UDS-FIM: An Efficient Algorithm of Frequent Itemsets Mining over Uncertain Transaction Data Streams

Le Wang [a,b,c], Lin Feng[b,c, *], and Mingfei Wu [b,c]
[a] College of Information Engineering, Ningbo Dahongying University, Ningbo, Zhejiang, China 315175.
[b] School of Computer Science and Technology, Faculty of Electronic Information and Electrical Engineering, Dalian University of Technology, Dalian, Liaoning, China 116024.
[c] School of Innovation and Experiment, Dalian University of Technology, Liaoning, China 116024.
lelewater@gmail.com; fenglin@dlut.edu.cn;merphy.wmf@gmail.com

***Abstract*—In this paper, we study the problem of finding frequent itemsets from uncertain data streams. To the best of our knowledge, the existing algorithms cannot compress transaction itemsets to a tree as compact as the classical FP-Tree, thus they need much time and memory space to process the tree. To address this issue, we propose an algorithm UDS-FIM and a tree structure UDS-Tree. Firstly, UDS-FIM maintains probability values of each transactions to an array; secondly, compresses each transaction to a UDS-Tree in the same manner as an FP-Tree (so it is as compact as an FP-Tree) and maintains index of probability values of each transaction in the array to the corresponding tail-nodes; lastly, it mines frequent itemsets from the UDS-Tree without additional scan of transactions. The experimental results show that UDS-FIM has achieved a good performance under different experimental conditions in terms of runtime and memory consumption.**

***Index Terms*—frequent itemset, frequent pattern, uncertain dataset, data streams, data mining**

I. Introduction

In recent years, frequent itemsets mining (FIM) on uncertain datasets [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16] has been a topic in data mining with the emergence of uncertain datasets in many applications. For example, the locations of moving objects obtained through RFID or GPS are not precise; data obtained from wireless sensors are not precise. Table 1 shows an example of uncertain transaction dataset, each transaction of which represents that a customer might buy a certain product with a probability. For instance, the first transaction T_1 shows that a customer *A* might purchase products "C", "D" and "E" with 80%, 85% and 75% chances in near future. These probability values may be obtained by analysing the users' shopping records; for example, the data of transaction T_1 may be obtained by

TABLE I.
An Example of Uncertain Transaction Dataset

TID	Transaction itemset
T_1	(C, 0.8), (D, 0.85), (E, 0.75)
T_2	(B, 0.9), (C, 0.8) , (D, 0.6), (F, 0.2)
T_3	(A, 0.15), (B, 0.8) ,(D, 0.4)
T_4	(B, 0.85), (C, 0.6), (D, 0.7)
T_5	(A, 0.6), (B, 0.25), (C, 0.3), (E, 0.5)
T_6	(A, 0.7) , (B, 0.8), (D, 0.25)
T_7	(A, 0.65), (B, 0.7) , (C, 0.5)
T_8	(C, 0.5), (D, 0.65) , (F, 0.6)
T_9	(A, 0.6), (B, 0.82), (C, 0.63)

analyzing customer *A*'s shopping history statistically and applying the probability of purchasing a certain product as the corresponding probability values of that item.

There have been many studies aimed at FIM on precise static datasets and data streams, such as Apriori [17], FP-Growth [18], H-Mine [19], DST [20], CPS [21], FP-Streaming [22], etc. However, these existing algorithms cannot mine frequent itemsets from uncertain transaction datasets. In the past few years, several algorithms [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16] have been proposed for FIM on uncertain transaction datasets, among which [10, 11, 14] are for data streams and the others are for static datasets. The algorithms in [10, 11, 14] maintain the transaction itemsets information to a UF-Tree [8]; however a UF-Tree is not as compact as the original FP-Tree [18] and cannot efficiently maintain transaction itemsets in terms of memory space. Thus these algorithms require a large amount of computational time and memory space to process tree nodes.

In this paper, we propose a tree structure called UDS-Tree for maintaining uncertain transaction itemsets information. We also give a corresponding algorithm called UDS-FIM for mining frequent itemsets from the UDS-Tree without additional scan of transaction datasets.

This work was supported by National Natural Science Foundation of P.R. China (61173163, 51105052) and Liaoning Provincial Natural Science Foundation of China (Grant No. 201102037)
Corresponding Author: Lin Feng; E-mail: lelewater@gmail.com

UDS-FIM employs the sliding window model to process data streams. UDS-Tree is a tree as compact as the original FP-Tree. Meanwhile, UDS-FIM removes the obsolete data from the UDS-Tree through scanning a part of the UDS-Tree, instead of scanning the whole tree. The experimental results show that the proposed algorithm has a good performance.

The key contributions of this paper include:

- We propose a new tree structure named UDS-Tree for maintaining uncertain transaction itemsets information;
- We also give an algorithm named UDS-FIM for FIM over uncertain transaction data streams;
- Both sparse and dense datasets, including real-world and synthetic datasets, are used in our experiments to test the performance of the proposed algorithm.

The rest of this paper is organized as follows: Section 2 is the description of the problem and definitions; Section 3 describes related works; Section 4 describes our algorithm UDS-FIM; Section 5 shows the experimental results; Section 6 gives the conclusions.

II. Problem definitions

In this section, we provide background information about FIM on uncertain transaction data streams. Assume a uncertain transaction data stream *UDS* (i.e. $UDS = \{T_1, T_2, \ldots, T_n, \ldots\}$) contains *m* distinct items (i.e. $I = \{i_1, i_2, \ldots, i_m\}$), and each transaction itemset *t* ($t \in UDS$) is represented as $\{x_1, p_1, x_2, p_2, \ldots, x_v, p_v\}$, where $\{x_1, x_2, \ldots, x_v\}$ is a subset of *I* and the decimal value p_u ($1 \le u \le v$) is called the existential probability of item x_u in the transaction itemset *t* (denoted as $p(x_u, t)$). An itemset containing *k* distinct items is called a *k*-itemset, and its length is *k*.

Definition 1 ([17]**).** The *support number* (*SN*) of an itemset *X* is the number of transaction itemsets containing *X* in a dataset.

Definition 2 ([8, 13]**).** The probability of an itemset *X* in a transaction itemset *t* is denoted as $p(X, t)$, and is defined by

$$p(X,t) = \prod\nolimits_{x \in X, x \in t} p(x,t).$$

Definition 3([8, 13]**).** The *expected support number* (*expSN*) of an itemset *X* in an uncertain transaction dataset is the sum of its probability values in all transaction itemsets containing *X*, denoted as *expSN*(*X*), and is defined by

$$expSN(X) = \sum\nolimits_{t} p(X,t).$$

Definition 4 ([8, 13]**).** The *minimum expected support threshold minExp* is a predefined percentage of the total number of transactions (which is denoted as *n*), and then the *minimum expected support number* (*minExpSN*) is defined by

$$minExpSN = n \times minExp.$$

Assume a sliding window contains *w* batches of data, and each batch of data contains *p* transaction itemsets (i.e., a window contains *w*p* transaction itemsets).

Definition 5. Thus, in the sliding window, the *minimum expected support number* (*minExpSN*) is defined by

$$minExpSN = w \times p \times minExp.$$

Definition 6. In a sliding window, an itemset *X* is called frequent itemset if its expected support number in the window is not less than *minExpSN*.

Frequent itemsets mining on uncertain data streams has two important processes: (1) Updating new data to the tree and removing the obsolete data from the tree; (2) Mining all frequent itemsets of the current sliding window according to the request of user.

III. Related Work

A. Frequent Itemsets Mining over Static Datasets

FP-Growth [18] is a classical pattern-growth algorithm for traditional FIM over precise datasets. Since the publication of FP-Growth, many well-known algorithms have been developed based on it to get better performance, such as FP-Streaming [22], UF-Growth [8] and SUF-Growth [14]; the UDS-FIM algorithm proposed in this paper is also based on FP-Growth for fast FIM on uncertain data streams.

FP-Growth utilizes a 2-step approach for this job: firstly, it finds all frequent 1-itemsets under the condition of *k*-itemset *X* ($k \geq 1$); secondly, it generates frequent (k+1)-itemsets using those frequent 1-itemsets and *X*. It maintains the transaction itemsets to a FP-Tree, thus it finds all frequent 1-itemsets under the condition of *X* by scanning the FP-Tree. FP-Tree is created with the following rules: (1) itemsets are rearranged in descending order of support numbers of items, and then are added to a FP-Tree; (2) itemsets will share the same node when the corresponding items are same.

