An Algorithm of Top-*k* High Utility Itemsets Mining over Data Stream

Tianjun Lu
School of Software, Nanyang Institute of Technology, Nanyang, Henan 473000, China
Email: lvtianjun@163.com

Yang Liu
School of Innovation and Experiment, Dalian University of Technology, Liaoning, China 116024.
Email: Dovebaby@mail.dlut.edu.cn

Le Wang
School of Information Engineering, Ningbo Dahongying University, Ningbo, Zhejiang, China 315175.
Email: lelewater@gmail.com

***Abstract*— Existing top-k high utility itemset (HUI) mining algorithms generate candidate itemsets in the mining process; their time & space performance might be severely affected when the dataset is large or contains many long transactions; and when applied to data streams, the performance of corresponding mining algorithm is especially crucial. To address this issue, propose a sliding window based top-k HUIs mining algorithm TOPK-SW; it first stores each batch data of current window as well as the items' utility information to a tree called HUI-Tree, which ensures effective retrieval of utility values without re-scan the dataset, so as to efficiently improve the mining performance. TOPK-SW was tested on 4 classical datasets; results show that TOPK-SW outperforms existing algorithms significantly in both time and space efficiency, especially the time performance improves over 1 order of magnitude.**

***Index Terms*—data stream, high utility itemset, frequent itemset, data mining, top-k**

I. Introduction

High utility itemsets (patterns) mining is an extension of frequent pattern mining, and is becoming a hot topic in data mining [1-11]; its main research focus is on improvement of the space and time efficiency of corresponding algorithms. The algorithm Two-Phase [5] utilizes hierarchical method to generate candidate itemsets; it may generate too many candidates, and needs multiple scans on dataset. The algorithm CTU-Mine[11] utilizes tree structure to improve the mining efficiency, but it only outperforms Two-Phase on dense datasets. The algorithm IHUP[3] scans dataset twice and utilizes pattern growth approach to generate candidate itemsets; the number of candidates is reduced comparing existing algorithms and the mining performance is improved significantly. UP-Growth[1, 2] is an improvement of IHUP to further reduce the number of candidates.

In spite of these research achievements, choosing an appropriate minimum utility threshold is a difficult task for application users: if the threshold is high, there might be no HUI; if the threshold is low, there might result too many HUIs, and the mining performance might be severely affected, even leading to memory overflow. It would also be a time-consuming task if one tries to determine the threshold value through various testing calculations. To address this issue, Wu [10] proposes top-k algorithm, mining the top k itemsets with the highest utility values without presetting the minimum threshold. But this algorithm needs to scan the dataset one more time to calculate the utility values of candidates; and in the case of large dataset or there are too many long-dense transaction itemsets, the performance of Wu's approach is not satisfactory.

Because of the massive, real-timing and dynamic property of data streams, mining algorithms over data streams needs to be more efficient on both running time and memory usage. For the problem of mining top-k HUIs, we propose a sliding window based algorithm TOPK-SW (Top-k HUIs Mining based on Sliding Window) for mining HUIs without generating candidate itemsets. In this approach, transaction itemsets and their effective information are stored to a tree structure; the utility value of each itemset can be retrieved from the tree without generating candidate itemsets or additional scan of the dataset, so as to improve the time and space efficiency of the algorithm significantly.

The **contributions** of this paper are summarized as follows:

(1) We propose a new tree structure named HUI-Tree (High Utility Itemsets Tree) for maintaining a dataset;
(2) We also give an algorithm named TOPK-SW (Top-K high utility itemsets mining based on Sliding-Window) for mining high utility itemsets over data streams;
(3) Both sparse and dense datasets are used in our experiments to compare the performance of the

proposed algorithm against the state-of-the-art algorithms.

The rest of this paper is organized as follows: Section 2 is the description of the problem and definitions; Section 3 describes our algorithm TOPK-SW; Section 4 shows the experimental results; and Section 5 gives the conclusion and discussion.

