International Journal of Computational Intelligence Systems
Vol. **12(1)**; 2019, *pp.* 342–350
DOI: https://doi.org/10.2991/ijcis.2019.125905650; ISSN: 1875-6891; eISSN: 1875-6883
https://www.atlantis-press.com/journals/ijcis/

A New Algorithm of Mining High Utility Sequential Pattern in Streaming Data

Huijun Tang, Yangguang Liu*, Le Wang

School of Information Engineering, Ningbo Dahongying University, N0. 899, XueYuan Road, YinZhou District Ningbo, Zhejiang, P.R. China, 315175

ARTICLE INFO

Article History
Received 03 Jan 2019
Accepted 11 Jan 2019

Keywords
High utility sequential pattern
Data streaming
Sliding windows
Tree structure
Header table

ABSTRACT

High utility sequential pattern (HUSP) mining has emerged as a novel topic in data mining, its computational complexity increases compared to frequent sequences mining and high utility itemsets mining. A number of algorithms have been proposed to solve such problem, but they mainly focus on mining HUSP in static databases and do not take streaming data into account, where unbounded data come continuously and often at a high speed. The efficiency of mining algorithms is still the main research topic in this field. In view of this, this paper proposes an efficient HUSP mining algorithm named HUSP-UT (utility on Tail Tree) based on tree structure over data stream. Substantial experiments on real datasets show that HUSP-UT identifies high utility sequences efficiently. Comparing with the state-of-the-art algorithm HUSP-Stream (HUSP mining over data streams) in our experiments, the proposed HUSP-UT outperformed its counterpart significantly, especially for time efficiency, which was up to 1 order of magnitude faster on some datasets.

1. INTRODUCTION

Sequential pattern mining discovers sequences of itemsets that frequently appear in a sequence database. For example, in market basket analysis, mining sequential patterns from the purchase sequences of customer transactions is to find the sequential lists of itemsets that are frequently appeared in a time order. However, the traditional methods of sequential pattern mining lose sight of the number of occurrences of an item inside a transaction (e.g., purchased quantity), so is the importance (e.g., unit price/profit) of an item in the order of the transaction. Thus, not only some infrequent patterns that bring high profits to the business may be missed, but also a large number of frequent patterns having low selling profits are discovered. To address the issue, high utility sequential pattern (HUSP) mining [1–3] has emerged as a challenging topic in data mining.

In HUSP mining, each item has an internal number and external weight (e.g., profit/price), the item may appear many times in one transaction, the goal of HUSP mining is to find sequences whose total utility are more than a predefined minimum utility value. A number of improved algorithms have been proposed to solve this problem in terms of execution time, memory usage, and number of generated candidates [4–9]. But they mainly focus on mining HUSP in static dataset and do not consider the streaming data. However, in real world there are many data streams (DSs) such as wireless sensor data, transaction flows, call records, and so on. Users are more interested in information that reflects recent data rather than old ones in a period of time. So it has been an important research issue in the field of data mining to mine HUSP over DSs.

Many studies [1, 10–14] have been conducted to mine sequential patterns over DSs. For example, A. Marascu *et al.* propose an algorithm based on sequences alignment for mining approximate sequential patterns in web usage DSs. B. Shie *et al.* aim at integrating mobile data mining with utility mining for finding high utility mobile sequential patterns, two types of algorithms named level-wise and tree-based methods [12] are proposed to mine high utility mobile sequential patterns. Chang *et al.* proposed SeqStream [14] for mining closed sequential patterns over DSs. However, all these methods are for finding frequent sequential patterns, despite its usefulness, sequential pattern mining over DS has the limitation that it neither considers the frequency of an item within an item set nor the importance of an item (e.g., the profit of an item). Thus, some infrequent sequences with high profits may be missed. Although some preliminary works have been conducted on this topic, they may have the following deficiencies: 1. They are not developed for HUSP mining over DS and may produce too many patterns with low utility (e.g., low profit). 2. Most of the algorithms produce too many candidates and the efficiency of algorithms still need to be improved.

In 2017, Morteza Zihayat [15] *et al.* proposed HUSP-Stream for mining HUSP based on sliding windows. Two efficient data structures named ItemUtilLists (Item Utility Lists) and High Utility Sequential Pattern Tree (HUSP-Tree) for maintaining the essential information of HUSPs were introduced in the algorithm. To the best of our knowledge, HUSP-Stream is the first method to find HUSP over DSs. Experimental results on both real and synthetic

* *Corresponding author. Email:* y.g.liu@foxmail.com

datasets demonstrate impressive performance of HUSP-Stream, it is the state-of-the-art algorithm for mining HUSP in stream data.

The efficiency of mining algorithms is still the main research topic in this field [16–18], in this paper, we propose a new mining algorithm named HUSP-UT (utility on Tail Tree), the newly proposed algorithm is compared with the state-of-the-art HUSP-Stream algorithm in the experiments, theoretical analysis, and experiments are carried out to prove its effectiveness. The organization of this article is as follows: Section 2 provides a description of the problem and defines relevant terms, Section 3 introduces a structure UT-tree and a corresponding algorithm, Section 4 shows experimental results under different scenes, and Section 5 gives conclusions.

2. PROBLEM STATEMENT AND DEFINITIONS

A DS can formally be defined as an infinite sequence of transactions, DS = $\{t_1, t_2, \cdots, t_m, \cdots\}$, where t_i is the ith arrival of transactions. t_j could be known as $\{(x_1{:}c_1), (x_2{:}c_2), \cdots, (x_v{:}c_v)\}$, where v is the length of the transaction t_j. $c_k = q(x_k, t)$ is the quantity of item x_k and $p(x_k)$ is the profit value. An example of a sequential DS could be seen from Tables 1 and 2 is the external utility of the items.

Table 1 | High utility transaction dataset.

Transaction	Sequences
t_1	(a, 2)(d, 3)(e, 4)(a, 1)
t_2	(a, 2)(c, 8)(d, 2)(c, 1)(g, 2)
t_3	(a, 2)(c, 8)(b, 1)(d, 2)
t_4	(a, 4)(d, 8)(b, 1)
t_5	(a, 3)(c, 2)(d, 2)
t_6	(a, 6) (d, 4) (b, 5) (c, 4)
t_7	(a, 2)(c, 2)(b, 7)(a, 1)
t_8	(a, 4) (d, 3) (b, 4) (f, 1)

Table 2 | A profit table.

