Incremental mining of closed inter-transaction itemsets over data stream sliding windows

Shih-Chuan Chiu, Hua-Fu Li, Jiun-Long Huang and Hsin-Han You

Journal of Information Science 2011 37: 208 originally published online 28 February 2011
DOI: 10.1177/0165551511401539

Article

JIS

Journal of Information Science
37(2) 208–220

DOI: 10.1177/0165551511401539
jis.sagepub.com
SAGE

Incremental mining of closed inter-transaction itemsets over data stream sliding windows

Shih-Chuan Chiu
Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan, ROC

Hua-Fu Li
Department of Information Management, Kainan University, Taoyuan 338, Taiwan, ROC

Jiun-Long Huang
Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan, ROC

Hsin-Han You
Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan, ROC

Abstract
Mining inter-transaction association rules is one of the most interesting issues in data mining research. However, in a data stream environment the previous approaches are unable to find the result of the new-incoming data and the original database without re-computing the whole database. In this paper, we propose an incremental mining algorithm, called DSM-CITI (Data Stream Mining for Closed Inter-Transaction Itemsets), for discovering the set of all frequent inter-transaction itemsets from data streams. In the framework of DSM-CITI, a new in-memory summary data structure, ITP-tree, is developed to maintain frequent inter-transaction itemsets. Moreover, algorithm DSM-CITI is able to construct ITP-tree incrementally and uses the property to avoid unnecessary updates. Experimental studies show that the proposed algorithm is efficient and scalable for mining frequent inter-transaction itemsets over stream sliding windows.

Keywords
data mining; data streams; incremental mining; stream sliding window mining; frequent inter-transaction itemsets

1. Introduction

Mining inter-transaction association rules is one of the interesting issues in knowledge discovery and data mining. Traditional association rule mining algorithms focus on finding the intra-transaction itemsets from a large dataset, inter-transaction association rule mining is able to find the relationships among itemsets from different transactions in a large dataset. For example, using traditional rule mining techniques for a stock market database, it can find *intra-transaction* association rules like 'If the stock price of IBM and SUN both go up 5 per cent, 80 per cent of probability the stock price of Microsoft goes up 5 per cent **at the same day**'. In the same database, if we use the inter-transaction association rule mining approach, it can find a rule such like 'If the stock price of IBM and SUN both go up 5 per cent, 80 per cent of probability the stock price of Microsoft will go up 5 per cent **two days later**.'

A number of works about inter-transaction association mining have been investigated [1–7] in the last decade. Lu et al. [1] first proposed the concept of inter-transaction rules and used it to predict stock market movements. Furthermore, Lu et al. [2] proposed two algorithms, E-Apriori and EH-Apriori, for finding inter-transaction itemsets efficiently. E-Apriori is an

Corresponding author:
Jiun-Long Huang, Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan, ROC.
Email: jlhuang@cs.nctu.edu.tw

Apriori-based inter-transaction association rule mining approach and EH-Apriori improves the performance of E-Apriori by employing a hash-based pruning strategy. Feng et al. [3] proposed efficient algorithms based on EH-Apriori for mining inter-transaction association rules under rule templates by using optimization techniques. Tung et al. [4] proposed a lexicographic tree-based approach for mining inter-transaction association rules. In their proposed framework, it first finds the length-1 patterns. Then, a lexicographic tree is constructed and employed to provide a path to find inter-transaction itemsets. After that, an efficient algorithm FITI (First Intra Then Inter) is developed and composed of two phases. First, FITI finds the frequent intra-transaction itemsets, i.e. frequent itemsets. Second, FITI builds a data structure among intra-transaction frequent itemsets for efficient mining of inter-transaction frequent itemsets. Luhr et al. [5] proposed an effective tree structure, EFP-tree (Extended Frequent Pattern Tree), which is a FP-tree-based data structure, to solve this problem. Lee et al. [6] proposed an efficient depth-first searching approach, ITP-Miner, and an effective tree structure, ITP-tree, to discover all frequent inter-transaction itemsets from a large database. Furthermore, the approach of inter-transaction itemset mining is extended from single dimension to multiple dimensions [7]. Many applications of inter-transaction association rules are proposed, such as web usages in web log databases [8], stock price movement in stock market data [9], and polyphonic repeating patterns in music data [10].

To reduce the number of the generated inter-transaction itemsets, the concept of 'closed' is incorporated into inter-transaction itemset mining. Huang et al. defined the closed inter-transaction (also known as continuity) and proposed an efficient algorithm ClosedPROWL to discover closed inter-transaction itemsets [11, 12]. Algorithm ClosedPROWL consists of three phases. In the first phase, ClosedPROWL generates all length-1 closed frequent itemsets. In the second phase, these length-1 closed frequent itemsets are encoded and an encoded horizontal database is constructed based on the encoded length-1 closed frequent itemsets. Finally, a lexicographic tree is used to find all closed frequent inter-transaction itemsets. Lee et al. [13] developed another tree-based approach, ICMiner, to discover closed inter-transaction itemsets efficiently. Algorithm ICMiner uses effective pruning strategies to avoid costly candidate generation and repeated support counting. Dong et al. proposed a novel approach to mine the closed inter-transaction itemsets by using bit approaches [14].

In recent years, database and data mining communities have focused on a new data model in the form of *continuous streams*. It is often referred as *data streams* or *streaming data*. A large number of applications generate large amount of data in real time, such as web log generated from web server, sensor data generated from sensor networks, online transaction flows generated from a retail chain, web click-streams maintained in web applications, call records maintained in telecommunications, performance measurement recorded in network monitoring, etc. Compared with mining static databases, the issue of mining of data streams poses a different challenge in the following aspects [15]. First, each element of streaming data should be examined, at most, once. Second, the memory usage of the data mining systems should be bounded, although new data elements are generated continuously from the data streams. Third, each element of data streams should be processed as fast as possible. Finally, the analytical results of discovered patterns in the stream should be instantly available while it is requested. For mining various interesting patterns from data streams, several problems have been discussed, such as frequent itemset mining [16–20], maximal frequent itemset mining [17–20], sequential pattern mining [21], data clustering [22], data classification [23], and regression analysis [24, 25].

In this paper, we focus on a new problem of mining closed inter-transaction itemsets from data streams. For mining frequent inter-transaction itemsets from streaming data, the existing approaches have to be re-executed for new-incoming data elements to obtain the current result. However, it is inefficient and inappropriate for mining data streams. Hence, in the paper, a new stream mining algorithm, called DSM-CITI (Data Streams Mining for Closed Inter-Transaction Itemset), is proposed for discovering closed inter-transaction itemsets from data streams efficiently. The proposed algorithm DSM-CITI employs an in-memory summary data structure, ITP-tree, to maintain frequent inter-transaction itemsets generated from current data streams. Algorithm DSM-CITI is composed of four stages. First, it reads a basic window of transactions from the buffer in the main memory. Second, it constructs and maintains the ITP-tree, which maintains all frequent inter-transaction patterns. Third, it prunes the out-of-date window and update ITP-tree. Finally, it traverses the ITP-tree to generate all closed inter-transaction itemsets. Note that stages 1 and 2 of DSM-CITI are performed in sequence for a new-incoming basic window. Stages 3 and 4 are performed periodically or when it is needed. Experimental results show that the proposed algorithm is efficient and scalable for the set of all closed inter-transaction itemsets over the sliding window of the data streams. To the best of our knowledge, efficient mining of closed inter-transaction itemsets from data streams has not been investigated in the literature.

The remainder of this paper is organized as follows. We give the problem definition of mining closed and frequent inter-transaction itemsets from data streams over sliding windows in Section 2. In Section3, the proposed algorithm, DSM-CITI, is discussed. The performance results are presented in Section 4. Finally, we conclude our study in Section 5.

2. Problem definition

In this section, the problem of mining closed inter-transaction itemsets in the sliding window over a data stream is defined. To facilitate the understanding of our problem, we refer to the prior work [13] and make the following definitions.