II. Problem Description and Related Definitions

Given a dataset $DB = \{t_1, t_2, \ldots, t_n\}$ which contains m distinct items $I = \{x_1, x_2, \ldots, x_m\}$ and n transaction itemsets. An itemset X containing k distinct items is called a k-itemset and k is its length. Each transaction itemset t_j is represented as $\{(x_1, c_1)(x_2, c_2)\ldots(x_v, c_v)\}$ (v is the length of t_j), where $\{x_1, x_2, \ldots, x_v\}$ is a subset of I, and c_u ($1 \le u \le v$) is an **internal utility** of item x_u in t_j (denoted as $q(x_u, t_j)$). Each item x in itemset I has a unit profit $p(x)$ and this profit value is denoted as **external utility** of this item. For example, Table 1 is an example of dataset and table 2 shows the profit of each item in Table 1. $|DB|$ represents the number of transaction in dataset DB.

TABLE I.
An Example of Data Stream

TID	Transaction	*tu*
1	(B,4)(C,3)(D,3)(E,1)	28
2	(B,2)(C,2)(E,1)(G,3)	16
3	(B,3)(C,4)(F,1)	17
4	(A,1)(C,6)(D,2)	20
5	(A,2)(C,2)(D,2)(B,2)(E,1))	33
6	(C,3)(D,1)(E,1)(G,2)	10
7	(A,2)(C,6)(D,10)	46
8	(B,2)(C,6)(D,10)	34
…	…	…

TABLE II.
Profit Table (External Utility)

Item	Profit
A	10
B	4
C	1
D	2
E	3
F	1
G	1

Definition 1. The utility of item x in transaction t is denoted as $u(x,t)$ and defined by

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

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

$$u(X,t) = \sum\nolimits_{x \in X \wedge X \subseteq t} u(x,t) \tag{2}$$

Definition 3. The utility of itemset X in transaction database DB is denoted as $u(X)$ and defined by

$$u(X) = \sum\nolimits_{t \in DB \wedge t \supseteq X} u(X,t) \tag{3}$$

Definition 4. The *transaction utility* of transaction t is denoted as $tu(t)$ and defined by

$$tu(t) = \sum\nolimits_{x \in t} u(x,t) \tag{4}$$

Definition 5. The *transaction-weighted-utilization* of an itemset X is denoted as $twu(X)$, and is defined by

$$twu(X) = \sum\nolimits_{t \in DB \wedge t \supseteq X} tu(t) \tag{5}$$

Definition 6. An itemset/item X is called a candidate itemset/item for the high utility itemsets/item if $twu(X) \ge minUti$, and it is also called a *promising itemset/item*, otherwise it is an *unpromising itemset/ item*.

Definition 7. The *support number* (*sn*) of itemset X is the number of transaction containing X in dataset DB.

Theorem 1: *Transaction-weighted downward closure property*: any subset of a promising itemset is a promising itemset and any superset of an unpromising itemset is an unpromising itemset[5].

The window in sliding-window model slides one batch each time, and it only contains a fixed number of batches of the latest data. The problem of mining top-*k* HUIs over data stream based on sliding window model is in fact a problem of mining top-*k* HUIs from the current window; it can be divided into the following two tasks: (1) maintain window data and corresponding utility information, including deleting the oldest batch data and maintain new coming data; (2) mining top-*k* HUIs for the window. For these tasks, we propose algorithm TOPK-SW (Top-*k* HUIs Mining based on Sliding Window); it utilizes a tree structure HUI-Tree to maintain data batches and their corresponding utility information without losing any utility value of the items in the batch, so as to discover top-*k* HUIs without generating candidates itemsets; the tree structure also ensures efficient removal of obsolete batches.