Item	Profit
a	2
b	6
c	3
d	8
e	10
f	1
g	1

Definition 1. The utility value of item x in transaction t is set as $u(x, t)$, it is defined as

$$u(x,t) = q(x,t) * p(x) \tag{1}$$

For example, in Tables 1 and 2, $u(d, t_1) = 8 * 3 = 24$. The item may appear in the transaction more than once, the maximum utility of an item among all its occurrences in t is used as its utility in the transaction. Thus $u(a, t_1) = 2 * 2 = 4$.

Definition 2. The utility of sequential itemset X in transaction t is denoted as $u(X, t)$, it is defined as

$$u(X,t) = \begin{cases} 0, & \text{if } X \not\subseteq t, \\ \sum_{x \in X} u(x,t), & \text{if } X \subseteq t. \end{cases} \tag{2}$$

For example, in Tables 1 and 2, $u(\{de\}, t_1) = u(d, t_1) + u(e, t_1) = 24 + 40 = 64$.

Definition 3. The utility of sequential itemset X in the current window of the data stream (WDS) is denoted as $u(X)$. It is defined as

$$u(X) = \sum_{t \in WDS} u(X,t) \tag{3}$$

For example, in Tables 1 and 2, if the current window size $w = 3$, if consists of t_1, t_2, t_3. $u(\{ad\}) = u(\{ad\}, t_1) + u(\{ad\}, t_2) + u(\{ad\}, t_3) = 28 + 20 + 20 = 68$.

Definition 4. The utility of sequential transaction t is denoted as $stu(t)$, it is defined as

$$stu(t) = \sum_{x \in t} u(x,t) \tag{4}$$

For example, in Tables 1 and 2, $stu(t_4) = u(a, t_4) + u(d, t_4) + u(b, t_4) = 8 + 64 + 6 = 78$.

Definition 5. The current WDS is denoted as $swu(WDS)$, and it is defined as

$$swu(WDS) = \sum_{t \in WDS} stu(t) \tag{5}$$

For example, in Tables 1 and 2, if the current window size $w = 3$, it consists of t_1, t_2, t_3. $swu(WDS) = stu(t_1) + stu(t_2) + stu(t_3) = 70 + 49 + 50 = 169$.

Definition 6. Give a user-specified threshold $\delta(0 < \delta < 1)$, minimum utility threshold ($MinUti$) set is defined as

$$MinUti = \delta * swu(WDS) \tag{6}$$

An sequential itemset X is an HUSP if its utility is no less than the minimum utility $MinUti$.

Definition 7. An sequential itemset X in the current window of DS is a candidate if its utility is no less than the minimum utility value.

$$swu(X) = \sum_{\substack{X \subseteq t \\ t \in WDS}} stu(t) \tag{7}$$

3. MINING HUSP IN DS

3.1. UT-Tree

A tree structure named UT-tree is introduced in this section. There are two types of nodes in the structure of UT-tree, one is ordinary-node [19], the other is tail-node [20], compared to the ordinary-node, tail-node owes the utility of the transactions in the window. Also, the tail pointer table is introduced in the construction of the UT-tree, which is used to delete obsolete data for updating.

We use an example to illustrate the construction of UT-tree based on the data in Tables 1 and 2. The window size w = 3 and in each batch of data contains two transactions. The first batch of data with transaction t_1 and t_2 are added to a tree in its original order. Note that the last node of each transaction that is added to the tree is a tail-node, such as the node "a" in the transaction {a, d, e, a} and "g" in the transaction {a, c, d, c, g}, the tail-node records the utility of the items in its transaction. Figure 1(a) is the result of the first batch of two transactions added to a UT-tree. Figure 1(b) shows the final UT-tree after the third batch of data is added. When constructing a UT-tree, if a tail-node which is added to the tree has already being a corresponding ordinary-node, we simply convert this ordinary-node to a tail-node. For example, t_5 = {a, c, d}, when the transaction is added into the tree, the same path has already existed after adding the transaction t_2, but in transaction t_5, "d" is the tail-node while it is an ordinary-node in t_2, simply convert the node to a tail-node with the utility of t_5. The global header table which records the *swu* of batch in t[1], t[2], and t[3]. The tail table (Figure 1(c)) records the order of the batch which is used to delete obsolete data, we also add the pointer to the node when the item is appeared more than once in the transaction, it is pointed to the same item which is nearest to it. For example, in transaction {a, c, d, c, g}, the second item "c" points to the first "c" for calculating the utility of same item or sequence in the same transactions.

3.2. Data Updating

When a new batch with two transactions comes, two important processes occur: deleting obsolete data and adding new data.

The algorithm of deleting old data is shown in Algorithm 1 of Figure 2(a). Mainly its work is to process each obsolete tail-node which is recorded in the tail table. Delete the utility in the tail-node (line 2), if there is not any utility in the tail-node, delete the whole path (line 3–10), also delete the item in the tail table (line 11). The algorithm for adding new data is shown in algorithm 2 of Figure 2(a). Transactions are added to the tree (line 3–4), then, the nail-node is stored to the tail table (line 6). The variable y in algorithm 2 is the looping counter.

For UT-tree in Figure 1(b), when the fourth batch with transaction t_7 and t_8 comes, Figure 1(d) is the result after deleting the first batch of data and Figure 1(e) is the final result after updating.

3.3. Mining HUSP Algorithm

The new mining algorithm is proposed in Figure 2(b). Line 1 in the algorithm is to add a numeric list named Utility_cache to each leaf node on the UT-tree T. The elements in this list are the list of utility for each batch of data on the node (shown as "{}" on the tail-node), for example, for the tree of Figure 1(b), each leaf node adds a list of Utility_cache as shown in Figure 3(a). Line 2 in the algorithm is to create a header table H, and the entries and their order are obtained from global header table GH, scanning the tree and placing all the nodes of the same item in the link of H, the link order of the item is corresponding to the order of the batches. H could be obtained in Figure 3(b) based on the UT-tree and global head table of Figure 3(a).