Tid	Itemset	
1	*B*, *C*, *D*	W_1
2	*A*, *D*, *F*	
3	*C*	W_2
4	*C*, *D*	
5	*B*, *C*	
6	*A*, *C*, *E*	W_3
7	*G*	

I={*A*, *B*, *C*, *D*, *E*, *F*, *G*}
A={1, 2, 3, 4, 5, 6, 7}

D={<1, T_1>, <2, T_2>, <3, T_3>, <4, T_4>, <5, T_5>, <6, T_6>, <7, T_7>}

DS=[W_1, W_2, W_3]

Figure 1. An example transaction database D

Let $I = \{ i_1, i_2,\ldots, i_m \}$ be a set of *distinct items*, and $A = \{ a_1, a_2,\ldots, a_n \}$ be a set of *time stamp identifier*. An *itemset* X is a non-empty set of items, where $X \subseteq I$. A transaction database D is composed of a set of transactions in the form of <*tid*, T_{tid}>, where $tid \in A$ and $T_{tid} \subseteq I$. A transaction T_{tid} is an itemset, and *tid* is the time stamp identifier associated with the transaction T_{tid}. Note that in this paper the terms, transactions and itemsets, are used interchangeably.

For example in Figure 1, an example transaction database D is given and it contains seven transactions, <1, T_1>, <2, T_2>, <3, T_3>, <4, T_4>, <5, T_5>, <6, T_6> and <7, T_7>, where T_1 is an itemset {*B*, *C*, *D*} and occurs at *tid* = 1.

Definition 1 Let <u, T_u> and <v, T_v> be two transactions in a transaction database. The *relative distance* between u and v is defined as (u–v), where $u>v$, and v is called a *reference point*. With respect to v, an item i_k at u is called an *extended item* and denoted as $i_k(u–v)$, where (u–v) is called the *span* of the extended item u. Similarly, with respect to v (or the transaction at v), a transaction T_u is called an *extended transaction* and denoted as $T_u(u–v)$. Therefore, an extended transaction T_u consists of a set of extended items, i.e. $T_u(u–v) = \{ i_1(u–v),\ldots, i_s(u–v) \}$, where s is the number of items in T_u.

For example, in the database as shown in Figure 1, an extended transaction of the transaction T_6 with respect to the transaction T_2 is {*A*(4), *C*(4), *E*(4)}.

Definition 2 Let $x_i(d_i)$ and $x_j(d_j)$ be two extended items. We define the order between them, $x_i(d_i)<x_j(d_j)$ if (1) $d_i<d_j$, or (2) $d_i = d_j$ and $x_i<x_j$. In addition, if $x_i(d_i) = x_j(d_j)$ if $d_i = d_j$ and $x_i = x_j$.

For example, $C(0)<A(1)$ and $B(0)<D(0)$. Note that the lexicographic order is used to compare two letters.

Definition 3 An inter-transaction itemset (or a pattern) is defined as a set of extended items, $\{ x_1(d_1), x_2(d_2),\ldots, x_k(d_k) \}$, where $d_1 = 0$, $d_k \le maxspan$, *maxspan* is a user-specified maximum span, $x_i(d_i)<x_j(d_j)$, and $1 \le i < j \le k$.

Definition 4 For a transaction <d_1, T_{d1}> in a transaction database D, a mega-transaction, M_{d1}, is defined as a set of extended transactions in D with respect to $d1$, i.e. $M_{d1} = T_{d1}(0)\cup T_{d2}(d_2–d_1)\cup \ldots \cup T_{dk}(d_k–d_1)$, where <$d_1$, T_{d1}>, <d_2, T_{d2}>,…, <d_k, T_{dk}> are consecutive transaction in D, $d_i<d_{i+1}$, $i<k$, and $d_k–d_1 \le maxspan$. Note that M_{d1} is also an inter-transaction itemset, where d_1 is the reference point.

For example, the database as shown in Figure 1 contains six mega-transactions with *maxspan* = 1, M_1= {*B*(0), *C*(0), *D*(0), *A*(1), *D*(1), *F*(1)}, M_2= {*A*(0), *D*(0), *F*(0), *C*(1)}, M_3= {*C*(0), *C*(1), *D*(1)}, M_4= {*C*(0), *D*(0), *B*(1), *C*(1)}, M_5= {*B*(0), *C*(0), *A*(1), *C*(1), *E*(1)} and M_6= {*A*(0), *C*(0), *E*(0), *G*(1)}.

Definition 5 The number of extended items in a pattern is called the *length* of the pattern. A pattern of length l is called a **length-*l* pattern**.

For example, {*B*(0), *D*(0), *A*(1), *D*(1)} is a length-4 pattern.

Definition 6 (seniority) Let $X= \{ x_1(i_1), x_2(i_2),\ldots, x_m(i_m) \}$ and $Y= \{ y_1(j_1), y_2(j_2),\ldots, y_n(j_n) \}$ be two patterns. We say that $X=Y$ if $x_r(i_r)= y_r(j_r)$ for $1 \le r \le m$, where $i_1 = j_1 = 0$ and $m = n$. We also say that $X < Y$, if (1) $x_1(0) < y_1(0)$; or (2) there exists k (≥1) such that $x_r(i_r)= y_r(j_r)$ for $1 \le r \le k$, and $x_{k+1}(i_{k+1}) < y_{k+1}(j_{k+1})$.

For example, {*C*(0), *D*(1)}={*C*(0), *D*(1)}, {*C*(0), *D*(0), *F*(1), *E*(2)} < {*C*(0), *D*(0), *D*(2)}.

Definition 7 (subpattern) Let $X= \{ x_1(i_1), x_2(i_2),\ldots, x_m(i_m) \}$ and $Y = \{ y_1(j_1), y_2(j_2),\ldots, y_n(j_n) \}$ be two patterns. We say that Y is a supper pattern (or superset) of X if we can find a non-negative integer r and m extended items in Y, $y_{e1}(j_{e1})$, $y_{e2}(j_{e2}),\ldots, y_{em}(j_{em})$, such that $y_{e1}(j_{e1}–r) = x_1(i_1), y_{e2}(j_{e2}–r) = x_2(i_2),\ldots, y_{em}(j_{em}–r) = x_m(i_m)$. In other words, X is a subpattern (or subset) of Y, denoted as $X \subset Y$. Note that if $s = 0$, we say that Y contains X or X is a proper subset of Y, denoted as $X \subset_p Y$.

Definition 8 (frequent pattern) Given a transaction database D, a user-specified minimum support threshold *minsup*, and an inter-transaction itemset X, let D_x be the set of mega-transactions in D containing X. The support of X, *sup*(X), is defined as $|D_x|$. We can say that X is a frequent pattern if $sup(X) \ge minsup$.

Consider the example as shown in Figure 1. Assume that *maxpan* = 1, *minsup* = 2 and X={ *B*(0), *C*(0), *A*(1) }. Since the mega-transaction M_1 (formed by first and second transactions) contains X, M_1 contributes one support to pattern X. Similarly, the mega-transaction M_5 (formed by fifth and sixth patterns) contains X. The pattern X is contributed by two mega-transactions totally, i.e. sup(X) = 2. Since sup(X) ≥ 2, X is a frequent pattern.