III. Algorithm TOPK-SW

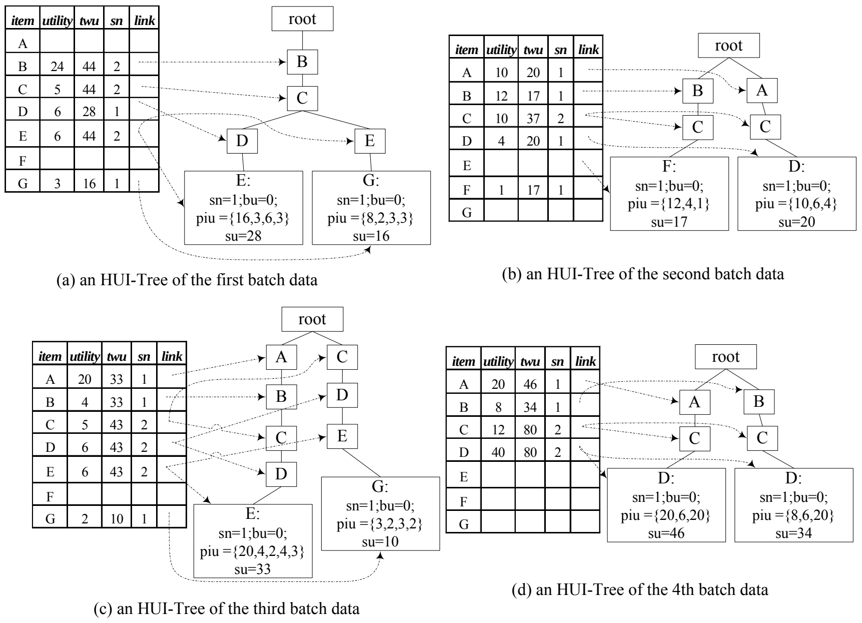
There are two tasks in algorithm TOPK-SW: (1) maintaining data of the current window; (2) mining top-*k* HUIs on a window.

A. Maintaining Window Data

Algorithm TOPK-SW maintains each batch of data to a tree structure called HUI-Tree. Each HUI-tree corresponds to one header table, maintaining the following information of each item on the tree: support number (*sn*), utility value (*utililty*), transaction weighted utilization (*twu*) and links to nodes with the same item name (*link*).

Taking the data in Table 1 and Table 2 for example, we now illustrate the process of maintaining window data of algorithm TOPK-SW. The batch size is set to 2 (p=2), window size (number of batches in a window) is 3 (w=3), and K=3.

Transaction itemsets and their utility information are stored in a certain order (without loss of generality, we use lexicographic order in our example) to an HUI-tree; as shown in Fig. 1(a), the items of the first batch are stored in the tree in lexicographic order. The last node of an itemset is called **tail-node**, in which all information about the same transaction itemset are stored; as an example, for path “root-B-C-D-E” in Fig. 1(a), “E” is a tail-node, and *sn* is the number of transaction itemset {BCDE}, *bu* is the utility value of **base-itemset** (here base-itemset is null, so this value is 0; base-itemset will be discussed in more detail later in the mining algorithm), *piu* is the utility values of items in the transaction itemset, *su* is the sum of *piu* (that is, the total utility value of the same transaction itemset); we call the above data **itemset information** (abbreviated as **itemset-info** hereafter). The utility value of each item in the transaction itemset (*utility*), their transaction weighted utilization (*twu*), support number (*sn*), as well as link to nodes with same name (*link*) are also stored in a header table, see Fig. 1, where the header table corresponds to each tree are illustrated to the left.



item utility twu sn link
A
B 24 44 2
C 5 44 2
D 6 28 1
E 6 44 2
F
G 3 16 1
root B C D E
E: sn=1;bu=0; piu ={16,3,6,3} su=28
G: sn=1;bu=0; piu ={8,2,3,3} su=16
(a) an HUI-Tree of the first batch data
item utility twu sn link
A 10 20 1
B 12 17 1
C 10 37 2
D 4 20 1
E
F 1 17 1
G
root B A C C
F: sn=1;bu=0; piu ={12,4,1} su=17
D: sn=1;bu=0; piu ={10,6,4} su=20
(b) an HUI-Tree of the second batch data
item utility twu sn link
A 20 33 1
B 4 33 1
C 5 43 2
D 6 43 2
E 6 43 2
F
G 2 10 1
root A B C D C D E
E: sn=1;bu=0; piu ={20,4,2,4,3} su=33
G: sn=1;bu=0; piu ={3,2,3,2} su=10
(c) an HUI-Tree of the third batch data
item utility twu sn link
A 20 46 1
B 8 34 1
C 12 80 2
D 40 80 2
E
F
G
root A B C C
D: sn=1;bu=0; piu ={20,6,20} su=46
D: sn=1;bu=0; piu ={8,6,20} su=34
(d) an HUI-Tree of the 4th batch data