The third to fifteenth lines of the algorithm start from the last item in H and sequentially process each item in H. Lines 4 and 5 calculate *swu* and utility values of the current processing item based on Utility_cache in the path. From the current processing item, the utility of the same item in another transaction can be obtained based

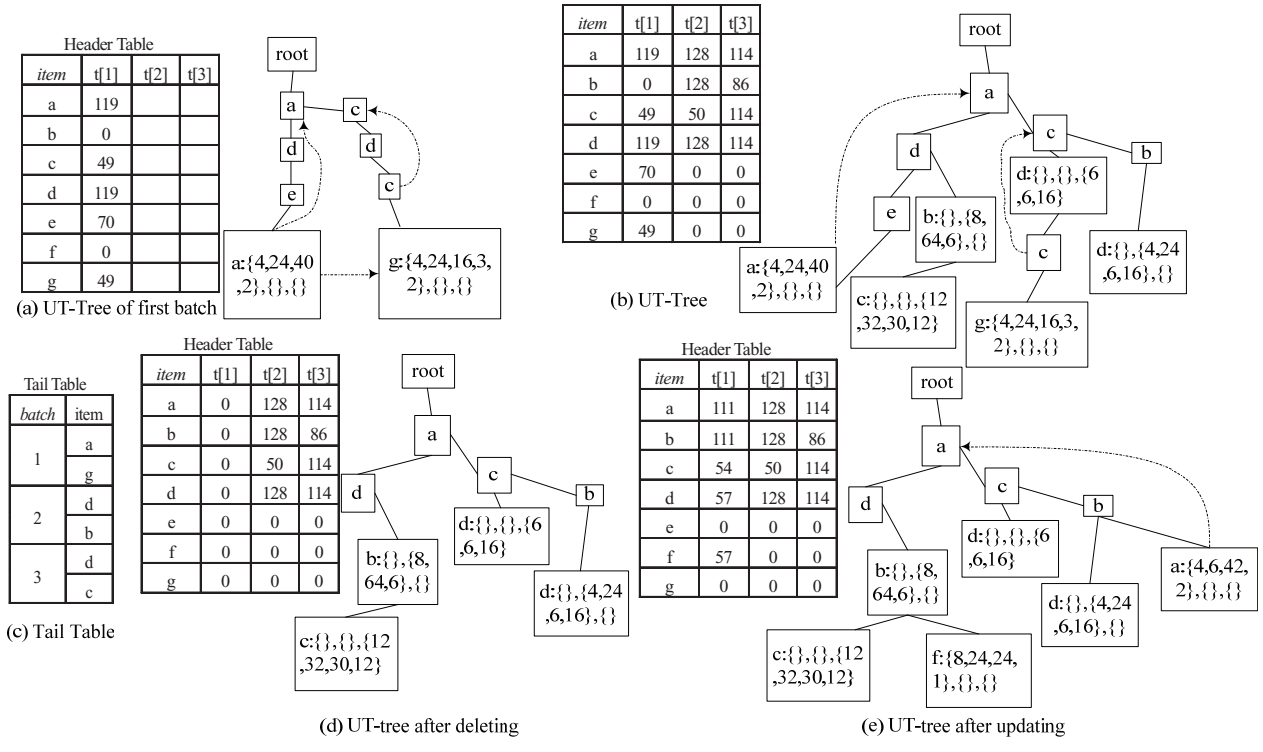


Figure 1 Construction of a UT-Tree.

```
Algorithm 1: deleting old data from UT-tree
Input: A UT-tree T, a tail table G, panel number q
Output: T
Begin
(1) For each tail item Titem in G[q]
(2)   delete the utility in position q;
(3)   if(Titem.utility==NULL&&Titem.Child==NULL)
(4)     TEMP=Titem.parent;
(5)     while(TEMP.utility==NULL&&TEMP!=root)
(6)       delete Titem;
(7)       Titem=TEMP;
(8)       TEMP=TEMP.parent;
(9)     End while
(10)        End if
(11) delete Titem;
(12) End For
End

Algorithm 2:Inserting data into T
Input: A UT-tree T, a tail table G, panel number q, pane size p
Output: T
Begin
(1) x=q%3, y=0;
(2) while(y!=p)
(3)   Tr=transaction form the current location of panel q;
(4)   insert Tr into T;
(5)   Titem =Tr's tail node;
(6)   Save Titem to G[x];
(7)   y=y+1;
(8) End while
(9) End
```

(a) updating data into a UT-tree

```
Procedure: Mining(T, GH, Min_Uti)
Input: A UT-Tree T, a global header table GH,
a minimum utility threshold Min_Uti.
Output: HUSPs (high utility sequential pattern )
(1) Add an attached list (Utility_cache) to T;
(2) Create a header table H for the tree T;
(3) For each item Q1 in H do
(4)     Calculate the swu value of the item Q1 to swu_a;
(5)     Calculate the utility value of the item Q1 to uti_a;
(6)     If(swu_a≥Min_Uti)
(7)       Generate a sequence X= {Q1} ;
(8)       If(uti_a≥Min_Uti
(9)           Copy X into HUSPs;
(10)      End if
(11)      Create a sub-header table Hx for X;
(12)      Create a sub-tree Tx for X;
(13)      Call SubMining(Tx, Hx, X)
(14)    End if
(15) End for
SubProcedure SubaMining (Tx, Hx, X)
(16) For each item Q2 in Hx do
(17)    Calculate swu_a of sequence {Q2+X} ;
(18)    Calculate uti_a of sequence {Q2+X} ;
(19)    If (swu_a≥Min_Uti) then
(20)      Generate a sequence Y= {Q2+X};
(21)    If (uti_a≥Min_Uti) then
(22)      Copy Y into HUSPs;
(23)    End if
(24)    Create a sub-header table Hy for Y;
(25)    Create a sub-Tree Ty for Y;
(26)    SubMining (Ty, Hy, Y);
(27)    End if
(28) End for
```

(b) mining HUSP algorithm

Figure 2 | Algorithms based on UT-tree.