The algorithms U-Apriori [13] and UF-Growth [8] are two representatives for FIM on static uncertain datasets. The algorithm U-Apriori is based on the algorithm Apriori [17] which is the first algorithm for FIM on precise transaction datasets and employs the level-wise method. It starts with finding all frequent 1-itemsets with one scan of dataset. Then in each iteration, it first generates candidate (k+1)-itemsets using frequent *k*-itemsets ($k \geq 1$), and then identifies real frequent (k+1)-itemsets from candidates with one scan of dataset. The iteration goes on until there is no new candidate. U-Apriori has a disadvantage: it generates candidates and requires multiple scans of datasets, so its time performance may become worse with the increase of the number of long transaction itemsets or decrease of the minimum expected support threshold. In 2011, Wang *et al.* [7] proposed the algorithm MBP based on Apriori for FIM on uncertain datasets. The authors proposed one strategy to speed up the calculation of the expected support number of a candidate itemset: MBP stops calculating the expected support number of a candidate itemset if the itemset can be determined to be frequent or infrequent in advance. MBP has a better performance than U-Apriori.

The algorithm UF-Growth [8] is based on FP-Growth [18] for FIM on uncertain datasets. It employs the pattern-growth method with 2 scans of the dataset. In the first scan, it first finds all frequent 1-itemsets, arranges the frequent 1-itemsets in descending order of support numbers, and then maintains them in a header table. In the second scan, it first removes infrequent items from each transaction itemset, rearranges the remaining items of each transaction itemset in the order of the header table, and then adds the sorted itemset to a tree structure called UF-Tree. After the UF-Tree is created, UF-Growth can recursively create sub header tables and prefix (conditional) UF-Trees in the same manner as the FP-Growth algorithm.

Although UF-Growth needs just two scans of dataset, it still has a disadvantage: it needs a large amount of memory to store UF-Tree because it only merges nodes with the same item and the same probability, when transaction itemsets are added to the UF-Tree. For example, when these two transaction itemsets {A, 0.53, B, 0.70, C, 0.23} and {A, 0.55, B, 0.80, C, 0.23} are added to a UF-Tree by lexicographic order, they will not share the node "A" because the probabilities of item "A" are not equal in these two transactions. To overcome this issue, Leung *et al.* [4] proposed an approximate algorithm based on the algorithm UF-Growth. This approximate algorithm considers that the items with the same k-digit value after the decimal point have the same probability. For example, when these two transactions {A, 0.53, B, 0.70, C, 0.23} and {A, 0.55, B, 0.80, C, 0.23} are added to a UF-Tree by lexicographic order, they will share the node "A" if k is set as 1 because both probabilities of item "A" are considered to be 0.5; if k is set as 2, they will not share the node "A" because the probabilities of "A" in these two transaction are 0.53 and 0.55 respectively. The smaller k is set, the less memory the approximate algorithm requires. However, it still cannot build a tree as compact as the original FP-Tree; moreover, it may lose some frequent itemsets.

The algorithm CUFP-Mine [1] outperforms the U-Apriori and UF-Growth when a dataset just is sparse and the predefined threshold is high. It requires a large amount of computational time and memory when transaction itemsets are not short and the threshold is small.

B. Frequent Itemsets Mining over Data Streams

Judging by the mechanism of the window algorithm, three window models can be applied in FIM: sliding window model, landmark window model and damped window model. The window size in the sliding window model is fixed and is specified by user in advance. In the landmark window model, the start point of the window is fixed and is specified by user. The damped window model considers the current data more valuable than old data, so it assigns a weight value for each batch of data or each transaction, and the weight value decreases over time. How to choose the window model depends on the user's interest of the data: the damped window model is employed if the user is interested in historical data and more interested in current data than historical data; the sliding window model is employed if the user is interested in a current fixed-size data. The commonly used approach of mining continuous data streams is the *sliding window model* [2, 14, 20, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32].

Several algorithms base their approaches on the algorithm FP-Streaming [22] which is for FIM over *precise* transaction data streams. FP-Streaming requires two parameters *PreMinsup* and *Minsup* (*PreMinsup* ≤ *Minsup*) that are the user specified minimum support numbers for a batch of data and a window data respectively. The algorithm first finds pre-frequent itemsets whose support numbers are not less than the value *PreMinsup* from each batch in the current window, and then it finds frequent itemsets (whose support numbers are not less than *Minsup*) from all pre-frequent itemsets of all batches in the current window. The FP-Streaming algorithm may lose some frequent itemsets and cannot handle uncertain data.

The algorithms proposed in papers [10, 11, 14] are for FIM over *uncertain* data streams. Based on the algorithms FP-Streaming and UF-Growth, Leung *et al.* [14] proposed two algorithms UF-Streaming and SUF-Growth. UF-Streaming employs the same method as the FP-Streaming: mining frequent itemsets using two minimum support numbers *PreMinsup* and *Minsup*. Thus UF-Streaming may lose some frequent itemsets and it also requires an extra data structure UF-Stream to maintain the pre-frequent itemsets of the current window. SUF-Growth is a precise algorithm and uses only a user specified minimum support number *Minsup*, it directly finds all frequent itemsets with support number that is not less than *Minsup* from a window, and does not lose any frequent itemsets. The algorithm SUF-Growth maintains data of a window to a UF-Tree by the following two rules: (1) each tree node of its UF-Tree contains two probability values and w support numbers; (2) when a new batch of data is coming, it first removes the obsolete batch of data from the tree and then adds new batch of data to it. If a user requests the frequent itemsets of the current window, it performs the frequent itemsets mining. Similar to the algorithm UF-Growth, the main weakness of SUF-Growth is that it requires a large amount of memory to store UF-Tree, thus it costs much memory and time to process the UF-Tree. The proposed algorithms in the paper [10, 11] employ the damped window model, but they still use the UF-Tree to maintain uncertain transaction itemsets information.

IV. The Proposed Method

The proposed algorithm UDS-FIM mainly consists of three procedures: (1) creating a global UDS-Tree; (2) mining frequent itemsets from the global UDS-Tree; (3) removing the obsolete data from the global UDS-Tree. We describe some structures used by UDS-FIM in Section 4.1, give an example of the construction of a UDS-Tree in Section 4.2, elaborate the algorithm UDS-FIM with an example in Section 4.3, and discuss the method of removing obsolete data from the current window in Section 4.4.

A. Structure of a UDS-Tree

Definition 7. Let itemset $X = \{x_1, x_2, x_3, \ldots, x_u\}$ be a sorted itemset, and the item x_u is called *tail-item* of X. When the itemset X is added into a tree T in accordance with its order, the node on the tree that represents this tail-item is called as a *tail-node*; a node that has no children is called as a *leaf node*; a node that is neither a tail-node nor a leaf-node is called as a *normal node*.

Before a transaction itemset is added into a UDS-Tree, its corresponding probability values are appended to an array (we call it TPA, Transaction Probability Array). For example, Figure 2(a) shows a TPA which maintains the first 2 transaction probability values in Table 1; each element of TPA is called a sub-TPA.

The proposed algorithm UDS-FIM needs two kinds of tree structures: (1) global UDS-Tree, which maintains the transaction itemsets of data streams; (2) prefix UDS-Tree, which is used to maintain transaction itemsets having the shared prefix itemset.