Figure 1. The HUI-Tree corresponding to each batch

```
Procedure : Mining
Input: all HUI-Tree Ts and header tables Hs in the current window, an itemset base-itemset (base-itemset is initialized as null), a list TKValueList, minimum utility value minUti.
Output: TKHUIs
Begin
(1)Find top-K maximal total utility values of items in Hs to TKValueList;
(2)Add a field add-information to each leaf-node;

(3)For each item Q in H1 do //from the last item of H1, and H1 is one of Hs
     //Step 1: Calculate utility information of the node Q
(4)        Float twu =0, BU=0, SU=0, NU=0;
(5)        For each header table H in Hs do
(6)            For each node N for the item Q in the tree T corresponding to H do
(7)                BU+= T.N.bu;
(8)                SU+= T.N.su;
(9)                NU+= T.N.nu; //N.nu is a utility for item Q in the list N.piu
(10)           End for
(11)       End for
(12)       twu = BU+SU
     //Step 2: Generate new itemset and create new sub tree and header table
(13)       If (twu⩾minUti) then
(14)           base-itemset = {Q};
(15)           Create a sub HUI-Tree subT and a header table subH for base-itemset ;
(16)           sub-Mining(subT, subH, base-itemset, TKValueList , minUti);
(17)           Remove item Q from itemset base-itemset;
(18)       End if
     //Step 3: Pass add-information on node Q to parent node
(19)       Move each node's bac-info to its parent;
(20)End for

(21)Delete itemsets whose values are less than minUti from TKHUIs;
(22)Return TKHUIs ;
End
```

Figure 2. The algorithm of Mining top-k High Utility itemsets from a window

When a new batch of data comes, delete the tree and header table of the oldest batch and create a new tree for this batch. For example, when the 4th batch in Table 1 comes, delete the tree and header table (Fig. 1(a)) of the first batch, and build a new tree and header table as shown in Fig. 1(d).

When a window is filled with data, a mining for top-*k* HUIs can be performed on this window if requested by the user.

B. Mining top-k High utility Itemsets from a Window

Fig. 2 is the main steps of algorithm TOPK-SW for mining top-*k* HUIs from the HUI-Tree corresponding to the current window.

Line 1 is the initialization of *TKHUIs* and *TKValueList*: if the number of items in the header table is not less than *K*, then store the top *k* items with the biggest utility values to *TKHUIs*, their utility values to *TKValueList*, and assign the smallest value in *TKValueList* to *minUti*; if the number of items in header table is less than *K*, store all items in the header table and their utility values to *TKHUIs* and *TKValueList*, and initialize *minUti* as 0.

Line 2 attaches a backup field **bac-info** for field **itemset-info** to each leaf node, in order to keep valid utility information for the next window in case of any changes made to **itemset-info**.