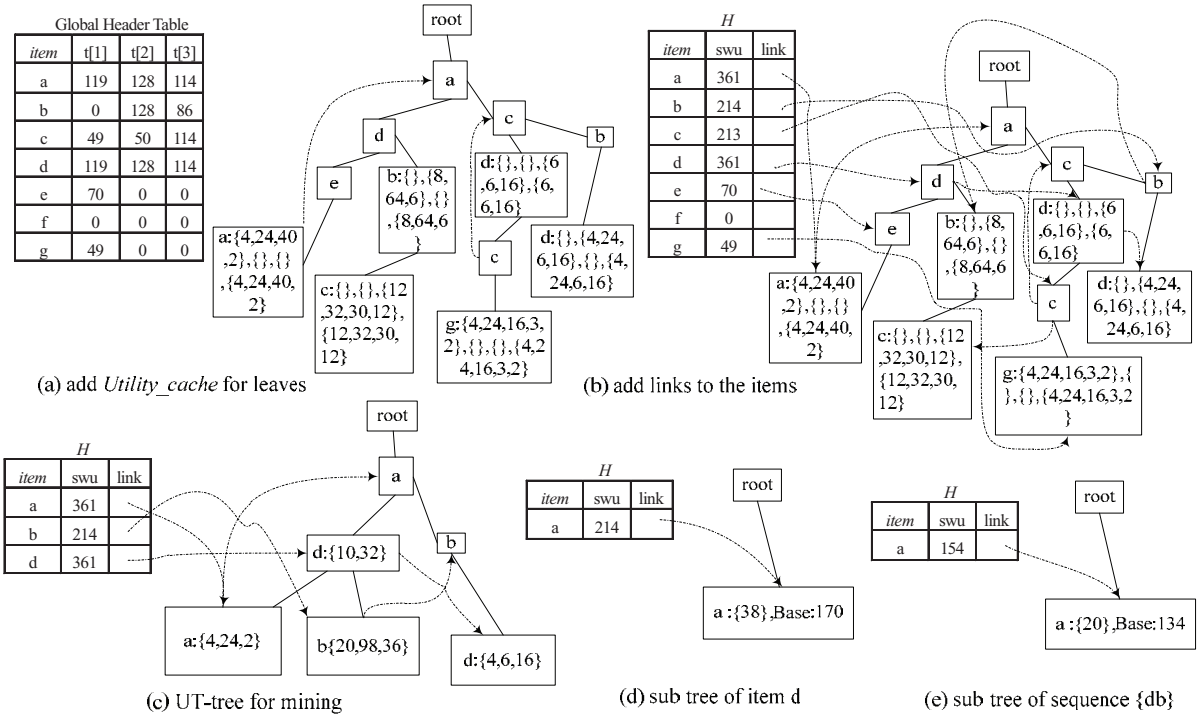


Figure 3 | Process of mining HUSPs from a UT-Tree.

on the links, if the item appears in a path more than once, the utility can be calculated based on definition 1. If the utility value of the current processing item is not less than *MinUti*, the item is an HUSP (sequence with length 1). If the *swu* of the item is not less than *MinUti*, you can create the sub-header table and sub-tree (lines 11 and 12 in the algorithm).

The creation of the sub-header table and sub-tree is described in line 16 and line 25 of Figure 2(b). For example, set *MinUti* = 214, Figure 3(c) is the final UT-tree after deleting the items whose *swu* is less than 214, and the tail-nodes of the tree contain the value of Utility_cache. The sub-tree of item "d" could be known from Figure 3(d), the value of base is corresponding to the processing item. To process the item "d" in *H* of Figure 3(c), there are two paths in the UT-tree. The path "root-a-d" and "root-a-b-d," the *swu* values of all items in these two paths are calculated, and the item whose *swu* is not less than *MinUti* is stored in a sub-header table, we can find that the sequence {ad} satisfies the demand. Figure 3(e) is the sub-tree corresponding to the sequence {db}.

The 26th line of the algorithm deals with the sub-header table and sub-tree. The process method of sub-header table and sub-tree is the same as the processing method of UT-tree. For details, refer to the lines 1–15 in Figure 2(b).

3.4. Algorithm Analysis

The proposed algorithm HUSP-UT (mining HUSPs-based UT-tree) stores the sequence's utility in the leaf node. It guarantees HUSP-UT to get the real utility value of the sequence from the UT-tree instead of estimating value, there is no candidates generating in the mining process, the storage consumption is effectively reduced. You can get the utility value of each item in the correlative sequence when creating sub trees (the creation of the sub-tree in Figure 3) based on the node pointer. This is also guaranteed to be able to calculate the utility of items in the head table and the new *swu* value fast (not contain the utility value of items that have been processed and not in the head table), it reduces the running time efficiently.

It is known that when it is going to create a sub-tree of the item or sequence *X*, the utility of *X* is calculated from the global tree and the sequence which contains X could be found based on the sub-tree. The utility can be calculated from a sub-tree to any item or sequence.

The above is the reason for the efficiency improvement of algorithm HUSP-UT, the experimental results are given in Section 4 of this paper.

Below we prove that our method for finding HUSPs does not miss any HUSPs. Firstly assuming the sequence Z is any HUSP mode, than the *swu* value of any non-empty subset will not be less than the minimum threshold. According to the algorithm HUSP-UT, all items in the sequence Z will appear in the header table corresponding to the UT-tree. When the item Z1 in the sequence Z is processed and the sub-header table is created based on the node pointer, the *swu* value of Z1 combined with the rest item of Z is not less than the minimum threshold. So these combined item will appear in the sub-tree of Z1. When processing the new item of Z1, it is iteratively calculated according to the same creation rules of the sub-tree, it can definitely get that the sequence Z is the HUSP mode and this algorithm HUSP-UT can mine all HUSPs.

4. EXPERIMENTAL ANALYSES

In this section, we evaluate the performance of the proposed algorithm HUSP-UT (mining HUSPs based UT-tree) and compare it with state-of-the-art HUSP-Stream on four datasets: Bible, FIFA, Kosarak10k, and SIGN. These datasets have varied characteristics and represent the main types of data typically encountered in real-life scenarios (dense, sparse, short, and long sequences). The dataset is obtained from SPNF [21] and paper [22]. The characteristics of datasets are shown in Table 3, where |I|, avgLength, and |SD| columns indicate the number of distinct items, the average sequence length, and number of sequences. Bible is moderately dense and contains many medium-length sequences. FIFA is moderately dense and contains many long 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. For all datasets, external utilities of items are generated between 0.01 and 10 by using a lg-normal distribution [23]. All algorithms were written in Java programming language. The configuration of the testing platform is as follows: Windows7 operating system, 2G Memory, intel

Table 3 Data characteristics.

Dataset	|I|	AvgLength	|SD|
Bible	13 905	21.64	36 369
FIFA	13 749	45.32	573 060
Kosarak10k	39 998	11.64	638 811
SIGN	267	93.00	730

(R) Core(TM) i3-2310 CPU@2.10 GHz, Java heap size is 1G. The two methods can mine all HUSPs in the dataset, and we evaluate the time and memory consumption efficiency on the four datasets.