Definition 9 (closed pattern) A frequent pattern X is closed if there does not exist an itemset X' such that (1) $X \subset X'$, and (2) $sup(X') = sup(X)$, i.e. X cannot be subsumed by any other patterns.

```
Algorithm DSM-CITI
Input: a data stream , DS = [W1, W2, …, WN], a user-specified minimum support threshold minsup
Output: all frequent inter-transaction itemsets
1:  Initialize a global ITP-tree Ψ;
2:  foreach basic window Wi do /* i=1, 2, …, N */ //stage 1
3:     foreach transaction IT = <tid, Ttid> do
4:        call ITP-tree-construction(IT); //stage 2
5:     end for
6:  end for
7:  if window is out-of-date do /* the recent k window is full*/
8:     l_tid = largest tid in the oldest basic window W1;
9:     for each length-1 pattern p1 with length-1 pattern do
10:       call ITP-tree-pruning(p1, l_tid); //stage 3
11:    end for
12: end if
13: traverse ITP-tree and output all frequent patterns; //stage 4
```

Figure 2. Algorithm DSM-CITI

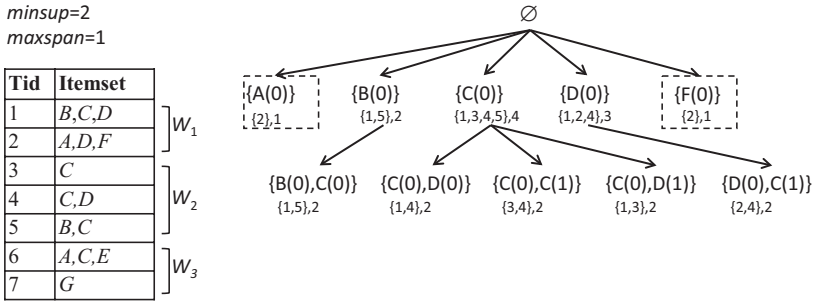


Figure 3. ITP-tree after processing W_1 and W_2

Definition 10 (inter-transaction association rule) Given two frequent patterns X and Y, $X \cap Y = \emptyset$, an inter-transaction association rule is an implication of the form $X \rightarrow Y$, whose support and confidence are not less than the user-specified thresholds *minsup* and minimum confidence *minconf*, respectively. The support of $X \rightarrow Y$ is defined as $sup(X \cup Y)$, while the confidence is defined as $sup(X \cup Y)/sup(X)$.

Definition 11 (data stream) A *data stream*, $DS = [W_1, W_2, \ldots, W_N]$, is an infinite sequence of *basic windows*, where each basic window W_i, $\forall\ i = 1, 2, \ldots, N$, is associated with a window identifier i, and N is the window identifier of the 'latest' basic window W_N. A *basic window* consists of a fixed number of transactions, where each transaction has the corresponding time stamp, $<tid, T_{tid}>$. The *size* of a basic window W is denoted by $|W|$. The *current length* (abbreviated as *CL*) of data stream is $|W_1|+|W_2|+\ldots+|W_N|$.

Problem statement We focus on mining the set of all frequent inter-transaction itemsets in *sliding window* over data streams. In the sliding window model, only the R most recent basic windows, $SW[W_{i+1}, W_{i+R}]$ is stored, where $i+R=N$. Consequently, given a data stream $DS = [W_1, W_2, \ldots, W_N]$, a minimal support threshold *minsup*, and a range of the most recent windows R, the problem is to find the set of all *close inter-transaction itemsets* in sliding window over a data stream.

3. The DSM-CITI algorithm

In this section, an efficient algorithm, called **DSM-CITI** (Data Stream Mining for Closed Inter-Transaction Itemsets), is proposed to mine a set of current closed inter-transaction itemsets over a stream of transactions. A suitable stream sliding window model is constructed and used the proposed algorithm DSM-CITI. The model receives transactions from the data stream and uses properties developed in the work to update the proposed data structure.

The proposed algorithm DSM-CITI is composed of four stages (Figure 2). First, DSM-CITI reads a window of transactions from the buffer in the main memory. Second, it constructs and maintains the in-memory summary data structure, ITP-tree (Inter-Transaction Pattern tree), which maintains all frequent inter-transaction patterns. Third, it prunes the out-of-date window and maintains the data structure ITP-tree. Finally, DSM-CITI traverses the current ITP-tree to output all closed inter-transaction patterns. Stages 1 and 2 are performed in sequence for a new-incoming basic window. Stages 3 and 4 are performed periodically or when needed.

```
Algorithm ITP-tree-construction
Input: a incoming transaction <tid,IT_tid>
Output: a ITP-tree generated so far
1:  foreach item x_i in IT do
2:    if x_i do not appear in length-1 pattern in ITP-tree Ψ then
3:      create a pattern {x_i(0)} as a child of root in ITP-tree Ψ;
4:    end if
5:    add tid to the list of the corresponding length-1 pattern p_1
6:  end for
7:  fi1_set = all frequent length-1 patterns in IT;
8:  foreach frequent length-1 pattern fp_1 do
9:    ITP-tree-maintenance(fp_1 , fi1_set);
10: end for
```

Figure 4. ITP-tree-construction algorithm

3.1. Efficient construction of the proposed summary data structure ITP-tree

In this section, an effective in-memory summary data structure, ITP-tree, is used for maintaining all frequent inter-transaction itemsets, where ITP-tree is developed by Lee et al. [6]. In this work, we modify the ITP-tree for mining frequent inter-transaction patterns from data streams.

Definition 12 (ITP-tree) Each node in an ITP-tree is a pattern composed of two fields: *list* and *children*, where the first field *list* is an index list which stores the time stamp when the *pattern* occurs among transactions, and the second field *children* is a list that links to its child patterns. The root of an ITP-tree is a null pattern Ø. Let $c\_pattern_1$, $c\_pattern_2$, …, $c\_pattern_i$ are the i child patterns of *pattern* in the ITP-tree. We have (1) $pattern \subset_p c\_pattern_1$, $pattern \subset_p c\_pattern_2$, …, $pattern \subset_p c\_pattern_i$, and (2) $pattern = c\_pattern_1 \cap c\_pattern_2 \cap \ldots \cap c\_pattern_i$.

For example, in Figure 3, the node of pattern $\{C(0)\}$ is contained by its child patterns, $\{C(0), D(0)\}$, $\{C(0), C(1)\}$ and $\{C(0), D(1)\}$.

Definition 13 (index list) Given a transaction database D, a pattern X, and let $D_x = \{M_{p1}, M_{p2}, ..., M_{pm}\}$ be the set of mega-transactions in D containing X, the *index list* of the pattern X, denoted by $X.list$, is $< p_1, p_2, \ldots, p_m >$.

For example, in the database as shown in Figure 1, a pattern $X = \{C(0), D(1)\}$ is contained in two mega-transactions, M_1 and M_3. Hence, the index list of X is generated, i.e. $X.list = < 1, 3 >$.

In the framework of ITP-tree, the extension of a candidate pattern of pattern X in ITP-tree is the key operation. Now, we describe the process of generating a candidate pattern c as a child pattern of pattern X in ITP-tree by the *join* operation. The *join* operation is performed on a frequent length-l pattern with a frequent length-1 pattern to generate a set of length-(l+1) candidate patterns c. In addition, we can also obtain the index list of a candidate pattern $c.list$ by *list_gen* operation from the index lists of these two patterns.

Definition 14 (*join* operation and *list_gen* operation) Let the index list of the length-l pattern $pattern_l = \{x_1(d_1), x_2(d_2), \ldots, x_l(d_l)\}$ be $pattern_l.list = < p_1, p_2, \ldots, p_m >$ and the index list of the length-1 pattern $pattern_1 = \{y_1(0)\}$ be $pattern_1.list = < q_1, q_2, \ldots, q_n >$, where $d_1 = 0$, $d_l \leq maxspan$, $x_i(d_i) < x_j(d_j)$, and $1 \leq i \leq j \leq l$. By joining these two patterns, a set of the length-(l+1) pattern is produced. These length-(l+1) patterns are $\{ x_1(d_1), x_2(d_2), \ldots, x_l(d_l), y_1(d_l+k) \}$, for all k satisfying $k \leq maxspan - d_l$ and $y_1(d_l+k) > x_l(d_l)$. The index list of the pattern $\{ x_1(d_1), x_2(d_2), \ldots, x_l(d_l), y_1(d_l+k) \}$ can be obtained by the formula, $pattern_l.list \cap left\_shift_k(pattern_1.list)$, where $shift_k(pattern_1.list) = < q_1-k, q_2-k, \ldots, q_n-k >$.