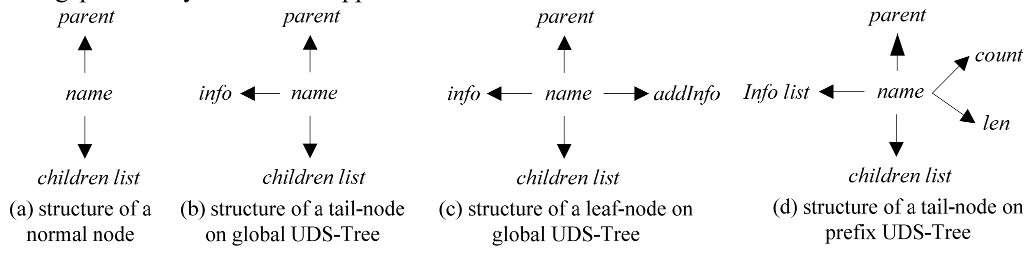

parent
name
children list
(a) structure of a normal node
parent
info
name
children list
(b) structure of a tail-node on global UDS-Tree
parent
info
name
addInfo
children list
(c) structure of a leaf-node on global UDS-Tree
parent
Info list
name
count
len
children list
(d) structure of a tail-node on prefix UDS-Tree

Figure 1. The structures of nodes on a UDS-Tree

The global UDS-Tree contains three kinds of nodes: normal node, tail-node and leaf-node. The structure of nodes is illustrated in Figure 1, where *name* is the item name of each node, *parent* represents the parent node of each node, and *children list* is a list of all the children node of a node. Figure 1(a) shows the structure of a normal node. Figure 1(b) and (c) show the structures of a tail-node and a leaf-node respectively, where field *info* and field *addInfo* include 3 sub-fields:

(1) ***count***: the support number of the node;

(2) ***len***: the length of itemsets;

(3) ***pro_ind***: *w* lists, containing the indexes of TPA of *w* batch data respectively (*w* is the width of the sliding window).

The field *addInfo* is only used when we need to mine the frequent itemsets of a current window, and is removed from the tree after the mining process.

Definition 8. Let T be a prefix tree of the itemset Y. When an itemset X containing itemset Y is added to the tree T, the probability of itemset Y in itemset X, $P(Y, X)$, is defined as the ***base probability*** of itemset X on the tree T.

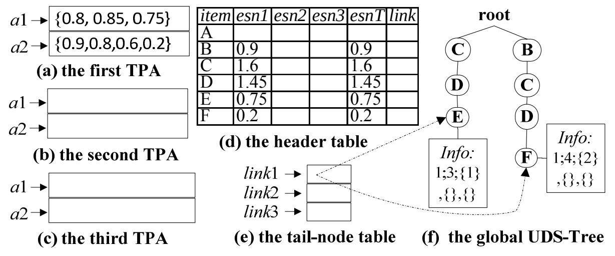
The prefix UDS-Tree contains two kinds of nodes: normal node and tail-node. The structure of normal node is as same as that of a global UDS-Tree, as shown in Figure 1(a). The structure of tail-node is shown in Figure 1(d), where *info list* is a list of *info* and the *info* includes 3 sub-fields:

(1) ***bp***: *w* lists maintaining *base probability* values of *w* batch data respectively (*w* represents the width of the sliding window);

(2) ***pro_ind***: *w* lists maintaining the indexes of each TPA of *w* batch data respectively;

(3) ***item_ind***: a list maintaining the indexes of all items of a path in sub-TPA.

B. Construction of a Global UDS-Tree



item	*esn1*	*esn2*	*esn3*	*esnT*	*link*
A					
B	0.9			0.9	
C	1.6			1.6	
D	1.45			1.45	
E	0.75			0.75	
F	0.2			0.2	

Figure 2. Processing the first batch of data

We illustrate the construction of a global UDS-Tree using the transaction itemsets in Table 1. Assume each window contains 3 batches of data, and each batch contains 2 transaction itemsets.

Before construction the UDS-Tree with the transaction itemsets, first create the following 4 data structures and initialize them as null:

(1) TPA: we need 3 TPAs for maintaining the transaction probability values of 3 batch of data in a current window, as shown in Figure 2(a)-(c);

(2) Header table: the header table contains 6 fields (*item* is the item name; *esn1*, *esn2* and *esn3* are the expected support number of 3 batches data of the current window, respectively; *senT* is the expected support number of the current window; *link* records all nodes of a corresponding item on a tree, but it is not shown in the Figures for simplicity.), as shown in Figure 2(d);

(3) Tail-node table: the tail-node table maintains the tail-nodes of each batch of data in the current window, as shown in Figure 2(e);

(4) Global UDS-Tree: the root of which is initially set as null.

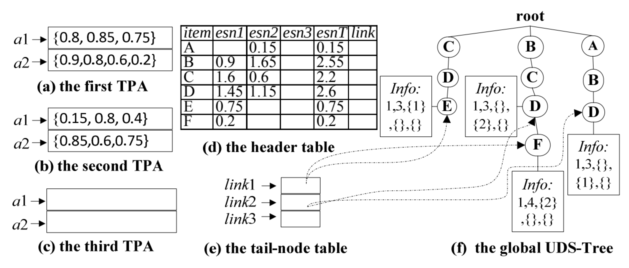
Then process each transaction itemset by the following steps:

Step 1: The probability values of all items in a transaction itemset are stored to a TPA. For example, the TPA in Figure 2(a) maintains the probability values of the first batch of data; *a1* and *a2* in Figure 2(a) correspond to the probability values of the first transaction and the second transaction of the first batch of data, respectively.

Step 2: The transaction itemset is added to a global UDS-Tree. For example, the global UDS-Tree in Figure 2(f) is the result after adding the first batch of data; the nodes "E" and "F" are 2 tail-nodes; as for the values of the *info* field (such as "1;3;{1};{};{}" of node "E"): the first value represents the support number, the second value is the length of the itemset, and the values in the 3 braces ({}) represent the corresponding indexes in the TPA of each batch of data in the current window (e.g. "1" in the first braches on node "E" represents the first row *a1* of the first TPA in Figure 2(a)).

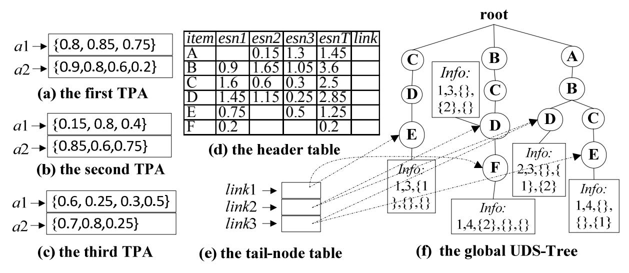
Step 3: The expected support number and *link* information of each item are maintained to the header table. For example, the header table in Figure 2(d) maintains the header information of the first batch of data (*link* is not shown in the Figures for simplicity).

Step 4: Each new tail-node of each batch of data is maintained to a tail-node table. For example, *link1* of the tail-node table in Figure 2(e) links to all tail-nodes of the first batch of data.



item	*esn1*	*esn2*	*esn3*	*esnT*	*link*
A		0.15		0.15	
B	0.9	1.65		2.55	
C	1.6	0.6		2.2	
D	1.45	1.15		2.6	
E	0.75			0.75	
F	0.2			0.2	

Figure 3. The result of processing the first two batches of data



item	*esn1*	*esn2*	*esn3*	*esnT*	*link*
A		0.15	1.3	1.45	
B	0.9	1.65	1.05	3.6	
C	1.6	0.6	0.3	2.5	
D	1.45	1.15	0.25	2.85	
E	0.75		0.5	1.25	
F	0.2			0.2	

Figure 4. The result of processing the first 3 batches of data

Figure 3 is the result after the first 2 batches of data are processed. When the second transaction ((B, 0.8), (C, 0.5), (D, 0.55)) of the second batch is added to the global UDS-Tree, the node "D" on the path "root-B-C-D-F" become a tail-node, as shown in Figure 3(f).

Figure 4 is the result after the first 3 batches of data are processed. When the second transaction of the third batch of data is added to the global tree, we just update the information on the existing tail-node "D" of the path "root-A-B-D", as shown in Figure 4(f).