Figure 4 shows the running time comparison of HUSP-UT and HUSP-Stream under four datasets, respectively. The number of data batch is set as $w = 3$ in each window and the batch-size are set as 10K. From the results on Figure 4, we can see that our algorithm HUSP-UT outperforms HUSP-stream on different minimum support thresholds. For example, when the minimum support threshold is 0.01% on the dataset SIGN, HUSP-UT spends 29.56 seconds while HUSP-stream spends 274.521 seconds. The time efficiency is up to 1 order of magnitude faster on the dataset. There will be more HUSPs when the threshold getting smaller, the total running time will increase along with the decrease of the threshold, but we can see that the running time is stable by HUSP-UT and the efficiency is improved by using the new method.

Figure 5 shows the running time comparison for evaluating the accumulated performance of HUSP-UT and HUSP-stream under varied window-width, the minimum utility threshold is set as 0.026%, 0.1%, 0.0174%, and 0.014%, respectively. Each batch-size is 10K, with the increasing of the number of batches in the window, the total running time of HUSP-stream increases more, mainly because it creates more candidates, while the time of HUSP-UT is not very big. The performance advantage of algorithm HUSP-UT accumulates along with window-width, it is stable under varied window-width.

Figure 6 shows that our algorithm HUSP-UT outperforms HUSP-stream on varied batch-sizes. In Figure 6, the minimum utility

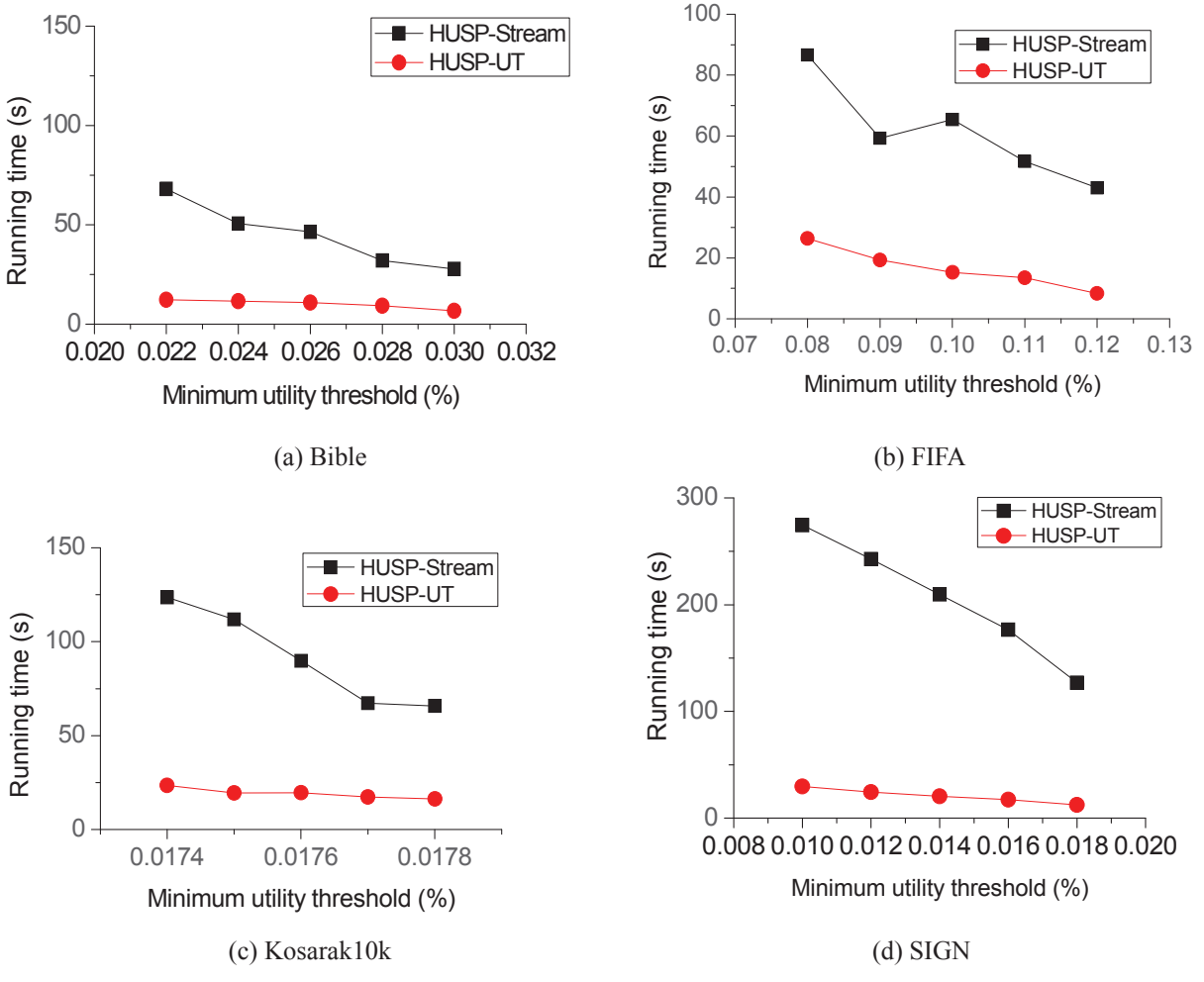
threshold was set to 0.026%, 0.1%, 0.0174%, and 0.014%, respectively. The window size was three batches, and the batch-size test range was 5K–25K. The results of this experiment are the same as the above experiments. The time performance of HUSP-UT is still better than that of HUSP-stream.

We also evaluate the memory usage of the algorithms under different utility thresholds. The number of data batch is set 3 in each window and the batch-size is set as 10K, the results are shown in Figure 7, which indicates our approach consumes less memory than HUSP-stream. For example, for the dataset FIFA, when the threshold is 0.026%, the memory consumption of HUSP-UT is around 200 MB, while that of HUSP-Stream is over 500 MB. A reason is that HUSP-Stream produces too many candidates during the mining process, which causes HUSP-Stream having a larger tree than that of HUSP-UT.