Line 3-19 process each item in the header table beginning from the last one: (1) re-calculate each item’s *twu* value (*BU*+*SU*); this *twu* value does not include the utility of those items that have already been processed, so its value might be less than the original *twu*. (2) if the new *twu* value is not less than *minUti*, add this item to a **base-itemset** *base-itemset* (which is initialized as *null*), and create sub-tree and sub-header table for *base-itemset*, then continue processing the sub-tree and sub-header table (Fig. 3 is the subroutine of this process); (3) modify the **bak-info** field of the corresponding node: subtract or remove the node's utility value from *piu* and *su* in **bak-info**, and pass it to the parent node: if the parent node contains the **itemset-info** field, then accumulate the corresponding values of **itemset-info** to **bak-info**.

```
Procedure: subMining
Input: A HUI-Tree T, a header table H, an itemset base-itemset, a list TKValueList,
minimum utility value minUti.
Output: TKHUIs
Begin
(1)  For each item Q in H do //from the last item of H
(2)      If( Q.twu >= minUti) then
(3)          float BU=0, SU=0, NU=0;
         //Step 1: Calculate utility information of the node Q
(4)          For each node N for the item Q in T do
(5)              BU+= N.bu;
(6)              SU+= N.su;
(7)              NU+= N.nu; //N.nu is a utility for item Q in the list N.piu
(8)          End for
         //Step 2: Generate new itemset and create new sub tree and header table
(9)          If (BU+SU⩾minUti) then
(10)             base-itemset = base-itemset ∪ {Q};
(11)             Create a sub HUI-Tree subT and a header table subH for base-itemset ;
(12)             If (BU+NU⩾minUti) then
(13)                 Copy base-itemset into TKHUIs ;
(14)                 Add the value BU+NU to TKValueList;
(15)                 If (TKValueList.size() == K+1) then
(16)                         Delete the minimum value of TKValueList;
(17)                         Set the current minimum value of TKValueList to minUti;
(18)                 EndIf
(19)             End if
(20)             subMining(subT, subH, base-itemset, TKValueList , minUti);
(21)             Remove item Q from itemset base-itemset;
(22)         End if
(23)     End if
         //Step 3: Pass add-information on node Q to parent node
(24)     Move each node's itemset-infomation to its parent;
(25) End for

(26) Delete itemsets whose values are less than minUti from TKHUIs;
(27) Return TKHUIs ;
End
```

Figure 3. The Procedure MTKHUIWsub

Line 15 of procedure **Mining** (Fig. 2) creates the sub-header table and sub-tree by scanning the path from the corresponding node to root and **bak-info** of that node; line 11 in procedure **sub-Mining** (Fig. 3) create a sub-header table and sub-tree by scanning the path from the corresponding node to root and **itemset-info** of the node. Here the paths (and the itemset it represents) and the itemset-info of the node is equivalent to a sub-dataset that corresponds to current *base-itemset* (that is, all transaction itemsets in the global dataset that contain this *base-itemset* are mapped to this sub-dataset), so the creation of the sub-header table needs only one pass of scanning of the sub-dataset; the sub-header table only contains those items whose *twu* values are not less than the current *minUti*, and all items are sorted by descending order of their support number. Steps of sub-tree creation: (1) for each transaction itemset in sub-dataset, delete those items that are included in the header table; (2) sort transaction itemsets in the order of the sub-header table; (3) add the processed transaction itemsets to a newly created sub HUI-Tree and store the **itemset-info** to the tail node (in the same manner as adding transaction itemsets of a global dataset to an HUI-tree).

IV. Experimental Results

The core task of sliding window based top-*k* HUIs mining algorithm over data streams is the mining on the data of the current window; and for each window, its data can be viewed as a static dataset; also the proposed algorithm is the first one dealing with data streams, so we can evaluate its efficiency by testing its performance on static dataset (which is equivalent to one window's data). Because the time and space efficiency of algorithm TKU cannot beat UP(Optimal) [10], we only need to compare our proposed algorithm against UP(Optimal). All programs used in this section are coded with Java language.

TABLE III.
DATASET CHARACTERISTICS

Dataset	I	AS	T	DS	Type
Chain-store	46,086	7.2	1,112,949	0.0156%	sparse
T10.I6.D100K	1,000	10	100,000	1%	sparse
Mushroom	119	23	8,124	19.33%	dense
Connect	129	43	67,557	33.33%	dense

Classical datasets are used in our experiments, including real and synthetic datasets, see Table 3. Chain-store [12] is a real dataset of a chain supermarket in California; T10.I6.D100K is created by IBM Quest Data Generator [13]; the other 2 datasets can be downloaded from FIMI website.

