Thai Journal of Mathematics
Volume 16 (2018) Number 1 : 1–23
http://thaijmath.in.cmu.ac.th
ISSN 1686-0209

An Incremental Approach to Share-Frequent Itemsets Mining

Chayanan Nawapornanan[†,1], **Sarun Intakosum**[†] **and Veera Boonjing**[‡]

[†]Department of Computer Science, Faculty of Science
King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand
e-mail : `56605008@kmitl.ac.th (C. Nawapornanan)`
`kisarun@kmitl.ac.th (S. Intakosum)`

[‡]International College King Mongkut's Institute of Technology Ladkrabang
Bangkok, Thailand
e-mail : `kbveera@kmitl.ac.th (V. Boonjing)`

Abstract : The share-frequent itemsets mining becomes an important topic in the mining of association rules because it can provide useful knowledge such as total quantity of items sold and total profit. In the past, the efficient MCShFI algorithm was successfully proposed to discover complete share-frequent itemsets on a database. When the database is updated, the algorithm can obtain current complete share-frequent itemsets by using the batch approach-mining the whole updated database. To improve mining execution time, we propose a new incremental approach to the problem with the Fast Update (FUP) concept. It obtains the current result by mining only new transactions and updating the previous existing result with this mined result.

Keywords : data mining; share-frequent itemsets mining; incremental mining.
2010 Mathematics Subject Classification : 68T20; 68W01.

1 Introduction

Mining frequent itemsets is the essential process and most expensive in association rule mining [1]. Many algorithms of mining frequent itemsets have been

[1]Corresponding author.

proposed to find frequent patterns such as level-wise approaches [2, 3, 4, 5, 6] and pattern-growth algorithms [7, 8, 9, 10]. The frequent itemsets mining has some limitations, that is, in each transaction, each item will appear in a binary frequency (either absent or present). Moreover, it only considers whether an item is bought in to a transaction or not [11]. Therefore, mining share-frequent itemsets [12] was proposed to accommodate the quantity of each item. Its share measure can provide useful knowledge about the numerical values that are typically associated with the transaction items, such as the total quantity of items sold or the total profit. [13] proposed an efficient algorithm for mining complete share-frequent itemsets, named MCShFI, for efficiently extracting complete share-frequent itemsets based on a level-wise generating property. It can reduce the runtimes by using transaction measure value as an effective upper bound of each item to generate only the promising candidate itemsets in each iteration. To obtain a current mining result from an updated database, however, this algorithm has to rescan the whole database. This paper proposes an incremental algorithm called IS-FUP which eliminates the rescanning by mining frequent itemsets only from new transactions and using this result to efficiently update the previous discovered frequent itemsets. It adopts the Fast Update (FUP) Algorithm [14] as an efficient update concept.

The remaining of this paper is organized as follows. Background and related work are given in Section 2. The share-frequent itemsets mining problem is described in Section 3. In Section 4, the proposed algorithm is described and some examples are given. The efficiency of IS-FUP is shown in Section 5 and conclusions are drawn in Section 6.

2 Related Work

2.1 Mining Share-Frequent Itemsets

In 1997, Carter et al. introduced the share-confidence model to discover useful knowledge about numerical attributes associated with items in a transaction [12]. This numerical attribute reflects all the profit, the cost of items, and associates to the transaction items. Several mining techniques have been proposed for efficiently discovering share-frequent itemsets. [15] and [11] proposed method named Zero pruning (ZP) and Zero subset pruning (ZSP) respectively that can extract the complete set of share-frequent itemsets from the transaction database using heuristic search method. However, they take an exponential increase of runtime as a minimum share threshold decreased. Some techniques were presented to mine all share-frequent itemsets, however, these methods cannot extract the complete set of share-frequent itemsets since their models do not satisfy the downward closure property. These include SIP [15], CAC [16] and IAB [11].

Later, the Fast Share Measure approach (ShFSM) was proposed to improve the previous approaches by using a property of level closure. However, this model does not hold the property of downward closure, therefore, the set of share-frequent itemsets cannot be discovered completely. After that, [17] proposed the Direct Candidates Generation(DCG) that relies on the downward-closure-property by using the transaction measure value of a pattern (definition 4) to decrease the number of useless candidates. This method adopts the level-wise candidate generation-and-test methodology that directly generates candidates without joining and pruning steps in each pass. Moreover, this method extracts the complete set of share-frequent pattern. However, the DCG requires more execution time to generate and test a large number of unwanted candidates in the mining process since it treats all *1*-items from the transaction database as *1*-candidate (C_1) for generating all *2*-candidates. That is, all *1*-candidates contain frequent itemsets and infrequent itemsets. For later round, the method produces the k-candidates (C_k) by joining the previous candidate (C_{k-1}) which has their transaction measure value beyond a certain threshold with a single pattern in C_1.