When the global UDS-Tree contains 3 batches of data, frequent itemsets mining operation can be performed if frequent itemsets are needed.

```
Input: A UDS-Tree T, a global header table H, and a minimum expected support number minExpSN.
Output: FIs (frequent itemsets)
(1)  Add the information on info field on each leaf-node to the field addInfo;
(2)  For each item x in H (from the last item) do
(3)    If(x.esnT ≥ minExpSN) //x.esnT is from the header table H
(4)      Generate an itemset X = x;
(5)      Copy X into FIs;
(6)      Create a header table H_x for X;
(7)      If(H_x is not empty)
(8)        Create a prefix UDS-Tree T_x for X;
(9)        Call SubMining(T_x, H_x, X)
(10)     End if
(11)   End if
(12)   Pass the information of addInfo field to parent nodes;
(13) End for
(14) Return FIs.

SubProcedure SubMining (T_x, H_x, X)
(15) For each item y in H_x (from the last item) do
(16)   Generate an itemset Y = X ∪ y;
(17)   Copy Y into FIs;
(18)   Create a header table H_y for Y;
(19)   If(H_y is not empty)
(20)     Create a prefix UDS-Tree T_y for Y;
(21)     Call SubMining(T_y, H_y, Y)
(22)   End if
(23)   Pass the information of info list field to parent nodes;
(24) End for
```

Figure 5. Mining frequent itemsets from a global UDS-Tree

C. Mining Frequent Itemsets from the Current Window

1) The mining algorithm

The algorithm UDS-FIM employs the pattern-growth approach like the algorithm FP-Growth for mining frequent itemsets. The main differences between UDS-FIM and FP-Growth lie in their tree structures and the information maintained in their header tables. The detailed mining algorithm is shown in Figure 5.

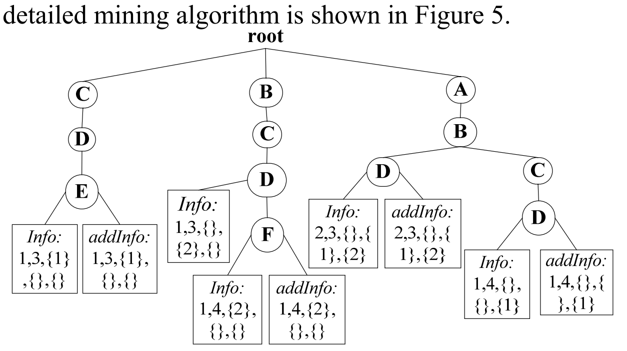


Figure 6. Adding a field addInfo on each leaf-node

Because the probability information on the global UDS-Tree is used not only for the current window, but also for subsequent windows, it cannot be modified when the frequent itemsets is being mining. But the algorithm UDS-FIM needs to modify the probability information when it mines frequent itemsets. So we add a field on each leaf-node to maintain the total probability information of its path (name this field as *addInfo,* as in line 1 in Figure 5); see Figure 6.

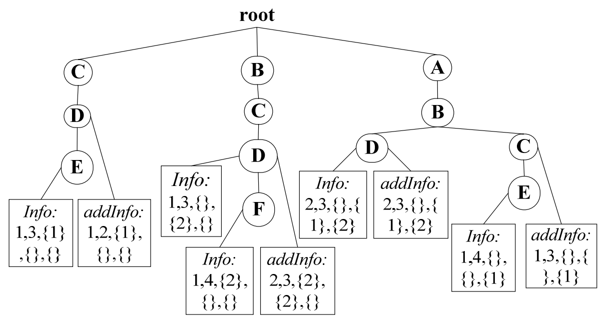


Figure 7. After passing field addInfo on leaf-nodes "E" and "F" to parent nodes

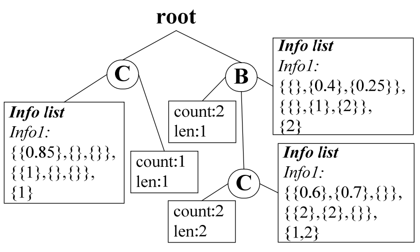
After the *addInfo* field on each node (we denote it as *AI*) is processed, it is passed to its parent node according to the following procedures if the parent node is not root (and if the parent node is root, just set *AI* as null.):

(1) If the parent node contains an *addInfo* field (we denote it as *PAI*), add the support number to *PAI* and copy *pro_ind* of *AI* to the existing *PAI*; otherwise pass *AI* to the parent node and perform the next procedure.

(2) If the parent node contains field *info*, add the support number in *info* to the field *addInfo* and copy *pro_ind* of *info* to the *addInfo* field. For example, Figure 7 is the result after passing *addInfo* on leaf-nodes "E" and "F" in Figure 6 to corresponding parent node.

item	*esn*	*sn*	*link*
B	1.655	4	
C	1.58	3	

(a) the sub header table of itemset {D}



(b) the prefix UDS-Tree of itemset {D}

Figure 8. The prefix UDS-Tree of itemset {D}

The structure of a prefix UDS-Tree is different from that of a global UDS-Tree. We illustrate steps of creating a prefix UDS-Tree by an example. When the item "D" in the header table in Figure 4(d) is processed, all nodes "D" contains the field *addInfo*, as shown in Figure 7. There are 3 branches containing node "D". First, a sub header table for the itemset {D} is created by scanning these 3 branches and their probability information on 3 nodes "D". Figure 8(a) shows the sub header table of the itemset {D} which only contains items whose *esn* (expected support number) are not less than 1.5. Second, a prefix UDS-Tree needs to be created because there is more than one item in the sub header table. The prefix UDS-Tree is created by the following steps:

Step 1: Retrieve the first path "root-C-D" containing the node "D" and its probability information *addInfo*, and process the itemset {C} on this path by the following Substeps:

Substep1.1: Insert the itemset {C} to a prefix UDS-Tree which root is initially set as null;

Substep1.2: Append the probability information to the tail-node of the itemset {C}; the result is the path "root-C" in Figure 8(b). "{{1},{},{}}" on the tail-node "C" maintains the *pro_ind* information; "{{0.85},{},{}}" maintains the *bp* information, e.g., "0.85" is the probability of the itemset {D} in the first transaction of the first batch; "{1}" maintains the *item_ind* information, e.g., "1" shows that item "C" is the first element in the corresponding transactions.

Step 2: Retrieve the second path "root-B-C-D" and its probability information *addInfo*, and process the itemset {BC} on this path by the following Substeps:

Substep2.1: Insert the itemset {BC} to the prefix UDS-Tree;

Substep2.2: Append the probability information which is obtained from *addInfo* to the tail-node "C" of the itemset {BC}; the result is the path "root-B-C" in Figure 8(b). On the node "C" of the path "root-B-C", "{{2},{2},{}}" maintains the *pro_ind* information (two "{2}" represent the second row of the first TPA and the second TPA respectively); "{{0.6},{0.7},{}}" maintains the *bp* information, e.g., "0.6" and "0.7" are the probability values of the itemset {D} in the second transaction of the first batch and the second batch, respectively; "{1,2}" maintains the *item_ind* information, e.g., "1" and "2" represent that items "B" and "C" is the first and second element in corresponding transaction respectively.

Step 3: Retrieve the third path "root-A-B-D" and its probability information *addInfo*, and process the itemset {AB} on this path by the following Substeps:

Substep3.1: Remove the item "A" that is not in the sub header table from the itemset;

Substep3.2: Insert the itemset {B} to the prefix UDS-Tree and maintains its probability information on its tail-node "B"; the result is the path "root-B" in Figure 8(b).

2) An example of mining frequent itemsets

The global tree in Figure 4(f) is used as an example here to illustrate the detailed process of mining frequent itemsets. The minimum expected support number is set to 1.5. The following are the detailed steps:

Step 1: Add the probability information on field *info* of each leaf-node to the field *addInfo* of each leaf-node, as shown in Figure 6;

Step 2: Orderly process the items "F" and "E" in the global header table in Figure 4(d) by the following Substeps:

Substep2.1: Because expected support numbers of items "F" and "E" are less than 1.5, the field *addInfo* on nodes "F" and "E" are passed to their parent node, the result is shown in Figure 7.

Step 3: Process the item "D" in the global header table in Figure 4(d) by the following Substeps:

Substep 3.1: Append item "D" to a *base-itemset* (which is initialized as null; and each new base-itemset is a frequent itemset), because the expected support number of item "D" is not less than 1.5;

Substep 3.2: Create a sub header table for the base-itemset {D} by scanning those paths containing node "D"; the sub header table is shown in Figure 8(a) ;

Substep 3.3: Create a prefix UDS-Tree for the base-itemset {D} by scanning those paths containing node "D" because the sub header table contains more than one items; the prefix tree is shown in Figure 8(b);

Substep 3.4: Mining the prefix tree in Figure 8(b):

Firstly process the item "C" in the header table in Figure 8(a). (1) Append the item "C" to the current base-itemset and get a new frequent itemset {DC}; (2) Create a sub header table for the current base-itemset {DC} by scanning the paths containing node "C" in Figure 8(b); and the sub header table is null; (3) Remove the item "C" from the current base-itemset.