The performance of the algorithms under different window sizes and batch number are also evaluated in this experiment, the minimum utility threshold is set to 0.026%, 0.1%, 0.0176%, and 0.014% for the four datasets, respectively. The results are shown in Figure 8. In Figure 8(a), each bar shows the memory consumption of HUSP-UT on a dataset under a window size when the number of batches is three. For example, the most left bar is the memory consumption of HUSP-UT on Bible when the window size is set to 10K. From Figure 8(a), we observe that the memory consumption of HUSP-UT increases very slowly with increasing window sizes.

Figure 8(b) shows the memory consumption of HUSP-UT under different batch number when the size of a window is 10K, the dataset with different number is displayed with the abscissa. For example, "Bible-3" indicates that there are three batches of transactions in a window with the dataset Bible. We also see that the memory consumption of HUSP-UT increases very slowly with increasing number of batches in a window.

Concluding the above experiments, we can see that our proposed algorithm HUSP-UT has achieved a better performance than HUSP-stream under varied minimum support thresholds, varied window-width, and varied batch-sizes, and its advantage is stable along with the accumulation of the data flow process.



(a) Bible

(b) FIFA

(c) Kosarak10k

(d) SIGN

Figure 4 Execution time under varied minimum utility threshold.

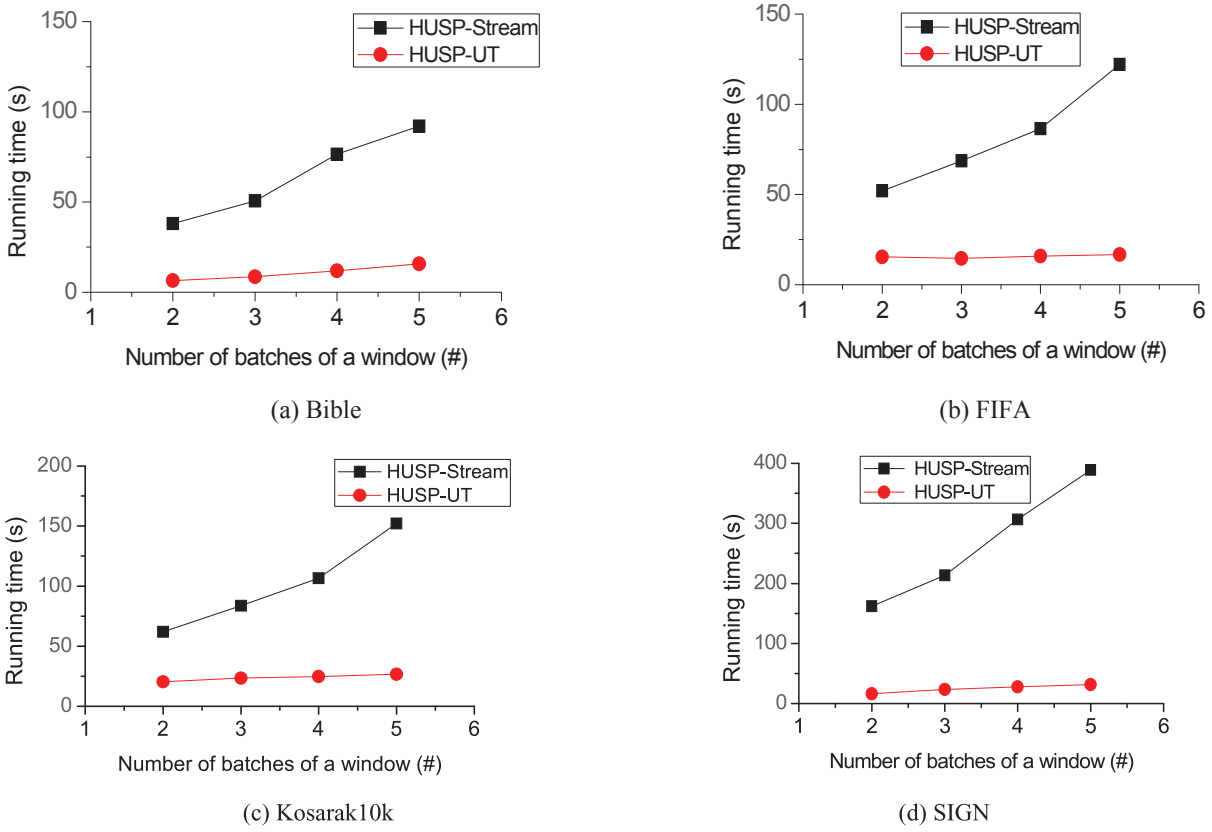


Figure 5 Execution time under varied window-width.