For example, assume that *maxspan* is 3. By joining one length-3 pattern $X = \{A(0), B(1), E(1)\}$ with $X.list = <1, 5, 7, 10>$ and another length-1 pattern $fi_1 = \{D(0)\}$ with $fi_1.list = <1, 2, 3, 7, 12, 13>$, two patterns are produced, i.e. $c_1 = \{A(0), B(1), E(1), D(2)\}$ and $c_2 = \{A(0), B(1), E(1), D(3)\}$ since $D(2)$ and $D(3)$ are larger than $E(1)$ when *maxspan* = 3. We can obtain $c_1.list$ by performing $X.list \cap left\_shift_2(fi1.list)$, i.e. $<1, 5, 7, 10> \cap <(1-2), (2-2), (3-2), (7-2), (12-2), (13-2)>$, and thereby $c_1.list = <1, 5, 10>$. Similarly, $c_2.list = <10>$ can be obtained by using the same method.

Figure 4 outlines the ITP-tree construction algorithm of algorithm DSM-CITI. The construction scenario of the ITP-tree is described as follows. First, ITP-tree construction algorithm reads the transaction $IT = <tid, T_{tid}>$ from the current basic window W_N in the buffer. If the item X of the transaction IT is already a length-1 pattern in the ITP-tree, the index list of the length-1 pattern is updated by adding a time stamp *tid*. Otherwise, a new length-1 pattern $\{X(0)\}$ is created, and its index list $list\{X(0)\}$ contains only one time stamp *tid*. After that, it maintains a set of frequent length-1 patterns in IT, $fi1\_set$. For each frequent length-1 pattern, it performs the ITP-tree maintenance algorithm as given in Figure 5.

While reading a transaction, an ITP-tree may need the following two tree modifications: **(Case 1.1)** *updating the index lists of some patterns*, and **(Case 1.2)** *generating new patterns for ITP-tree*. Algorithm ITP-tree-maintenance works in DFS manner to find out the nodes that need to be changed. To avoid traversing the whole ITP-tree to find these nodes,

```
Algorithm ITP-tree-maintenance
Input: a pattern pattern, the set of frequent items in IT fi1_set
Output: an updated ITP-tree
1:  lo=the last occurrence in pattern.list;
2:  if (lo + maxspan ≥ IT.tid) then // case 1: update/generate
3:    for each fi1 in fi1_set do
4:      candidate_set = join(pattern,fi1);
5:      for each candidate c in candidate_set do
6:        if c is one of pattern.children c_p then //case 1.1: update some patterns
7:          c_p={x1(i1), x2(i2), …, xm(im)};
8:          c_le = im; /* le is the value of the largest extension of item of pattern */
9:          c_les={x(i) | i = im}; /* les is the set of largest extension item of pattern */
10:         c_lo=the last value in c_p.list;
11:         if(c_les ⊂ fi1_set) and (there exists α in pattern.list such that α =IT.tid−c_le) then
12:           add α to c_p.list;
13:         end if
14:         ITP-tree-maintenance(c_p, fi1_set);
15:       else //case 1.2: generate new patterns
16:         derive c.list by list_gen operation;
17:         if (sup(c) ≥ minsup) then //examine whether c is frequent
18:           pattern.children is added a link to c;
19:           ITP-tree-maintenance(c,fi1_set);
20:         end if
21:       end if
22:     end for
23:   end for
24: else // case 2: do nothing
25:   return; // do nothing for the subtree
26: end if
```

Figure 5. ITP-tree-maintenance algorithm

procedure ITP-tree-construction makes use of the following property to ensure no changes of a subtree of the ITP-tree at certain nodes for improving the performance of algorithm ITP-tree-maintenance.

Property 1 (no change of a subtree) Let *IT* be the transaction that ITP-tree-construction reads. Let *lo* be the last time stamp of *pattern.list*, where *pattern* is the pattern that it is maintaining currently. If *lo* + *maxspan* < *IT.tid*, *pattern* and its subtree will not change.

Prof. The last occurrence of *pattern* in database locates at *lo*. The maximal span of the occurrence of *pattern* is bound at (*lo* + *maxspan*). Since *lo* + *maxspan* < *IT.tid*, it means that *IT* do not affect *pattern*. Moreover, the index list of each child pattern of *pattern* is contained by *pattern.list*. Thus, it implies that *IT* does not affect the subtree of *pattern*, either.

In the framework of ITP-tree-maintenance, it reads a pattern *pattern* first. Let the last time stamp in *pattern.list* be *lo*. Then, it is checked if *lo* + *maxspan* < *tid*. According to Property 1, if the condition is hold, no change will occur to the pattern and its subtree, i.e. it returns directly. Otherwise, the modification of the pattern and its subtree may be involved. Since the modification is involved, the *join* operation is performed to generate a set of candidate patterns by using *pattern* and each element fp_1 in *fi1_set*. For each candidate pattern *c*, it checks if *c* is one of *pattern.children c_p*. If *c* is equal to *c_p*, the index list of *c_p* may need to be updated according to Definition 15, i.e. update strategy. After that, ITP-tree-maintenance(*c_p*, *fi*1_*set*) is performed recursively as shown in Figure 5.

Definition 15 (update strategy) Let *c* be a set of candidate patterns generated by *join* operation, $c\_p = \{ x_1(i_1), x_2(i_2), \ldots, x_m(i_m) \}$ be the child pattern of *pattern*, where *c* = *c_p*. Let *c_le* be the largest extension of *c_p* and let *c_les* be the largest extension set of *c_p*. The index list of *c_p* needs to be updated if and only if the following two conditions are true: (1) *c_les* ⊂ *fi*1_*set* and (2) ∃ $\alpha \in$ *pattern.list* such that $\alpha = tid - c\_le$. If the above conditions are satisfied, the index list of *c_p* will be updated by adding α into the *c_p*.

On the other hand, if *c* is not one of *pattern.children*, *c.list* is generated by the index lists of which the patterns are used to generate *c* by *join* operation according to Definition 14. Since *c.list* is obtained, *sup*(*c*) can be derived from *c.list*. While *sup*(*c*) ≥ *minsup*, it generates a new pattern linked by *pattern*. Then, ITP-tree-maintenance(*c*, *fi*1_*set*) is performed for finding the longer patterns extended from *c*.

3.2. ITP-tree pruning

When the recently mining result we want to obtain or the main memory usage constraint is reached, the out-of-date basic windows are removed from the ITP-tree. The ITP-tree-pruning function is used to prune the nodes whose patterns are not frequent in the ITP-tree. The tree pruning mechanism is performed periodically when needed.

```
Algorithm ITP-tree-pruning
Input: a pattern pattern, the largest time stamp in the oldest basic window l_tid
Output: a curtailed ITP-tree
1: l_tid = largest time stamp in the oldest basic window W1;
2: delete the positions of pattern.list ≤ l_tid;
3: if sup(pattern) ≥ minsup then
4:    for each c_pattern in pattern.children do
5:       ITP-tree-pruning(c_pattern);
6:    end for
7: else if length(pattern)≠1 then
8:    eliminate the subtree of pattern;
9: end if
```

Figure 6. ITP-tree-pruning algorithm

The ITP-tree-pruning algorithm is shown in Figure 6. While the out-of-date basic windows are needed to be removed from the ITP-tree, *ITP-tree-pruning* first finds the largest *tid*, *l_tid*, in the out-of-date windows. The transaction which appears earlier than or equal to *l_tid* is not considered. Moreover, the index of each frequent pattern which stores the instances of the pattern is maintained in the ITP-tree. Thus, it updates the index of the pattern in DFS manner by removing those elements whose value is smaller than or equal to *l_tid*. After that, the support of *n.pattern* is also updated by subtracting the number of the elements it removed from the original *n.index*. According to the *Apriori* property, i.e. *if any length-l pattern is not frequent, its length-(l*+1*) supper-patterns can never be frequent*, the subtrees of *n* maintains the supper-patterns of *n.pattern* in ITP-tree. Thus, when *n.pattern* is not frequent, the subtrees of *n* can be eliminated from the current ITP-tree.

The next stage of algorithm DSM-CITI is to determine the set of all *closed inter-transaction patterns* from the ITP-tree constructed so far. This stage is performed only when the analytical results of the stream are requested.