As can be seen in the problem mentioned above, reducing the computation time is an important issue of mining share-frequents patterns. [13], therefore, introduced an efficient method for Mining Complete Share-frequent Itemsets (MCShFI) to solve the weakness of the DCG solution. The MCShFI uses transaction measure value (definition 4) as an effective upper bound of each item to reduce the computation time by moving out a number of unwanted candidate at the first round. Furthermore, the generated candidate itemsets of MCShFI are created from the combination of only itemsets with their transaction measure value beyond a certain threshold. As a result, the MCShFI approach is more efficient than the DCG method in terms of execution time.

2.2 The Fast Update Concept

The number of transactions in a database is increasingly growing up; therefore, the evaluation of derived association rules must be reproduced. In fact, some association rules are newly generated while some other rules become obsolete [18]. Every time the new inserted transaction occurs in the database, in the traditional batch mining algorithms, it has to re-process the whole updated database. This process; therefore, consumes lots of execution time and additionally waste existing mined knowledge.

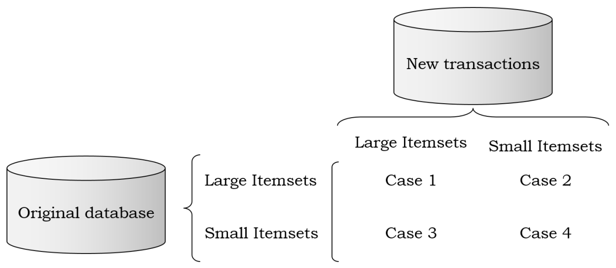


Figure 1: Four cases when new transactions are added into existing database

To reduce computation time, Cheung et al. [14] had introduced the Fast Update (FUP) algorithm. The FUP takes into account only mining results of an original database and new transactions. There are four cases under consideration as shown in Figure 1.

Case 1: An itemset is large both in an original database and in new transactions.
Case 2: An itemset is large in an original database but small in new transactions.
Case 3: An itemset is small in an original database but large in new transactions.
Case 4: An itemset is small both in an original database and in new transactions.

For Case 1, both the original database and the new transactions are large (frequent). Thus, the weighted average of the counts will be large. Identically, for Case 4, both small (infrequent) original database and small (infrequent) newly inserted transactions yield small weighted average of the counts. This implies that, both Case 1 and Case 4 will not influence the final frequent itemsets. Nevertheless, for Case 2 and Case 3, the weighted average of the counts can be small (infrequent), thus, affects the final frequent itemsets. Hence, to achieve the correct solution, the existing large (frequent) itemsets may be removed for Case 2 while it must be inserted for Case 3.

3 Problem Statement

In this section, the notation for the IS-FUP algorithm is described. Then the basic definitions which similar to the previous works [19, 17, 20, 13] are presented.

3.1 Notation

I	a set of m items, $I = \{i_1, i_2, \ldots, i_j, \ldots, i_m\}$ be a set of literals with counting attributes, where $i_1, i_2, \ldots, i_j, \ldots, i_m$ are called *item*.
X	an itemset, where $X \subseteq I$.
DB	a transaction database, $DB = \{T_1, T_2, \ldots, T_n\}$ be a set of transactions, where n is the number of transactions in DB and each transaction $T_q \in DB$ and also $T_q \subseteq I$, $(1 \leq q \leq n)$. The type of DB can be divided into two groups: O and $db+$.
O	an original transaction database.
$db+$	a set of the newly inserted transactions.
U	an entire updated database, $O \cup db+$
C_k	a candidate share-frequent itemsets of size k
$minLMV^{db+}$	The minimum local measure value in new transactions $db+$
$minLMV^{U}$	The minimum local measure value in the entire updated database U

TMV^O	The total measure value of the original transaction database O
TMV^{db+}	The total measure value of the newly inserted transactions $db+$
TMV^U	The total measure value of the entire updated database U
$Htmv^O$	a set of high transaction measure value itemsets in the original transaction database O
$Htmv^U$	a set of high transaction measure value itemsets in the entire updated database U
$Htmv^O_k$	a set of high transaction measure value k-itemsets in the original transaction database O
$Htmv^{db+}_k$	a set of high transaction measure value k-itemsets in the newly inserted transactions $db+$
$Htmv^U_k$	a set of high transaction measure value k-itemsets in the entire updated transactions U
$tmv^O(X)$	a transaction measure value of an itemset X of all transactions in the original transaction database O
$tmv^{db+}(X)$	a transaction measure value of an itemset X of all transactions in the newly inserted transactions $db+$
$tmv^U(X)$	a transaction measure value of an itemset X of all transactions in the entire updated transactions U
$lmv^O(X)$	a local measure value of all transactions in the original transaction database O that containing X
$lmv^{db+}(X)$	a local measure value of all transactions in the newly inserted transactions $db+$ that containing X
$lmv^U(X)$	a local measure value of all transactions in the entire updated transactions U that containing X