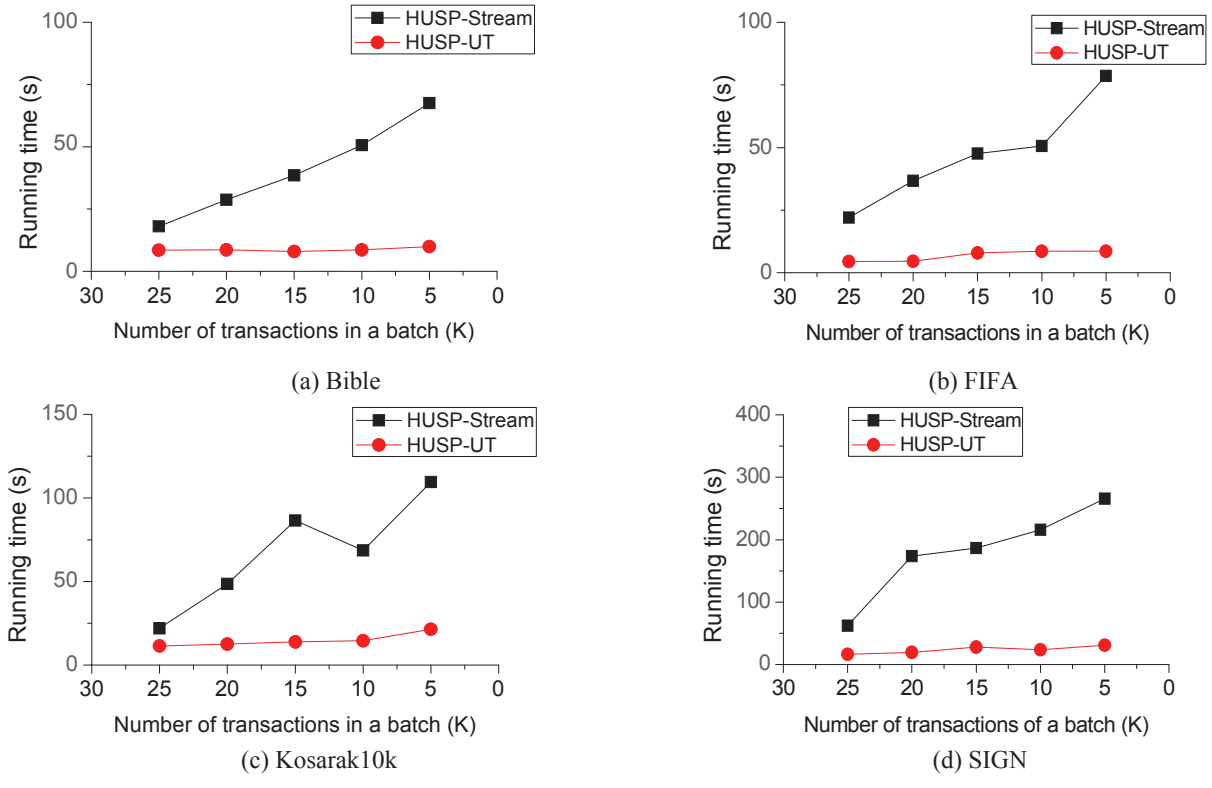


Figure 6 Execution time under varied batch-size.

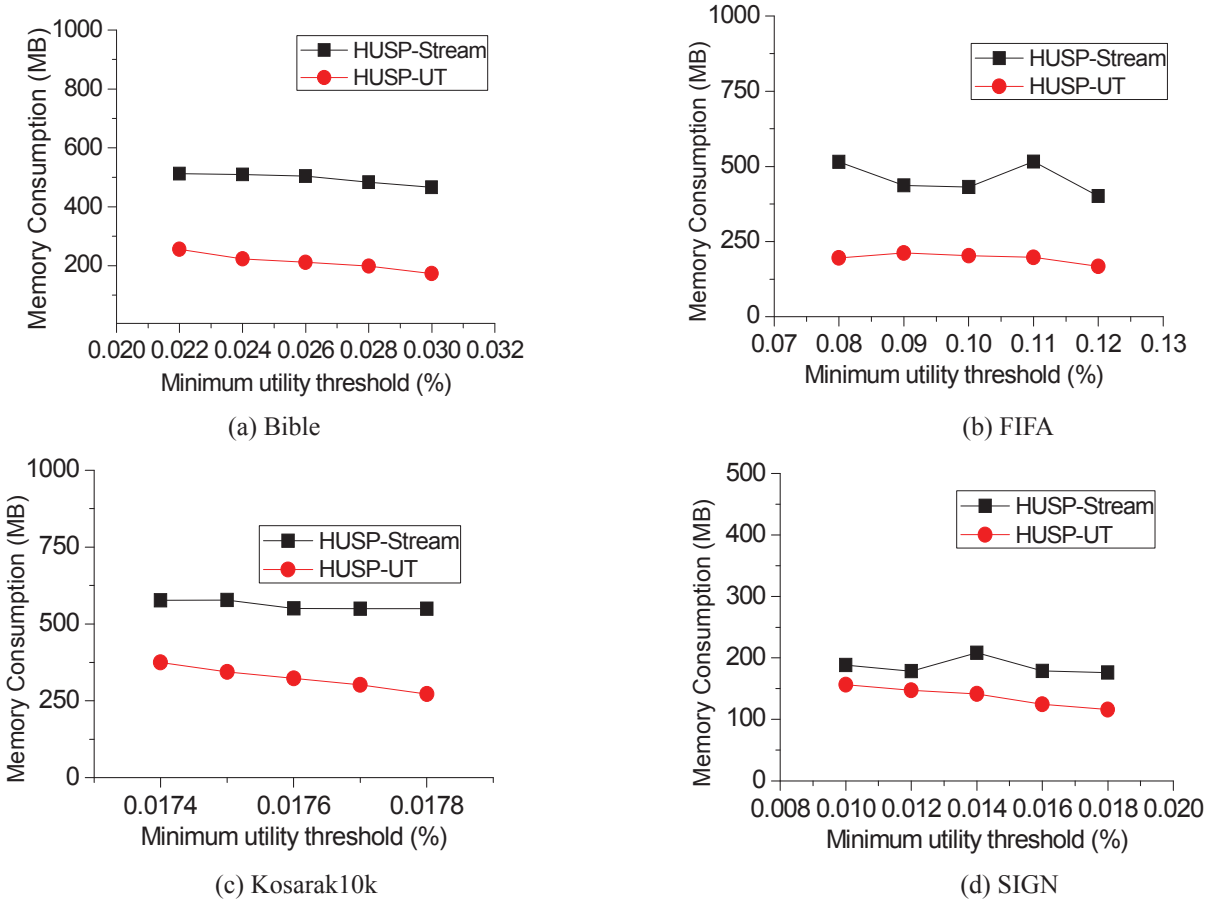


Figure 7 Memory consumption on four datasets.