Dataset T10.I6.D100K, Mushroom and Connect do not contain internal & external utility values. Adopting the approaches in reference [3, 5, 11, 14], the number of a certain item in a transaction (internal utility) is generated as a random integer between 10 and 0; item's profit (external utility) is also generated randomly as a value between 0.0100 and 10.0000; and because in the real world, the number of items with high profit is relatively fewer, we apply a logarithm normal distribution on the generated external utility values. Table 3 is the characteristics of the datasets, where "I" is number of unique items in the dataset, "AS" is the average length of transaction itemsets, "T" is the number of transaction itemsets in the dataset, and "DS" is the density of data (larger value means denser, and smaller value means sparser). A dataset with density less the 10% is called a sparse dataset, otherwise it is a dense dataset [15].

Fig. 4-7 is the experimental result on 4 datasets respectively; in these figures, part (a) is the running time, part (b) is the number of generated patterns. TOPK-SW can obtain the patterns' utility value when they are generated, while UP(Optimal) needs an additional scan of dataset to calculate the utility values and to confirm the top *K* patterns; so the more patterns UP(Optimal) generates, the less time- & space-efficient it will be; as show in Fig. 4-7(a), the bigger the *K* value is, the more itemsets need to be generated, and the less efficient UP(Optimal) will be. On *Connect*, UP (Optimal) overflows when *K* is greater than 100.