Secondly process the item "B" in Figure 8(a) and get a new frequent itemset {DB}; do not need to create a prefix tree because the new sub header table is null.

Substep 3.5: Remove the item "D" from the current base-itemset.

Substep 3.6: Pass the field *addInfo* on nodes "D" to its parent node.

Step 4: Process the item "C" in the global header table in Figure 4(d) by the following Substeps:

Substep 4.1: Append the item "C" to the current base-itemset and get a new frequent itemset {C};

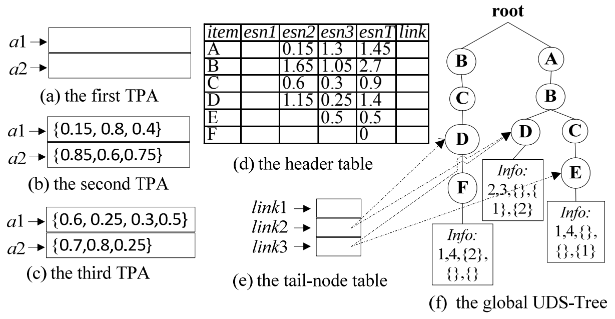
Substep 4.2: Create a sub header table for the base-itemset {C}; the result of the sub header table is null.

Substep 4.3: Pass the field *addInfo* on nodes "C" to its parent node.

Step 5: Process the remaining items in header table in Figure 4(d), and get one new frequent itemset {B}.

The first window of the dataset in Table 1 contains 5 frequent itemsets: {D}, {DC}, {DB}, {C} and {B}.

D. Removing Obsolete Data



item	*esn1*	*esn2*	*esn3*	*esnT*	*link*
A		0.15	1.3	1.45	
B		1.65	1.05	2.7	
C		0.6	0.3	0.9	
D		1.15	0.25	1.4	
E			0.5	0.5	
F				0	

Figure 9. After removing the first batch of data

The tail-node table (see in Figure 4(e)) maintains all tail-nodes of each batch of the current window. When obsolete batch of data need removing, the useless nodes can be removed from the global UDS-Tree by scanning all tail-nodes of the obsolete batch of data.

For example, as shown in Figure 4(f), when the first batch of data are removed from the tree, there are two tail-nodes: the node "E" on path "root-C-D-E" and the node "F" on path "root-B-C-D-F". The path "root-C-D-E" can be removed because the node "E" on this path has no children and there is no other tail-node on this path. When processing the node "F", just one node "F" is removed because its parent node is a tail-node. After removing the obsolete data from the global tree, clear the first TPA and *link1* on the tail-node table for the coming data. After removing the first batch of data, the result is shown in Figure 9.

V. Experiments

In this section, we evaluate the performance of the proposed algorithm UDS-FIM. In our experiments, we also implemented a variant of the algorithm MBP to mine uncertain data stream and denote the revision algorithm as UDS-MBP. We compare the algorithm UDS-FIM with the state-of-the-art algorithms SUF-Growth and UDS-MBP on both types of datasets: the sparse transaction datasets and dense transaction datasets. These three algorithms were written in Java programming language. The configuration of the testing platform is as follows: Windows 7 operating system (64bit), 4G memory, Intel(R) Core (TM) i3-2100 CPU @ 3.10 GHz; Java heap size is 2G.

Table 2 shows the characteristics of 4 datasets used in our experiments. "|D|" represents the number of transactions; "|I|" represents the total number of distinct items; "ML" represents the mean length of all transaction itemsets; "SD" represents the degree of sparsely or density. The real-world datasets *connect* and *kosarak* were obtained from FIMI Repository [33]. The dataset *connect* is a well-known dense dataset whose degree of density is 33.33%; moreover, the length of each transaction itemset is very long, i.e., 43. The dataset *kosarak* is a sparse dataset whose degree of sparsely is 0.02%; it contains 990K transaction itemsets and is a large dataset. The synthetic datasets *T10I4D100k*, and *T20I6D100K* came from the IBM Data Generator [17]. Since these four original datasets do not provide probability values for each item of each transaction itemset, as suggested by literatures [4, 8, 15], we assign a randomly generated existential probability of range (0, 1] to each item of each transaction itemset. The runnable programs and testing datasets can be downloaded from the Google Code repository at http://code.google.com/p/uds-tree/downloads/list.

TABLE II.
DATASET CHARACTERISTICS

Dataset	|D|	|I|	ML	SD (%)	Type
connect	67,557	129	43	33.33	dense
kosarak	990,002	41271	8	0.02	sparse
T10I4D100K	100,000	870	10	1.16	sparse
T20I6D100K	100,000	980	20	2.03	sparse

TABLE III.
EXPERIMENTAL PARAMETERS

Parameters	***Description***
min_exp	minimum expected support threshold
w	window size: number of batches in a window (#)
p	batch size: number of transactions in a batch (K)

The algorithms UDS-FIM, SUF-Growth and UDS-MBP can mine all frequent itemsets from a dataset, so the main performance measures used in this paper are *running time* and *memory size*. The experimental parameters are listed in Table 3, and the parameter values are chosen according to the characteristics of the datasets.

A. Evaluation on varied minimum expected support threshold

In this section, we illustrate the performance of the proposed algorithm under varied *minimum expected support threshold* while the window size and batch size are fixed.

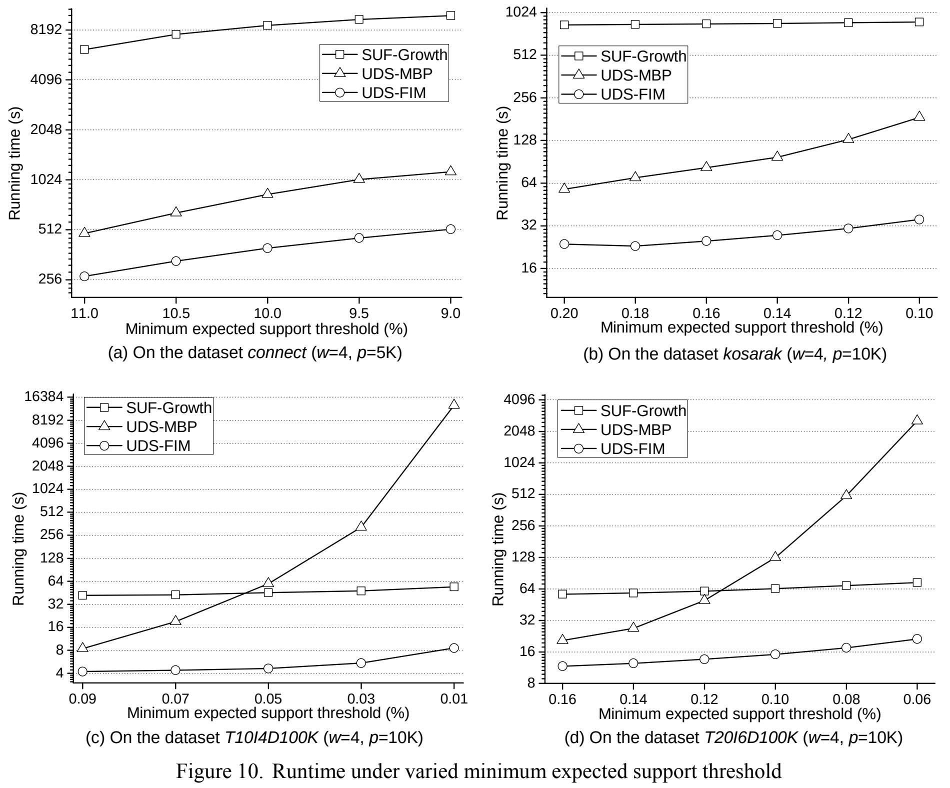


Figure 10. Runtime under varied minimum expected support threshold

Because the dataset *connect* is more dense than the other three datasets, the number of transactions in each batch data is set as 5000 for *connect*, and it is set as 10000 for the other three datasets; the datasets *connect*, *kosarakt*, *T10I4D100K* and *T20I6D100K* are divided into 13, 99, 10 and 10 batches, respectively. On these four datasets, mining operation was performed on 10, 96, 7 and 7 consecutive windows respectively. Figure 10 shows runtime comparison of the algorithms UDS-FIM, SUF-Growth and UDS-MBP; Figure 11 compares the memory usage on four datasets under different minimum expected support thresholds. In this case, UDS-FIM has achieved the best performance among three algorithms. The reason is that the number of tree nodes generated by UDS-FIM is less than that by SUF-Growth (especially on real-world datasets) and that UDS-MBP generates too many candidates; for example, SUF-Growth generates 43732G tree nodes while UDS-FIM generates 443G tree nodes, and UDS-MBP generates 314086 candidates (on *connect* with *w*=4, *p*=5000, *min_exp*=9%). As the minimum expected support threshold decreases, the number of candidate itemsets generated by UDS-MBP sharply increases on synthetic datasets. This leads to the sharp increase of running time of UDS-MBP, as shown in Figure 10.

