaches and algorithms. ACM Comput. Surv. **45**, 1–39 (2013)
5. Shie, B.-E., Cheng, J.-H., Chuang, K.-T., Tseng, V.S.: A one-phase method for mining high utility mobile sequential patterns in mobile commerce environments. In: Jiang, H., Ding, W., Ali, M., Wu, X. (eds.) IEA/AIE 2012. LNCS (LNAI), vol. 7345, pp. 616–626. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31087-4_63
6. Song, W., Zhang, Z., Li, J.: A high utility itemset mining algorithm based on subsume index. Knowl. Inf. Syst. **49**, 315–340 (2016)
7. Yin, J., Zheng, Z., Cao, L.: USpan: an efficient algorithm for mining high utility sequential patterns. In: 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 660–668. ACM Press, New York (2012)
8. Zihayat, M., Davoudi, H., An, A.: Mining significant high utility gene regulation sequential patterns. BMC Syst. Biol. **11**, 1–14 (2017)