3.3. Traverse ITP-tree for discovering closed inter-transaction itemsets

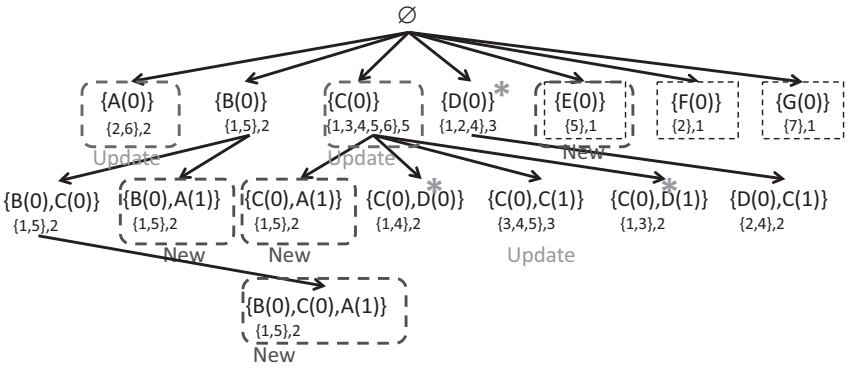
Since all frequent inter-transaction itemsets are maintained in the ITP-tree, it provides a medium result for further deriving closed patterns. In this section, two strategies, *prefix pruning* and *hash pruning*, are developed for filtering non-closed patterns while traversing ITP-tree in DFS manner. The *prefix pruning strategy* filters most non-closed patterns and the *hash pruning strategy* is to hash the rest patterns for subpattern check in the same bucket to obtain the closed patterns.

For prefix pruning, the strategy is designed according to the definition of ITP-tree and DFS manner. Each length-*l* pattern p_l of the ITP-tree is the *l* prefix of its child pattern cp_{l+1}. According to the definition of *closed*, if a pattern p_l is closed, there exists no super pattern of *p* whose support is equal to the support of p_l. In ITP-tree, any child pattern cp_{l+1} of p_l is a super pattern of p_l, because the child pattern of p_l is a prefix pattern of p_l. Thus, if the other condition of *closed*, i.e. $sup(p_l) = sup(cp_{l+1})$, is hold during DFS, p_l is not a closed pattern and will not be generated.

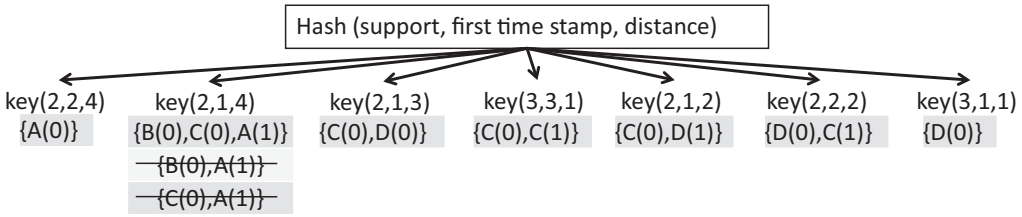
The prefix pruning strategy is able to filter non-closed patterns in the same branch of ITP-tree. To further filter non-closed patterns in different branches, *hash pruning* is employed. Let $p_l.list$ be $<a_1, a_2, \ldots, a_n>$. For each pattern p_l which is not filtered by prefix pruning strategy, p_l is hashed according to *three features* of p_l. The first feature is $sup(p_l)$, and the second one is the first time stamp in $p_l.list$. The third feature is the *distance* between a_1 and a_2, i.e. $a_2 - a_1$. For the patterns in the same bucket, each pattern is checked if it is the subpattern of the other patterns of the same bucket. The checking process starts from the patterns with the shortest length in the bucket. After performing these processes, the rest of the frequent patterns in the buckets are the closed patterns.

3.4. A running example of algorithm DSM-CITI

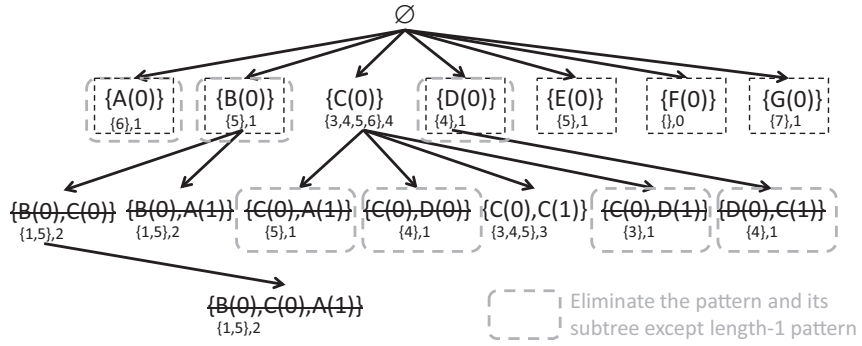
In this section, a running example of incremental mining process of algorithm DSM-CITI over data streams is given. Figure 3 shows the current ITP-tree after processing W_1 and W_2 with *minsup*=2 and *maxspan*=1. Assume that there is a new incoming basic window, W_3. Algorithm DSM-CITI sends each transaction of W_3 to *ITP-tree-construction*. As the new-incoming transaction, $IT = <6, \{A, C, E\}>$, is considered, for each item in *IT*, the corresponding length-1 pattern in ITP-tree is updated by adding the current time stamp into its index list, i.e. the time stamp of 6 is added into the index lists of patterns, $A(0)$, $C(0)$ and $E(0)$. Because the item *E* has not appeared in ITP-tree yet, the length-1 pattern $\{E(0)\}$ is formed and only one time stamp, 6, is stored in its index list. The supports of the updated length-1 patterns are re-checked to examine whether the updated length-1 patterns are frequent or not. The *fi*1*_set* is maintained by collecting the frequent



(a) The ITP-tree after processing W_3 ({A, C, E}, and then {E}).



(b) An example of outputting closed patterns after processing W_3.



(c) The ITP-tree after discarding W_1.

Figure 7. An example of algorithm DSM-CITI

length-1 patterns in *IT*, i.e. *fi*1_*set* = {*A*(0), *C*(0)}. The items in *fi*1_*set* will be used to extend the patterns by using *join* operation.

Next, for each frequent length-1 patterns in the ITP-tree, {*A*(0)}, {*B*(0)}, {*C*(0)} and {*D*(0)}, the first step in process *ITP-tree-maintenance* is performed to examine whether a pattern and the patterns of its subtrees have to be modified. According to Property 1, if *lo* + *maxspan* ≥ *IT.tid*, where *lo* is the last occurrence of the pattern and *IT.tid* is the current time stamp, it is possible that the modification of the pattern and its subtrees are involved. The pattern {*A*(0)} satisfies the condition since 6 + 1 ≥ 6. By using *join* operation, the candidate patterns {*A*(0), *C*(0)}, {*A*(0), *A*(1)} and {*A*(0), *C*(1)} are generated. But, none of them satisfies the frequency examination. The further process of examining its subtrees stops here. For the frequent length-1 pattern {*B*(0)}, the candidate patterns, {*B*(0), *C*(0)}, {*B*(0), *A*(1)} and {*B*(0), *C*(1)} are generated. It finds that {*B*(0), *C*(0)} is already in the ITP-tree thereby following the link to {*B*(0), *C*(0)}.

For pattern {*B*(0), *C*(0)}, two candidate patterns, {*B*(0), *C*(0), *A*(1)} and {*B*(0), *C*(0), *C*(1)}, are generated by using *join* operation. The *list*{*B*(0), *C*(0), *A*(1)} can be derived from *list*{*B*(0), *C*(0)} and *list*({*A*(1)}). By intersecting *list*({*B*(0), *C*(0)}) = <1, 5> and *left_shift*(*list*({*A*(1)})) = <(2 − 1), (6 − 1)>, we can obtain *list*({*B*(0), *C*(0), *A*(1)}) = <1, 5>. The total number of elements in *list*({*B*(0), *C*(0), *A*(1)}) is 2, i.e. *sup*({*B*(0), *C*(0), *A*(1)}) = 2. Since *sup*({*B*(0), *C*(0), *A*(1)}) ≥ 2, {*B*(0), *C*(0), *A*(1)} is a frequent pattern and also a *child* pattern of {*B*(0), *C*(0)}. On the other hand, {*B*(0), *C*(0), *C*(1)} is

Table 1. Stock numbers and names of companies

Stock Number	Company Name	Stock Number	Company Name
2308	AELTA	2311	ASE
2312	Kinpo	2313	Compeq
2317	Foxconn	2321	TECOM
2324	Compal	2330	TSMC

Table 2. Categories of the changes of stock prices

Category	Description
Up-High (UH)	$\delta > 3.5\%$
Up-Low (UL)	$0 < \delta < 3.5\%$
Unbiased (UN)	$\delta = 0\%$
Down-Low (DL)	$-3.5\% < \delta < 0\%$
Down-High (DH)	$\delta < 3.5\%$

not frequent, thereby being eliminated. Furthermore, only the candidate pattern, $\{B(0), A(1)\}$ of $\{B(0)\}$ is frequent after counting support from their index list.