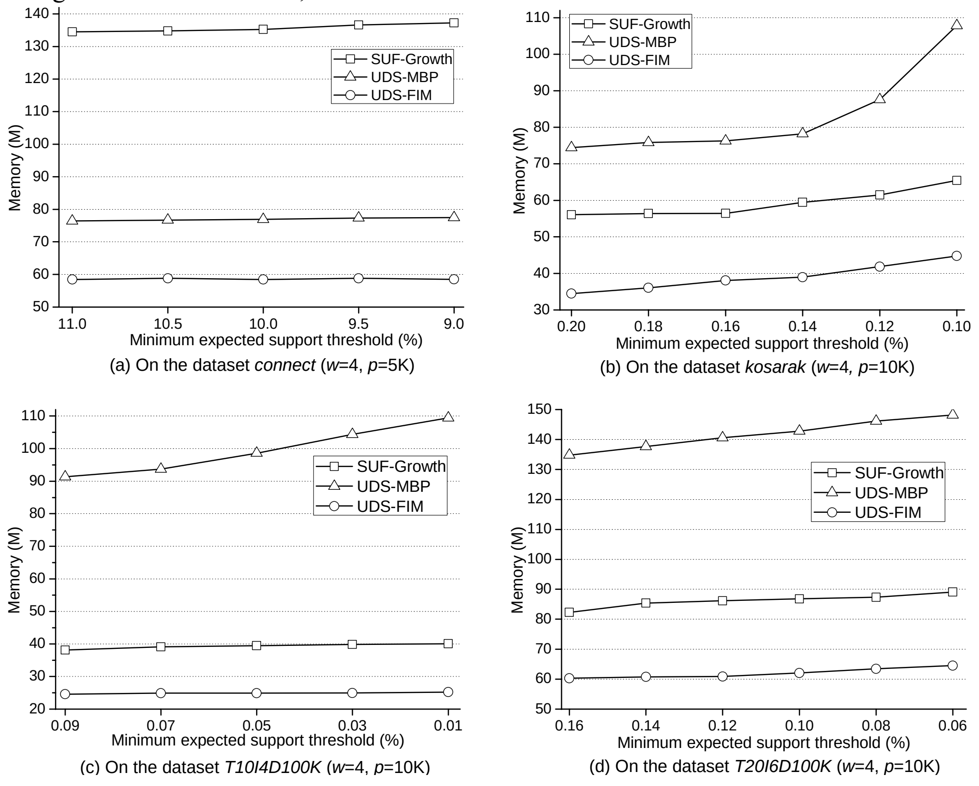


Figure 11. Memory under varied minimum expected support threshold

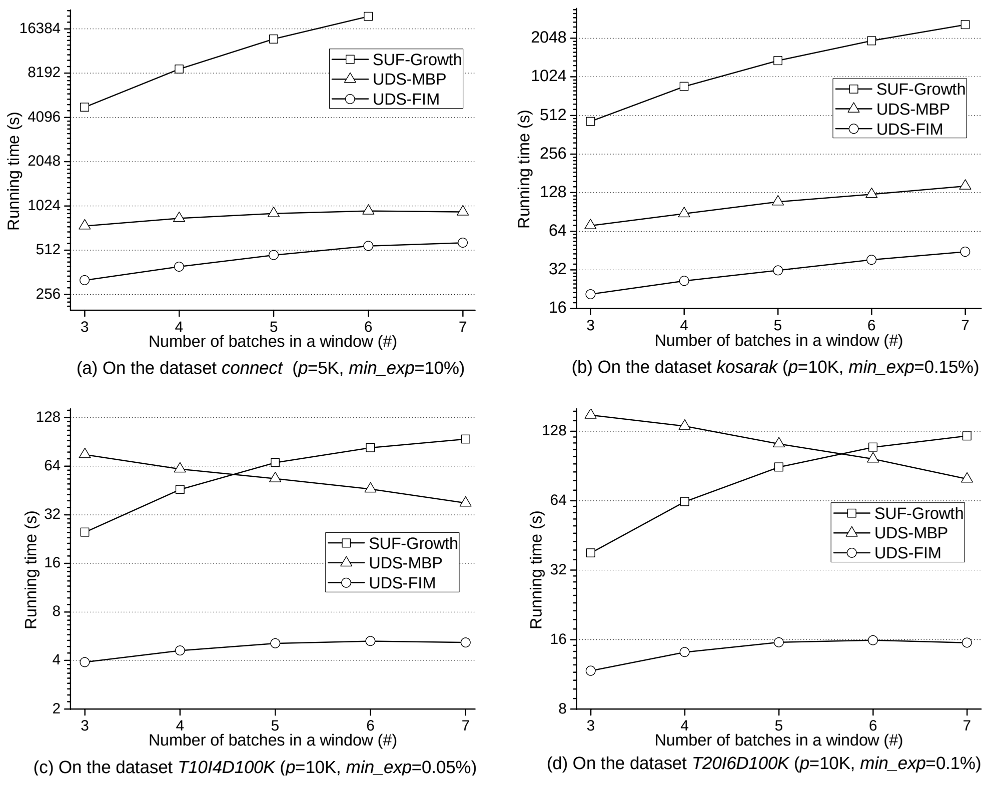
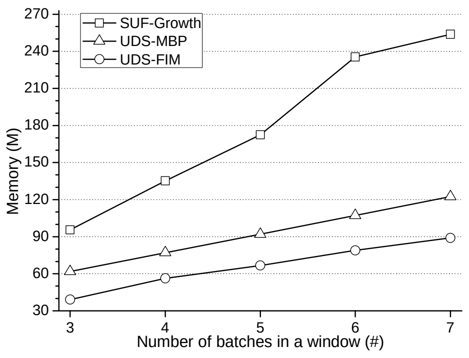
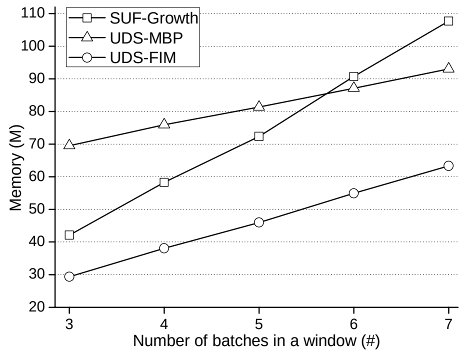


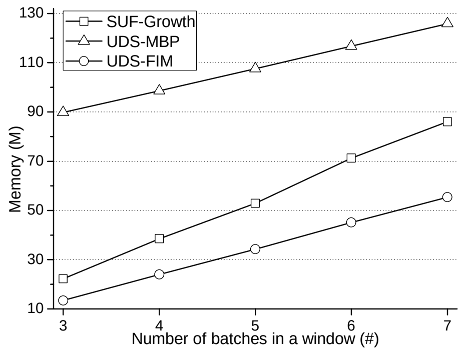
Figure 12. Runtime under varied window size



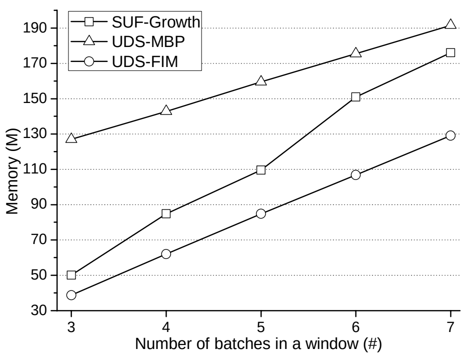
(a) On the dataset *connect* (*p*=5K, *min_exp*=10%)