Table 1: Example of a transaction database with counting

	TID	Transaction	Count
Original Database (O)	T_1	{A,B,F,G,H}	{2,1,2,1,2}
	T_2	{A,C,E}	{5,3,3}
	T_3	{B,C,H}	{3,2,2}
	T_4	{C}	{5}
	T_5	{C,E,F,G}	{3,1,2,1}
The newly inserted transactions ($db+$)	T_6	{A,C,D,E,F}	{4,2,1,1,5}
	T_7	{A,D}	{1,2}
	T_8	{B,C,H}	{4,3,2}
	T_9	{A,C,E}	{2,1,1}

3.2 Basic Definitions

Definition 3.1. The measure value of item i_p in transaction T_q represents a numerical value associated with item i_p in T_q denoted by,

$$mv(i_p, T_q). \tag{3.1}$$

For example, in Table 1, $mv(\{C\}, T_2)$ =3.

Definition 3.2. The itemset measure value $imv(X, T_q)$ is the *total measure value* of itemset X in transaction T_q defined as follow,

$$imv(X, T_q) = \sum_{i_p \in X} mv(i_p, T_q). \tag{3.2}$$

For instance in Table 1, $imv(\{AC\}, T_2) = mv(\{A\}, T_2) + mv(\{C\}, T_2)$ = 5+3 = 8.

Definition 3.3. A transaction measure value of transaction T_q, $tmv(T_q)$ is defined by,

$$tmv(T_q) = \sum_{i_p \in T_q} mv(i_p, T_q). \tag{3.3}$$

For example in Table 1, $tmv(T_2) = mv(\{A\}, T_2) + mv(\{C\}, T_2) + mv(\{E\}, T_2)$ = 5+3+3 =11.

Definition 3.4. A transaction measure value of itemset X denoted by $tmv(X)$ describes a total value of all transactions containing X. It is defined by,

$$tmv(X) = \sum_{X \subseteq T_q \in DB_X} tmv(T_q) \tag{3.4}$$

where DB_X is a set of transaction that contains itemset X.

For example, $tmv(\{AC\}) = tmv(T_2) + tmv(T_6) + tmv(T_9)$ = 11 + 13 + 4 = 28, in Table 1.

Definition 3.5. A local measure value of itemset X is given by,

$$lmv(X) = \sum_{T_q \in DB_X} \sum_{i_p \in X} imv(i_p, T_q). \tag{3.5}$$

For example in Table 1, $lmv(\{AC\}) = imv(\{AC\}, T_2) + imv(\{AC\}, T_6)$
$+ imv(\{AC\}, T_9) = 8 + 6 + 3 = 17.$

Definition 3.6. A total measure value of the entire updated transaction, $TMV(U)$, represents the summation of all the transaction measure values which is given by,

$$TMV(DB) = \sum_{T_q \in DB} \sum_{i_p \in T_q} mv(i_p, T_q). \tag{3.6}$$

For example in Table 1, $TMV(O \cup db+) = 67$.

Definition 3.7. A share value of itemset X, denoted by $SH(X)$, is a ratio of the local measure value of X to the total measure value of DB, as calculated by,

$$SH(X) = \frac{lmv(x)}{TMV(U)}. \tag{3.7}$$

For example, $SH(\{AC\}) = = 17/67 = 0.2537$.

Definition 3.8. Given that a minimum share ($minShare$) is a minimum share threshold, if the share value of itemset X, $SH(X)$, is greater than $minShare$ then we can conclude that itemset X is a share-frequent itemset.

For the example database, if $minShare$ is 0.25, $\{AC\}$ is a share-frequent itemset, as $SH(\{AC\}) = 0.2537$.

Definition 3.9. A minimum local measure value ($minLMV$) can be defined by,

$$minLMV = ceiling(minShare \times TMV(O \cup db+)) \tag{3.8}$$

Therefore, in Table 1, $minLMV = ceiling(0.25 \times 67) = 17$. For any itemset X, if $lmv(X) \geq minLMV$, then itemset X is a share-frequent itemset.

From the property of the downward closure (also called anti-monotonicity) of the support measure, it can be seen that this property can maintain in the share-frequent patterns mining by using the *tmv* value of itemset X (Definition 4). For any itemset X, if $tmv(X)$ is less than $minLMV$, all supersets of X can be pruned (including X) immediately without further consideration. For example, let minimum share threshold is 0.25, then $tmv(\{D\}) = 16 < minLMV = 17$. According to the property of downward closure, all supersets of $\{D\}$ are not generated as a candidate patterns. Therefore, we can prune $\{D\}$ at the early step.