Considering pattern $\{C(0)\}$, $\{C(0), A(1)\}$ is generated in the same way. Note that, for $\{C(0), D(0)\}$, $lo + maxspan \geq IT.tid$ $(4 + 1 < 6)$ is not hold. According to Property 1, $\{C(0), D(0)\}$ and its subtrees do not be changed. Then, considering the candidate pattern $\{C(0), C(1)\}$ of $\{C(0)\}$, it is already in the ITP-tree. The set of the largest extension $c\_les = \{C(1)\}$ is contained by the $fi1\_set$ and there is a value αin the index list of $\{C(0)\}$ (<1, 3, 4, 5, 6>) such that $\alpha = IT.tid - c\_le$ $(5 = 6 - 1)$. The index list of $\{C(0), C(1)\}$ needs to be updated by adding α. The next pattern that it processes is $\{C(0), D(1)\}$ and then $\{D(0)\}$. Both of them satisfy the property. Hence, the process does not have to go deeper. Consequently, algorithm *ITP-tree-maintenance* for the transaction $\{A, C, E\}$ is completed.

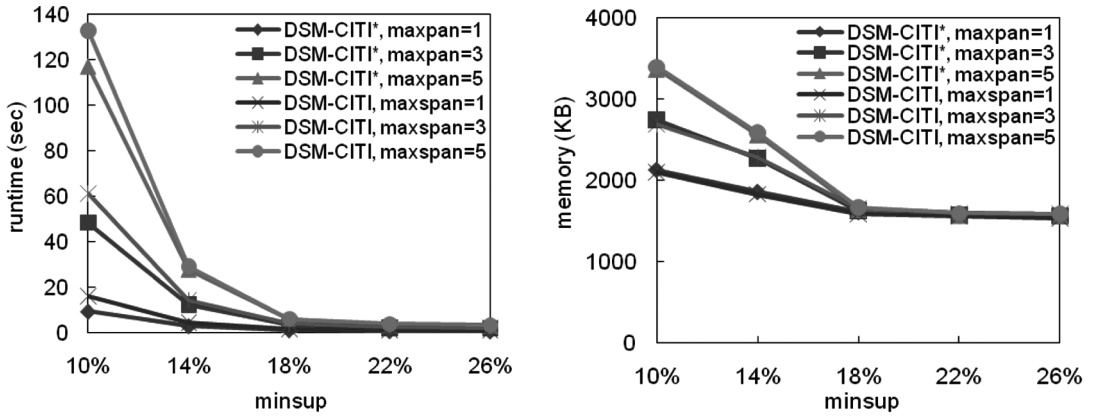
Assume that the next new-incoming transaction IT is <7, $\{G\}$> in W_3. A new item is identified. Then, it generates a length-1 pattern for it. However, the incoming transaction $\{G\}$ contains no frequent items, that is, the $fi1\_set$ is empty. Since $fi1\_set$ is an empty set, we do not modify the ITP-tree. The result of the ITP-tree is given in Figure 7a.

Now, for instance, a user query is given for obtaining the mining result, i.e. *returning all closed patterns* of W_1, W_2 and W_3. By traversing the ITP-tree in DFS manner, the patterns derived sequentially are $\{A(0)\}$, $\{B(0)\}$, $\{B(0), C(0)\}$, $\{B(0), C(0), A(1)\}$, $\{C(0)\}$, and so on. During traversing the ITP-tree, $\{B(0)\}$ and $\{B(0), C(0)\}$ can be filtered according to prefix pruning strategy. The patterns which are not filtered by the prefix pruning strategy are hashed according to their features. The hashed results are given in Figure 7b. In the bucket of $key(2, 1, 4)$, there are three patterns, i.e. $\{B(0), C(0), A(1)\}$, $\{B(0), A(1)\}$ and $\{C(0), A(1)\}$. After that, we check if any pattern is contained by the others beginning from the shortest length patterns. When $\{B(0), A(1)\}$ and $\{C(0), A(1)\}$ are contained by $\{B(0), C(0), A(1)\}$, they are *not* closed patterns and are eliminated. Thus, all closed patterns in this instance are $\{A(0)\}$, $\{B(0), C(0), A(1)\}$, $\{C(0), D(0)\}$, $\{C(0), C(1)\}$, $\{C(0), D(1)\}$, $\{D(0), C(1)\}$, and $\{D(0)\}$.

Assume that the result of the recent two windows is maintained in the ITP-tree. That means all transactions of window W_1 have to be removed from the ITP-tree. First, the *ITP-tree-pruning* process finds l_tid, i.e. $l\_tid = 2$. It updates the index list of each length-1 pattern by removing the occurrences occ in the index list where $occ \leq l\_tid$. In Figure 7c, the length-1 patterns in the dotted block indicate that the pattern turns from frequent to infrequent. The process is applied for each updated pattern in DFS manner. When the pattern is infrequent, the pattern and its subtrees are eliminated but length-1 pattern is not removed from the ITP-tree. Thus, all subtrees of $\{B(0)\}$ are deleted. While $\{C(0)\}$ is frequent, it goes deeper and eliminates $\{C(0), A(1)\}$. It performs for the rest of the patterns in the same way. Finally, the frequent patterns in the ITP-tree are $\{C(0)\}$ and $\{C(0), C(1)\}$.

4. Performance evaluation

In this section, we present several experimental results on the performance of our proposed DSM-CITI algorithm. All the experiments were conducted on an IBM desktop computer with a 3.20 Ghz Intel(R) Pentium(R) dual-core processor with three gigabytes main memory running FreeBSD 7.2-Release operating system. The proposed DSM-CITI algorithm was implemented in C++ with Boost C++ library. For the experimental evaluation, we used both real and synthetic datasets. Since algorithm DSM-CITI was able to discover the same results of the prior algorithms, we focus on the performance of algorithm DSM-CITI with or without Property 1 by using varied characteristics of datasets. Note that algorithm



(a) Runtime versus minimum support. (b) Memory usage versus minimum support.

Figure 8. Experimental results of algorithms DSM-CITI* and DSM-CITI on the real dataset

Table 3. Experimental parameters

Parameter	Description
$\|D\|$	number of transactions
$\|T\|$	average size of transactions
$\|L\|$	number of potentially frequent inter-transaction itemsets
$max(\|L\|)$	maximum size of the transactions
$\|I\|$	average length of the potentially frequent inter-transaction itemsets
$max(\|I\|)$	maximum length of the potentially frequent inter-transaction itemsets
$\|\Sigma\|$	number of items
R	maximum interval of itemsets

DSM-CITI with the property and algorithm *DSM-CITI without the property* are denoted as *DSM-CITI** and *DSM-CITI*, respectively.