(b) On the dataset *kosarak* (*p*=10K, *min_exp*=0.15%)

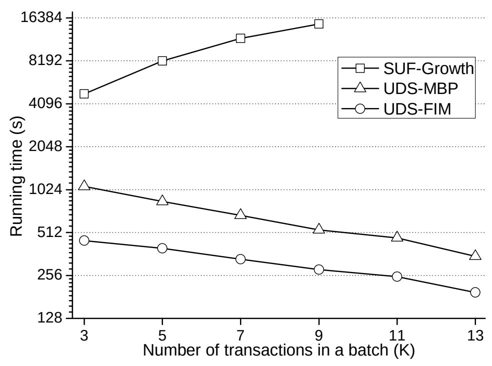


(c) On the dataset *T10I4D100K* (*p*=10K, *min_exp*=0.05%)

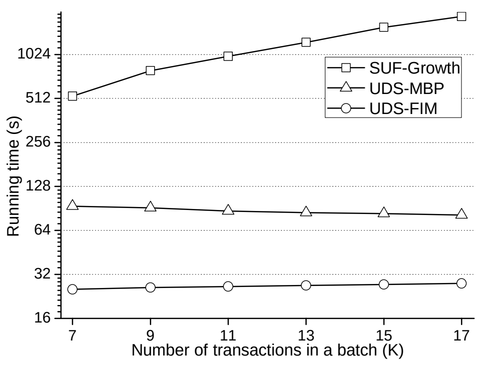


(d) On the dataset *T20I6D100K* (*p*=10K, *min_exp*=0.1%)

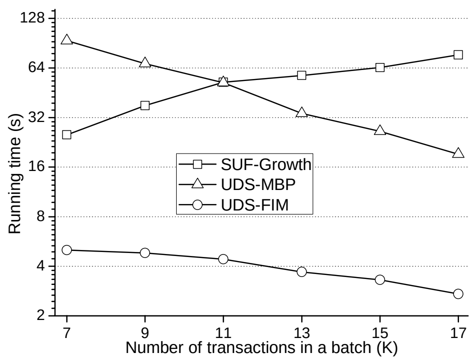
Figure 13. Memory under varied window size



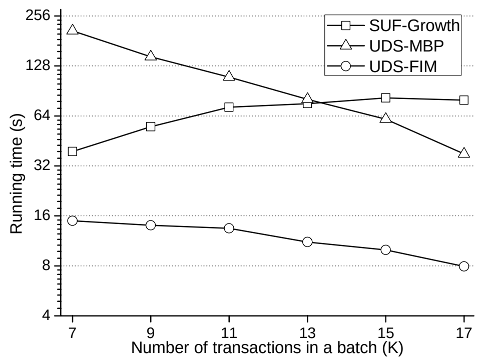
(a) On the dataset *connect* (*w*=4, *min_exp*=10%)



(b) On the dataset *kosarak* (*w*=4, *min_exp*=0.15%)

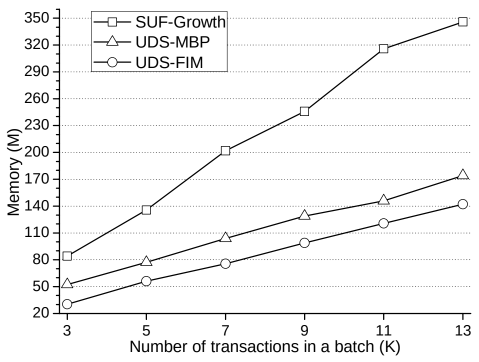


(c) On the dataset *T10I4D100K* (*w*=4, *min_exp*=0.05%)

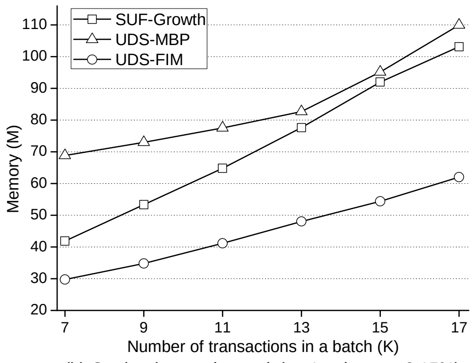


(d) On the dataset *T20I6D100K* (*w*=4, *min_exp*=0.1%)

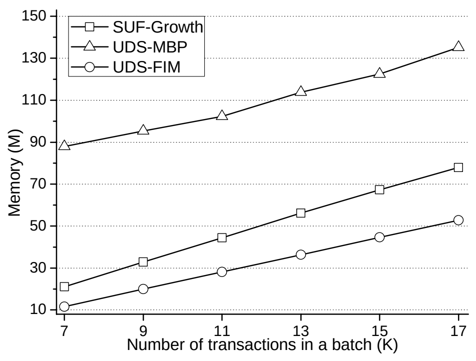
Figure 14. Evaluation under varied batch size



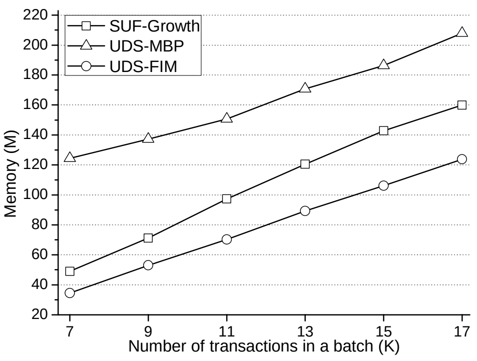
(a) On the dataset *connect* (*w*=4, *min_exp*=10%)



(b) On the dataset *kosarak* (*w*=4, *min_exp*=0.15%)



(c) On the dataset *T10I4D100K* (*w*=4, *min_exp*=0.05%)



(d) On the dataset *T20I6D100K* (*w*=4, *min_exp*=0.1%)

Figure 15. Memory under varied batch size

B. Evaluation on Varied Window Size

In this section, we evaluate the performance of the proposed algorithm under varied number of batches in a window while the parameters *p* and *min_exp* are fixed. A mining operation was performed on each window.

Figures 12-13 show the performance (runtime and memory consumption) of UDS-FIM, SUF-Growth and UDS-MBP using the four datasets respectively when the window size is changed.

From Figure 12, we can see that the performance of the proposed algorithm UDS-FIM beats SUF-Growth and UDS-MBP in terms of running time. The main reason is that UDS-FIM generates less number of tree nodes than SUF-Tree, and it does not generate candidates as UDS-MBP, so UDS-FIM consumes less time to process tree nodes under the same circumstances. (Note that Figure 12 (c, d) indicate an interesting feature: on synthetic datasets, the number of candidates generated by UDS-MBP decreases with the increase of the window size, so the running time of UDS-MBP decreases when the window size increases; however it still can not outperform UDS-FIM).

Moreover, Figures 12 shows that the running time of UDS-FIM is more stable with the increase of the window size on sparse datasets, as shown in Figure 11 (c-d). As the window size increases, the number of transaction itemsets in each window increases, thus the memory consumption of three algorithms increases with the increase of the window size.

Figure13 shows that UDS-FIM has better performance than SUF-Growth and UDS-MBP in terms of memory. This is because that UDS-FIM more efficiently compresses transaction itemsets to a tree than SUF-Growth; moreover, it does not generate candidates and maintains data of a window like UDS-MBP.

C. Evaluation on Varied Batch Size

In this section, we evaluate the performance of the proposed algorithm under varied number of transactions in a batch while the parameters *w* and *min_exp* are fixed.

As the batch size increases, the times of mining operation decreases fast. For example, the mining operation is performed 19 times with *p* =3K while 2 times with *p*=13K on the dataset *connect*; it is 138 times with *p*=7K while 55 times with *p*=17K on the dataset *kosarak*. Thus the total running time (processing a whole dataset) of the algorithms UDS-FIM and UDS-MBP decrease on the datasets *connect*, *T10I4D100K* and *T20I6D100K* though the running time of processing one window increases with the increase of the batch size, as shown in Figure 14 (a, c, d). Note that SUF-Growth generates a bigger tree (too many tree nodes) with the increase of the batch size, its running time of processing one window sharply increases; this leads to increase of its total running time though the times of its mining operations decreases, as shown Figure 14.