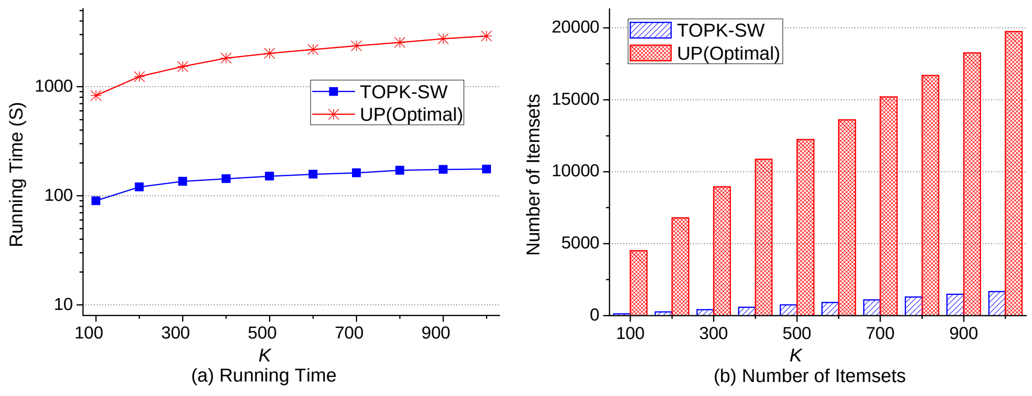


Figure 4. The dataset Chain-store

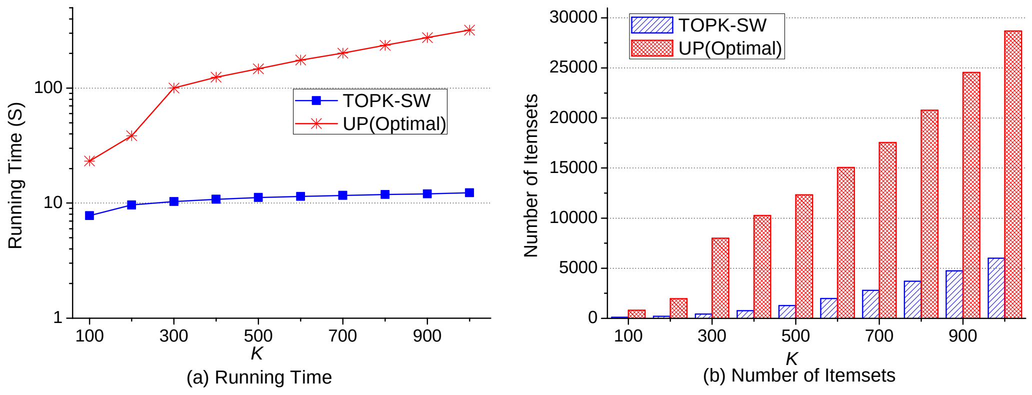


Figure 5. The dataset T10.I6.D100k

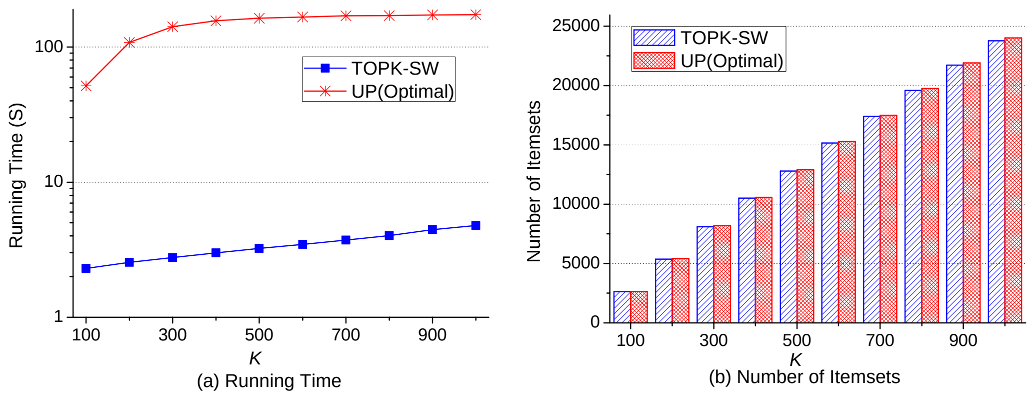


Figure 6. The dataset Mushroom

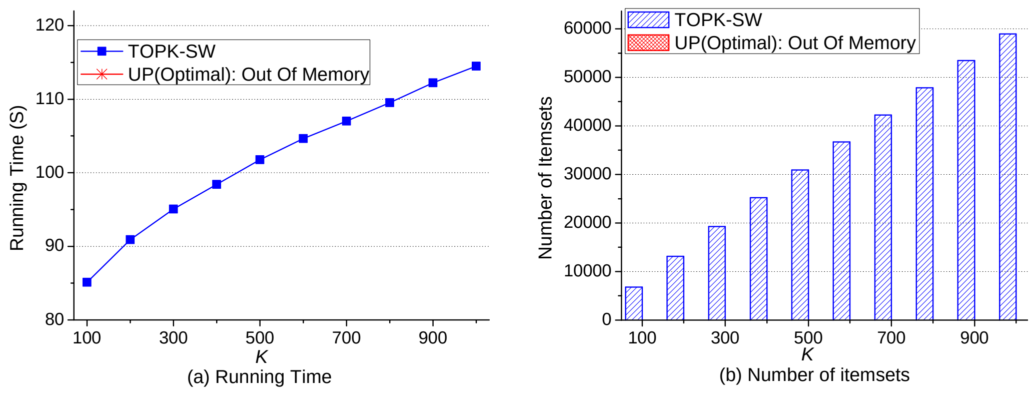


Figure 7. The dataset Connect

Because TOPK-SW maintains utility information of transaction itemsets to trees and ensures their efficient retrieval, and the newly calculated *twu* values are smaller than that in UP (Optimal), there might be cases when TOPK-SW does not need to create more sub-trees and generate new patterns for *base-itemset* while UP (Optimal) must, UP (Optimal) will generate more patterns, as shown in Fig. 4-7(b); because dataset *Connect* is a dense dataset with long transaction itemsets, UP (Optimal) overflows when *K* is greater than 100.

We can see from the above experimental results and analysis, the performance of the proposed algorithm TOPK-SW is improved significantly, especially its time efficiency improves over 1 order of magnitude; on dense & long transaction dataset, TOPK-SW performs well while UP (Optimal) overflows easily; the time performance of TOPK-SW is also stable along with the variance of the *K* value.