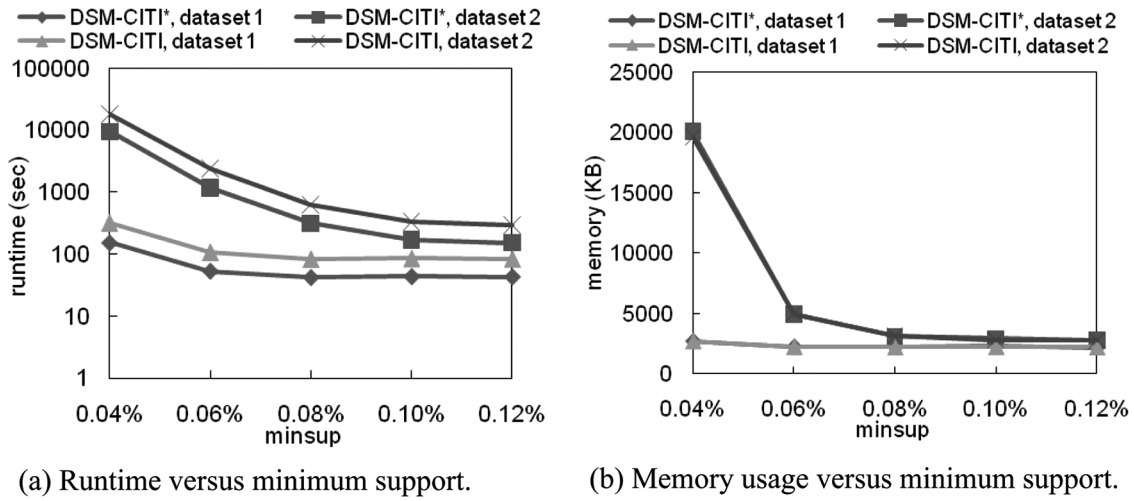
4.1. Experiments on the real dataset

The performance evaluation of algorithm DSM-CITI* (DSM-CITI with Property 1) and DSM-CITI (DSM-CITI without Property 1) are evaluated by conducting the experiments on the real dataset. We use the stock dataset as our real dataset. We first retrieved the stock data of eight companies as listed in Table 1 from 1 January 1995 to 20 May 2010 from Taiwan Stock Exchange Daily Official.[1] Then, the change of the stock price (denoted as δ) was split into five categories as given in Table 2. Hence, there were 3993 transaction days during this period and 40 distinct elements. We had mined 1000 transaction days in advance and stored the results in the proposed ITP-tree. We evaluated the execution time of processing an incoming basic window of 200 transaction days and pruning the oldest basic window.

Experimental results of execution time and memory usage of algorithms DSM-CITI* and DSM-CITI on the real dataset are given in Figure 8. Figure 8a shows the execution time of algorithms DSM-CITI* and DSM-CITI, with different *maxspan*, 1, 3, and 5 when *minsup* increases from 0.05% to 0.2%. As can be seen, we can find that the execution time decreases when *minsup* increases. Moreover, the increasing of the algorithm with higher *maxspan* is much higher when *minsup* is decreasing. Because the size of the ITP-tree grows when *maxspan* increases, it takes much more time to maintain the ITP-tree of a larger size. Figure 8b shows the memory usage of our proposed algorithm when *minsup* increases with various *maxspan*. As observed, the memory usage increases as *minsup* decreases and *maxspan* increases. When *minsup* becomes larger, less frequent patterns are generated. On the other hand, when *maxspan* becomes larger, more frequent patterns with longer lengths are found. Notice that the memory usages of different *maxspan* are similar when *minsup* excesses 18% because no more frequent patterns can be found. Even though *maxpsan* increases, no longer patterns can be found. Consequently, for the same *maxspan*, algorithm DSM-CITI* is 1.5–1.7 times faster than algorithm DSM-CITI. This is because the property is able to reduce abundant unnecessary traverses for ITP-tree modification.

Table 4. Parameter settings of two datasets

Parameter	Data Set 1	Data Set 2
$\|D\|$	100K	100K
$\|T\|$	5	8
$max(\|T\|)$	10	16
$\|L\|$	10000	10000
$\|I\|$	4	5
$max(\|I\|)$	8	10
$\|\Sigma\|$	500	500
R	3	4



(a) Runtime versus minimum support. (b) Memory usage versus minimum support.

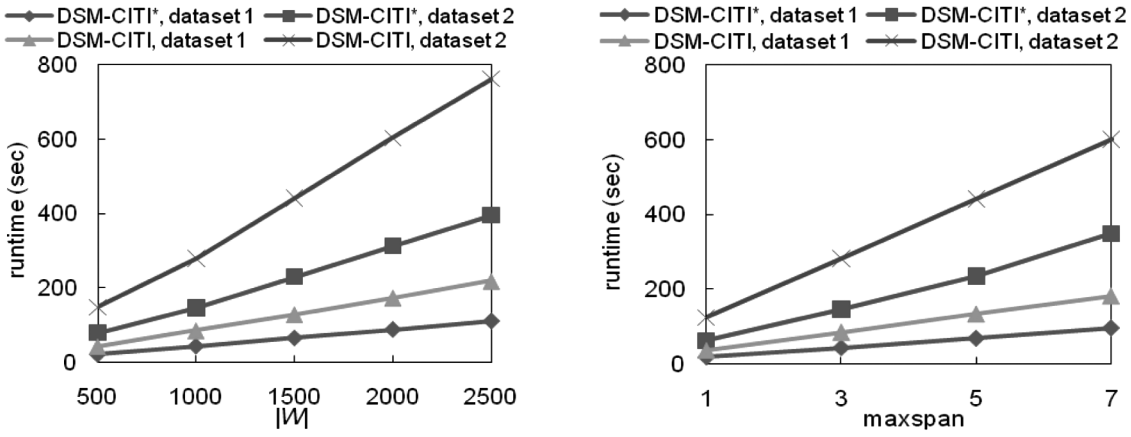
Figure 9. Experimental results on synthetic datasets (minimum support)

4.2. Experiments on the synthetic datasets

To evaluate the scalability of the proposed DSM-CITI algorithm, the experiments on the synthetic datasets were conducted. The proposed algorithm is evaluated on two synthetic datasets generated by using the method [4]. The parameters used in these experiments are shown in Table 3. The synthetic data generation process consists of two steps. First, the potentially frequent inter-transaction itemsets are generated. After that, the transactions in the database are generated from these itemsets. The length of each potentially frequent inter-transaction itemset is derived from a Poisson distribution with mean = $|I|$ and the size of each transaction is derived from a Poisson distribution with mean = $|T|$.

The parameter settings of two synthetic datasets, denoted by *dataset-1* and *dataset-2*, are listed in Table 3. Both datasets have 1,000,000 transactions and 500 distinct items. Dataset-1 has the average size of transaction $|T|$ of 5 items and the average length of potentially frequent inter-transaction $|I|$ is 4 items. In dataset-2, the average size of transaction $|T|$ and the average length of potentially frequent inter-transaction $|I|$ are set to 8 and 5, respectively. From the characteristic of the datasets, dataset-1 is a *sparse* dataset but dataset-2 is a *dense* dataset. In the following experiments, the synthetic dataset stream is broken into 100 basic windows for simulating the continuous characteristic of streaming data, where each basic window contains 1000 transactions. Hence, 20 basic windows, i.e. 20,000 transactions are mined and the result is maintained in the ITP-tree. We evaluate the execution time of algorithm DSM-CITI for processing a new incoming basic window and pruning an oldest basic window. The execution time is an average time of processing 80 basic windows. Furthermore, the default value of user-defined minimum support *minsup* is 0.15%, the maximal span *maxspan* is 3, and the size of each basic window $|W|$ is 1000.

Figures 9a and 9b show the effect of *minsup* on execution time and memory usage of algorithms DSM-CITI* and DSM-CITI, respectively. For a better visual inspection, the y axis of Figure 9a is presented on a log scale. From Figure 9,



(a) Runtime versus the size of the basic window.

(b) Runtime versus the maximum span.

Figure 10. Effect of other parameters

we see that both execution time and memory usage of algorithms DSM-CITI* and DSM-CITI decrease when *minsup* increases. When *minsup* increases, less patterns are found and stored in the ITP-tree. Hence, the smaller ITP-tree makes tree maintenance efficiently. Furthermore, the performances of both algorithms on dense dataset dataset-2 are faster than that of the sparse dataset dataset-1. Based on the experimental results of Figure 9, algorithm DSM-CITI* is 1.7–2 times faster than algorithm DSM-CITI. But the memory usages of algorithm DSM-CITI* and DSM-CITI are almost the same because using the property does not affect the patterns they found.