With the increase of the batch size, the memory usage increases, as shown Figure 15. However, UDS-FIM still has achieves the best performance among three algorithms in terms of runtime and memory usage.

VI. CONCLUSIONS

In this paper, we propose an efficient algorithm named UDS-FIM for mining frequent itemsets over uncertain transaction data streams based on sliding window method, and also propose a data structure named UDS-Tree for maintaining uncertain transaction itemsets. Unlike the existing algorithms on the discussed problem, whose tree structures are not as compact as FP-Tree structure, the data structure (UDS-Tree) in our proposed algorithm is a tree as compact as the original FP-Tree. The algorithm UDS-FIM firstly maintains probability information of transaction itemsets to an array; then it maintains transaction itemsets to a UDS-Tree and maintains indexes of transaction itemsets in the array to the corresponding tail-nodes. It mines frequent itemsets with just one scan of database. The experimental results show that the performance of UDS-FIM is better than that of SUF-Growth under different experimental conditions, including varied minimum utility thresholds, varied window size, and varied batch size, on both real-world and synthetic datasets.

In this paper, we just employed the sliding window model, but the proposed tree structure UDS-Tree can be applied to the landmark window model and the damped window model for frequent itemsets mining over uncertain data streams. Meanwhile, the proposed algorithm can be adopted for parallel computing. After the header table and the global tree are constructed, the items in the header table can be processed by parallel.

References

[1] C.W. Lin and T.P. Hong, "A new mining approach for uncertain databases using CUFP trees," *Expert Systems with Applications*, Vol.39, no.4, pp.4084-4093, 2011.

[2] G. Liao, L. Wu, C. Wan, and N. Xiong, A practice probability frequent pattern mining method over transactional uncertain data streams, in *8th International Conference on Ubiquitous Intelligence and Computing*. 2011, pp.563-575.

[3] C.C. Aggarwal and P.S. Yu, "A survey of uncertain data algorithms and applications," *IEEE Transactions on Knowledge and Data Engineering*, Vol.21, no.5, pp.609-623, 2009.

[4] C.K. Leung, M.A.F. Mateo and D.A. Brajczuk, A tree-based approach for frequent pattern mining from uncertain data, in *12th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2008)*. 2008, pp.653-661.

[5] X. Sun, L. Lim and S. Wang, "An approximation algorithm of mining frequent itemsets from uncertain dataset," *International Journal of Advancements in Computing Technology*, Vol.4, no.3, pp.42-49, 2012.

[6] T. Calders, C. Garboni and B. Goethals, Approximation of frequentness probability of itemsets in uncertain data, in *IEEE International Conference on Data Mining (ICDM 2010)*. 2010, pp.749-754.

[7] L. Wang, D.W. Cheung, R. Cheng, S. Lee, and X. Yang, "Efficient Mining of Frequent Itemsets on Large Uncertain Databases," *IEEE Transactions on Knowledge and Data Engineering*, no.99(PrePrints), 2011.

[8] C.K. Leung, C.L. Carmichael and B. Hao, Efficient mining of frequent patterns from uncertain data, in *International Conference on Data Mining Workshops (ICDM Workshops 2007)*. 2007, pp.489-494.

[9] Q. Zhang, F. Li and K. Yi, Finding frequent items in probabilistic data, in *International Conference on Management of Data (ACM SIGMOD)*. 2008, pp.819-831.

[10] C.K. Leung and F. Jiang, Frequent itemset mining of uncertain data streams using the damped window model, in *26th Annual ACM Symposium on Applied Computing (SAC 2011)*. 2011, pp.950-955.

[11] C.K. Leung and F. Jiang, Frequent pattern mining from time-fading streams of uncertain data, in *13th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2011)*. 2011, pp.252-264.

[12] C.C. Aggarwal, Y. Li, J. Wang, and J. Wang, Frequent pattern mining with uncertain data, in *15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2009)*. 2009, pp.29-37.

[13] C. Chui, B. Kao and E. Hung, Mining frequent itemsets from uncertain data, in *11th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2007)*. 2007, pp.47-58.

[14] C.K.S. Leung and B. Hao, Mining of frequent itemsets from streams of uncertain data, in *International Conference on Data Engineering*. 2009, pp.1663-1670.

[15] L. Sun, R. Cheng, D.W. Cheung, and J. Cheng, Mining uncertain data with probabilistic guarantees, in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2010)*. 2010, pp.273-282.

[16] T. Bernecker, H.P. Kriegel, M. Renz, F. Verhein, and A. Zuefle, Probabilistic frequent itemset mining in uncertain databases, in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (kDD 2009)*. 2009, pp.119-127.

[17] R. Agrawal and R. Srikant, Fast algorithms for mining association rules in large databases, in *International Conference on Very Large Data Bases (VLDB 1994)*. 1994, pp.487-487.

[18] J. Han, J. Pei and Y. Yin, Mining frequent patterns without candidate generation, in *International Conference on Management of Data (ACM SIGMOD)*. 2000, pp.1-12.

[19] J. Pei, et al., "H-Mine: Fast and space-preserving frequent pattern mining in a large databases," *IIE Transactions (Institute of Industrial Engineers)*, Vol.39, no.6, pp.593-605, 2007.

[20] C.K.S. Leung and Q.I. Khan, DSTree: A tree structure for the mining of frequent sets from data streams, in *IEEE International Conference on Data Mining (ICDM 2007)*. 2007, pp.928-932.

[21] S.K. Tanbeer, C.F. Ahmed, B. Jeong, and Y. Lee, "Sliding window-based frequent pattern mining over data streams," *Information Sciences*, Vol.179, no.22, pp.3843-3865, 2009.

[22] C. Giannella, J. Han, J. Pei, X. Yan, and P.S. Yu, "Mining frequent patterns in data streams at multiple time granularities," *Next generation data mining*, Vol.2003, no.212, pp.191-212, 2003.

[23] M. Deypir and M.H. Sadreddini, "A dynamic layout of sliding window for frequent itemset mining over data streams," *Journal of Systems and Software*, Vol.85, no.3, pp.746-759, 2012.

[24] M.S. Khan, F. Coenen, D. Reid, R. Patel, and L. Archer, "A sliding windows based dual support framework for discovering emerging trends from temporal data," *Knowledge-Based Systems*, Vol.23, no.4, pp.316-322, 2010.

[25] C. Li and K. Jea, "An adaptive approximation method to discover frequent itemsets over sliding-window-based data streams," *Expert Systems with Applications*, Vol.38, no.10, pp.13386-13404, 2011.

[26] C. Chu, V.S. Tseng and T. Liang, "An efficient algorithm for mining temporal high utility itemsets from data streams," *Journal of Systems and Software*, Vol.81, no.7,

pp.1105-1117, 2008.

[27] M. Deypir and M.H. Sadreddini, "Eclat: An efficient sliding window based frequent pattern mining method for data streams," *Intelligent Data Analysis*, Vol.15, no.4, pp.571-587, 2011.

[28] J.H. Chang and W.S. Lee, "estWin: Online data stream mining of recent frequent itemsets by sliding window method," *Journal of Information Science*, Vol.31, no.2, pp.76-90, 2005.

[29] B. Li, Fining frequent itemsets from uncertain transaction streams, in *2009 International Conference on Artificial Intelligence and Computational Intelligence (AICI 2009)*. 2009, pp.331-335.

[30] Y. Kim, E. Park and U. Kim, Mining approximate Frequent itemsets over data streams using window sliding techniques, in *International Conference on Database Theory and Application (DTA 2009)*. 2009, pp.49-56.

[31] H. Li and S. Lee, "Mining frequent itemsets over data streams using efficient window sliding techniques," *Expert Systems with Applications*, Vol.36, no.2 PART 1, pp.1466-1477, 2009.

[32] P.S.M. Tsai, "Mining top-k frequent closed itemsets over data streams using the sliding window model," *Expert Systems with Applications*, Vol.37, no.10, pp.6968-6973, 2010.

[33] B. Goethals. Frequent itemset mining dataset repository, http://fimi.cs.helsinki.fi/data/. Accessed 2011.