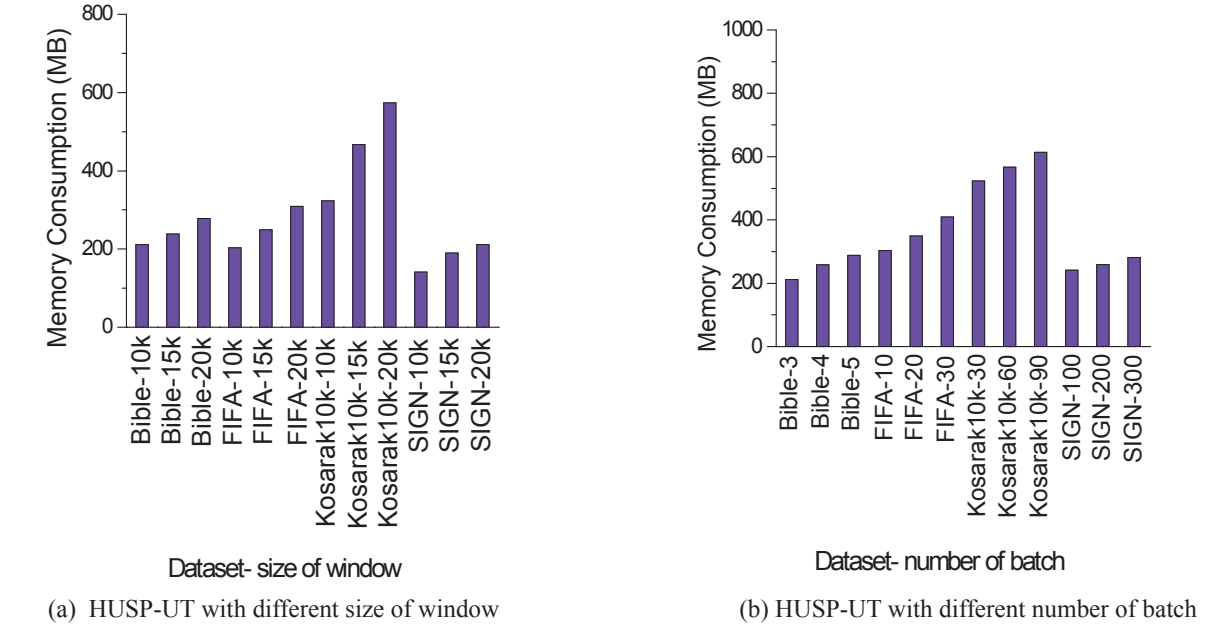


Figure 8 Evaluation of high utility sequential pattern utility on Tail Tree (HUSP-UT).

5. CONCLUSIONS

To improve the overall performance of the high utility sequence mining algorithm over DSs, the efficiency of updating data and mining HUSP should be considered. This study proposes a new data structure UT-tree and gives the corresponding algorithm HUSP-UT. We apply the new method to mine HUSPs over DSs. Theoretical and experimental analysis shows that the performance of our proposed algorithm outperforms the state-of-the-art algorithm HUSP-Stream.

ACKNOWLEDGMENTS

This work is supported by the Project of Zhejiang Provincial Public Welfare Technology Application and Research (LGF19H180002, 2017C35014) and Ningbo Natural Science Foundation (2017A610122).

REFERENCES

[1] J. Pei, J. Han, B. Mortazavi-Asl, PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth, in Proceeding IEEE International Conference on Data Engineering, New Jersey, 2001, pp. 215–552.
[2] M.J. Zaki, SPADE: an efficient algorithm for mining frequent sequences, Mach. Learn. 42 (2001), 31–60.
[3] P.P.C. Rassi, M. Teisseire. Speed: mining maximal sequential patterns over data streams, in Proceeding of the IEEE International Conference on Intelligent Systems, New Jersey, 2006, pp. 546–552.
[4] B. Zhang, C.W. Lin, P. Fournierviger, Mining of high utility-probability sequential patterns from uncertain databases, PLOS ONE. 12(7) (2017), e0180931.
[5] M. Zihayat, Y. Chen, A. An. Memory-adaptive high utility sequential pattern mining over data streams, Mach. Learn. 106 (2017), 799–836.
[6] J.Z. Wang, Z.H. Yang, J.L. Huang, An efficient algorithm for high utility sequential pattern mining, Frontier Innovation Future Comput. Commun. 30(1) (2014), 49–56.
[7] C.F. Ahmed, S.K. Tanbeer, B. Jeong. A novel approach for mining high-utility sequential patterns in sequence databases. Electron. Telecommun. Res. Inst. 32 (2010), 676–686.
[8] J. Yin, Z. Zheng, L. Cao, Uspan: an efficient algorithm for mining high utility sequential patterns, in Proceeding of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, 2012, pp. 660–666.
[9] J.Z. Wang, J.L. Huang, Y.C. Chen, On efficiently mining high utility sequential patterns, Knowl. Info. Syst. 49(2) (2016), 597–627.
[10] A. Marascu, F. Masseglia, Mining sequential patterns from temporal streaming data, Food Chem. 155(28) (2005), 186–191.
[11] M. Zihayat, A. An, Mining top-k high utility patterns over data streams, Info. Sci. 285 (2014), 138–161.
[12] B. Shie, H. Hsiao, V.S. Tseng, Efficient algorithms for discovering high utility user behavior patterns in mobile commerce environments, Knowl. Info. Syst. J. 37(2) (2013), 363–387.
[13] M. Zihayat, C.-W. Wu, A. An, V.S. Tseng, Mining high utility sequential patterns from evolving data streams, in Proceeding of the ASE BigData & Social Informatics, Kaohsiung, Taiwan, 2015, pp. 1–26.
[14] L. Chang, T. Wang, D. Yang, H. Luan, Seqstream: mining closed sequential patterns over stream sliding windows, in Proceeding of the IEEE International Conference on Data Mining, Pisa, 2008, pp. 83–92.
[15] M. Zihayat, C.W. Wu, A. An, Efficiently mining high utility sequential patterns in static and streaming data, Intell. Data Anal. 21 (2017), 103–135.
[16] Y. Wu, Z. Tang, H. Jiang, Approximate pattern matching with gap constraints, J. Info. Sci. 42(5) (2016), 639–658.
[17] W. Le, W. Shui, L. Sheng-Lan, W. Hui-Bing, An algorithm of Mining Sequential pattern with wildcards based on Index-Tree, Chin. J. Comput. 39(17) (2016), 1–9.
[18] Z. Farzanyar, M. Kangavari, N. Cercone, Max-FISM: mining (recently) maximal frequent itemsets over data streams using the sliding window model, Comput. Math. Appl. 64(6) (2012), 1706–1718.
[19] L. Wang, L. Feng, B. Jin, Sliding window-based frequent itemsets mining over data streams using tail pointer table, Int. J. Comput. Intell. Syst. 7(1) (2014), 25–36.
[20] M. Song, S. Rajasekaran, A transaction mapping algorithm for frequent itemsets mining, IEEE Trans. Knowl. Data Eng. 18(4) (2006), 472–481.
[21] P. Fournier-Viger, A. Gomariz, T. Gueniche, SPMF: a Java open source pattern mining library, J. Mach. Learn. Res. 15 (2014), 3389–3393.
[22] S. Zida, P. Fournier-Viger, C.W. Wu, Efficient mining of high-utility sequential rule, in Proceeding International Conference on Machine Learning and Data Mining, San Francisco, 2015, pp. 157–171.
[23] V.S. Tseng, C.W. Wu, B.E. Shie, UP-Growth: an efficient algorithm for high utility itemset mining, in Proceeding International Conference on Knowledge Discovery and Data Mining, Washington, 2010, pp. 253–262.