Figure 10a shows the execution time of algorithms DSM-CITI* and DSM-CITI when the size of the basic window is changed from 500 to 2500. It is intuitive that increasing the size of the basic window will increase the execution time because more transactions have to be processed. From Figure 10a, we can see that the dataset-1 increases slightly because of the sparseness of the dataset. On the other hand, the existence of the longer patterns in the dataset-2 makes the size of the ITP-tree larger. Due to the larger size of the ITP-tree, it takes more time to update the patterns maintained in the ITP-tree. For the same dataset, algorithm DSM-CITI* outperforms DSM-CITI in varied sizes of the basic window. Figure 10b gives the execution times of algorithms DSM-CITI* and DSM-CITI on both synthetic datasets as the parameter *maxspan* increases from 1 to 7. From this figure, we can find that the execution time of dataset-1 and dataset-2 increase as *maxspan* increases. The increment of *maxspan* indicates that the patterns with longer length are allowed to be mined. Since the length of pattern is longer, the height of the ITP-tree increases. Due to the DFS process of the ITP-tree-maintenance, the increment of tree height may disadvantage the performance. Furthermore, for the same dataset, algorithm DSM-CITI* is 1.5-2 times faster than algorithm DSM-CITI due to the utility of the property.

5. Conclusions

In this paper, we propose an efficient incremental mining algorithm DSM-CITI for discovering a set of all frequent inter-transaction itemsets from data streams over sliding windows. An in-memory summary data structure ITP-tree is employed in algorithm DSM-CITI to maintain discovered frequent inter-transaction itemsets. Moreover, algorithm DSM-CITI is able to construct ITP-tree incrementally and uses the property to avoid unnecessary updates. Experimental studies show the proposed algorithm is efficient and scalable for mining the set of all frequent inter-transaction itemsets from data streams. Further work includes mining frequent inter-transaction itemsets from damped stream sliding windows and mining top-*k* inter-transaction itemsets from stream sliding windows.

Note

1. www.twse.com.tw

References

[1] H. Lu, J. Han and L. Feng, Stock movement prediction and N-dimensional inter-transaction association rules, *Proceedings of ACM-SIGMOD Workshop on Research Issues on Data Mining and Knowledge, Seattle, USA, June 1998* (ACM, New York, 1998) 12:1–12:7.

[2] H. Lu, L. Feng and J. Han, Beyond intratransaction association analysis: mining multidimensional inter-transaction association rules, *ACM Transactions on Information Systems* 18(4) (2000) 423–454.
[3] L. Feng, J.X. Yu, H. Lu et al., A template model for multidimensional inter-transactional association rules, *VLDB Journal* 11(2) (2002) 153–175.
[4] A.K.H. Tung, H. Lu, J. Han et al., Efficient mining of intertransaction association rules, *IEEE transactions on Knowledge and Data Engineering* 15(1) (2003) 43–56.
[5] S. Luhr, G. West and S. Venkatesh, *An Extended Frequent Pattern Tree for Intertransaction Association Rule Mining: Technical Report* (Curtin University of Technology, Perth, 2005).
[6] A.J.T Lee and C.S. Wang, An efficient algorithm for mining frequent inter-transaction patterns, *Information Sciences* 28(8) (2007) 3453–3476.
[7] Q. Li, L. Feng and A. Wong, From intra-transaction to generalized inter-transaction: landscaping multidimensional contexts in association rule mining, *Information Sciences* 172(3–4) (2005) 361–395.
[8] J. Chen, L. Ou, J. Yin et al., Efficient mining of cross-transaction web usage patterns in large database, *Proceedings of the 7th International Conference on Data Warehousing and Knowledge Discovery, Copenhagen, Denmark, August* (2005) 519–528.
[9] H. Chhinkaniwala and P. Santhi Thilagam, InterTARM: FP-tree based framework for mining inter-transaction association rules from stock market data, *Proceedings of the 2008 International Conference on Computer Science and Information Technology (ICCSIT 08), Singapore, Singapore, August 2008* (IEEE Computer Society, Los Alamitos, 2008) 513–517.
[10] S.C. Chiu, M.K. Shan, J.L. Huang et al., Mining polyphonic repeating patterns from music data using bit-string based approaches, *Proceedings of International Conference on Multimedia and Expo (ICME'09), New York, USA, June 2009* (IEEE Computer Society, Los Alamitos, 2009) 1170–1173.
[11] K.Y. Huang, C.H. Chang and K.Z. Lin, ClosedPROWL: efficient mining of closed frequent continuities by projected window list technology, *Proceedings of 5th SIAM International Conference on Data Mining, Houston, USA, November 2005* (ACM, New York, 2005) 501–504.
[12] K.Y. Huang, C.H. Chang and K.Z. Lin, Efficient discovery of frequent continuities by projected window list technology, *Journal of Information Science and Engineering* 24(4) (2008) 1041–1064.
[13] A. J.-T. Lee, C.-S. Wang, W.-Y. Wang, Y.-A. Chen and H.-W. Wu, An efficient algorithm for mining closed inter-transaction itemsets, *Data and Knowledge Engineering* 66(1) (2008) 68–91.
[14] J. Dong and M. Han, IFCIA: An efficient algorithm for mining intertransaction frequent closed itemsets, *Proceedings of Fourth International Conference on Fuzzy Systems and Knowledge Discovery 2007* (IEEE Computer Society, Los Alamitos, 2007) 678–682.
[15] B. Babcock, S. Babu, M. Datar et al., Models and issues in data stream systems, *Proceedings of the 2002 ACM Symposium on Principles of Database Systems 2002* (ACM, New York, 2002) 1–16.
[16] J.H. Chang and W.S. Lee, estWin: Online data stream mining of recent frequent itemsets by sliding window method, *Journal of Information Science* 31(2) (2005) 420–432.
[17] K. Gouda and M. Zaki, Efficiently mining maximal frequent itemsets, *Proceedings of the IEEE International Conference on Data Mining, San Jose, USA, November 2001* (IEEE Computer Society, Los Alamitos, 2001) 163–170.
[18] H.F. Li, S.Y. Lee and M.K. Shan, Online mining maximal frequent structures in continuous landmark melody streams, *Pattern Recognition Letters* 26(11) (2005) 1658–1674.
[19] Q. Zou, W. Chu and B. Lu, SmartMiner: A depth first algorithm guided by tail information for mining maximal frequent itemsets, *Proceedings of the IEEE International Conference on Data Mining, Maebashi City, Japan, December 2002* (IEEE Computer Society, Los Alamitos, 2002) 570.
[20] H.F. Li and S.Y. Lee, Approximate mining of maximal frequent itemsets in data streams with different window models, *Expert Systems with Applications* 35(3) (2008) 781–789.
[21] J.H. Chang and W.S. Lee, Efficient mining method for retrieving sequential patterns over online data streams, *Journal of Information Science* 31(5) (2005) 420–432.
[22] S. Guha, N. Mishra, R. Motwani et al., Clustering data streams, *Proceedings of the Annual Symposium on Foundations of Computer Science, Redondo Beach, November 2000* (IEEE Computer Society, Los Alamitos, 2000) 359–366.
[23] L. O'Callaghan, N. Mishra, A. Meyerson et al., High-performance clustering of streams and large data sets, *Proceedings of the 2002 International Conference of Data Engineering, San Francisco, USA, April 2002* (IEEE Computer Society, Los Alamitos, 2002) 685–694.
[24] H. Wang, W. Fan, P.S. Yu et al., Mining concept-drifting data streams using ensemble classifiers, *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, USA, June 2003* (ACM, New York, 2003) 226–235.
[25] W.G. Teng, M.S. Chen and P.S. Yu, A regression-based temporal pattern mining scheme for data streams, *Proceedings of the 29th International Conference on Very Large Data Bases, Berlin, Germany, September 2003* (VLDB Endowment, 2003) 99–104.
[26] Y. Chen, G. Dong, J. Han et al., Multi-dimensional regression analysis of time-series data streams. In: *Proc. of 2002 International Conference on Very Large Data Bases, Hong Kong, China, August 2002* (VLDB Endowment, 2002) 323–334.