V. Conclusion and Discussion

We propose a top-*k* HUIs mining algorithm TOPK-SW for data streams based on sliding window model; transaction itemsets in a batch and their utility information are maintained to a tree, and in the mining process the utility of an itemset can be efficiently retrieved from this tree without additional scan of the dataset, so as to improve the mining efficiency significantly. 4 classical datasets are used in our experiments, including real and synthetic datasets. The results show that not only the time performance of TOPK-SW is improved significantly (over 1 order of magnitude), its space performance is also remarkable, especially for dense & long-transaction dataset, our algorithm discovers resulting patterns efficiently while the existing algorithm overflows.

References

[1] V.S. Tseng, B. Shie, C. Wu, and P.S. Yu, "Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases," IEEE Transactions on Knowledge and Data Engineering, 2012(PrePrint).

[2] V.S. Tseng, C.W. Wu, B.E. Shie, and P.S. Yu. "UP-Growth: An efficient algorithm for high utility itemset mining," ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2010, Washington, DC, United states.

[3] C.F. Ahmed, S.K. Tanbeer, B.S. Jeong, and Y.K. Lee, "Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases," IEEE Transactions on Knowledge and Data Engineering, 2009, 21(12), pp. 1708-1721.

[4] H. Yao and H.J. Hamilton, "Mining itemset utilities from transaction databases," Data and Knowledge Engineering, 2006, 59(3), pp. 603-626.

[5] Y. Liu, W.K. Liao and A. Choudhary, eds. A two-phase algorithm for fast discovery of high utility itemsets. Advances in Knowledge Discovery and Data Mining. Vol. 9th Pacific-Asia conference on Advances in Knowledge Discovery and Data Mining. 2005, Berlin: Springer: Hanoi, Viet nam. 689-695.

[6] H. Li, H. Huang, Y. Chen, Y. Liu, and S. Lee. "Fast and memory efficient mining of high utility itemsets in data streams," 8th IEEE International Conference on Data Mining, 2008.

[7] J. Liu, K. Wang and B. Fung. "Direct Discovery of High Utility Itemsets without Candidate Generation," 2012 IEEE 12th International Conference on Data Mining, 2012.

[8] C.W. Lin, T.P. Hong, G.C. Lan, J.W. Wong, and W.Y. Lin, Mining High Utility Itemsets Based on the Pre-large Concept, in Advances in Intelligent Systems and Applications-Volume 1. 2013, Springer. pp. 243-250.

[9] M. Liu and J. Qu. "Mining high utility itemsets without candidate generation," 21st ACM International Conference on Information and Knowledge Management, 2012, Maui, HI, United states.

[10] C.W. Wu, B. Shie, V.S. Tseng, and P.S. Yu. "Mining top-K high utility itemsets," 18th ACM SIGKDD international conference on Knowledge discovery and data mining, 2012.

[11] A. Erwin, R.P. Gopalan and N.R. Achuthan. "CTU-mine: An efficient high utility itemset mining algorithm using the pattern growth approach," 7th IEEE International Conference on Computer and Information Technology, 2007.

[12] J. Pisharath, et al. NU-MineBench Version 2.0 Scorce Code and Datasets, http://cucis.ece.northwestern.edu/projects/DMS /MineBench.html. Accessed July 2011.

[13] IBM Data Generator, http://www.almaden.ibm.com/software /quest/Resources/index.shtml. Accessed Dec. 2010.

[14] Y.C. Li, J.S. Yeh and C.C. Chang, "Isolated items discarding strategy for discovering high utility itemsets," Data and Knowledge Engineering, 2008, 64(1), pp. 198-217.

[15] F.Y. Ye, J.D. Wang and B.L. Shao. "New algorithm for mining frequent itemsets in sparse database," International Conference on Machine Learning and Cybernetics, 2005, Guangzhou, China.