4 A Proposed Incremental Algorithm

We propose an efficient algorithm for incremental share-frequent itemsets mining based on a Patternset Table Knowledge (PSTable knowledge). The proposed algorithm includes two processes: maintenance a PSTable knowledge, and the IS-FUP algorithm.

4.1 Maintenance a PSTable Knowledge

A PSTable knowledge is a set of itemsets with their count information and it is im-proved by the concept of a bittable structure [21, 22, 23, 20, 13]. This structure can avoid candidate generation by aggregate the transactions that have similar itemsets. In addition, the PSTable knowledge will be constructed once and it accommodates modified data in the future (do not need to rebuild). The following describes operation of this algorithm.

For creating a PSTable knowledge and inserting new transactions, the algorithm reads the data from each incoming transaction. After that, it checks the itemsets of the transaction whether the itemset is already available in the PSTable or not. If it exists, the quantity of the transaction (total count) is summed with the new entries to gain new value of total count, and the itemset count is incremented by 1. If the itemset is new, it will be inserted into the PSTable with the quantity of the transaction and set the itemset count to be 1. This process is repeated for all transactions.

For maintenance the PSTable knowledge (removing transactions), the algorithm reads the itemset from each removing transaction. After that, the itemset is searched from the PSTable knowledge, then the quantity of each item of the itemset and the quantity of the transaction (total count) are updated and also decrease the itemset count by 1. For the modified existing transactions case, the algorithm starts with ascendant sorting items in the PSTable knowledge. Next, the algorithm discovers the modified transaction and then the quantity of each item of the itemset and the quantity of the transaction (total count) are updated. All the mentioned steps are repeated for all of the transactions in database. For instance, the original transaction database in Table 1 was performed and the results shown in Table 2.

Table 2: The PSTable after the algorithm executed all transactions

PID	A	B	C	D	E	F	G	H	Total Count	Itemset Count
1	2	1	0	0	0	2	1	2	8	1
2	7	0	4	0	4	0	0	0	15	2
3	0	7	5	0	0	0	0	4	16	2
4	0	0	5	0	0	0	0	0	5	1
5	0	0	3	0	1	2	1	0	7	1
6	4	0	2	1	1	5	0	0	13	1
7	1	0	0	2	0	0	0	0	3	1

4.2 Incremental Share-Frequent Itemset Mining

4.2.1 The Concept of the IS-FUP Algorithm

We propose an efficient algorithm that handles insertion of transactions (named IS-FUP algorithm). It is extended from the MCShFI algorithm [13] in a batch way for discovering complete share-frequent itemsets from a PSTable knowledge based on the FUP concept [14]. Assuming that a PSTable knowledge is built in advance from the original database before the new transactions come and the large/frequent itemsets have been already derived from the initial database. In IS-FUP algorithm, itemsets are separated into four scenarios according to whether they are large (frequent) or small (infrequent) itemsets for both in the original database and in the new transactions. To determine whether the itemset is large or small, the value of *tmv* is calculated (by definition 3.4). As mentioned earlier, this value contains the properties of downward closure. If the value of *tmv* of any itemset is larger than or equal to the minimum local measure value ($minLMV$), it is in the set of high transaction measure value itemsets and in the group of large itemsets. Otherwise, it is in small itemsets.

Considering an original database and a new transaction that will be inserted, the following four cases and their results in given in Table 3.

Table 3: Example of a transaction database with counting

Cases : Original-New transactions	Results
Case 1: Large itemset - Large itemset	Always large itemset
Case 2: Large itemset - Small itemset	Determined by rescanning the newly added transactions
Case 3: Small itemset - Large itemset	Determined by rescanning the PSTable
Case 4: Small itemset - Small itemset	Always small itemset

For the newly inserted transactions, when some new records are inserted into the original database: In Case 1 and 4, itemsets do not change the final share-frequent itemsets and thus these two cases are extreme. In Case 2, itemsets may be removed from existing large itemsets, and itemsets in Case 3 may add new large itemsets. For the extreme cases 1 and 4, the proposed approach in share-frequent itemset mining can be mathematically proved and given below whereas the other two cases 2 and 3 can be shown by experiments using the IS-FUP algorithm. Also, a numerical example is later provided in the next sections illustrating all the four cases.

Theorem 4.1. *If an itemset X is large in both the original database and the newly inserted transactions then the itemset X is large in the entire updated database (Case 1).*

Proof. Assume, by contradiction, that itemset X is not large in the entire updated database, i.e.,

$$tmv(X)^{O} + tmv(X)^{db+} < minShare \times (TMV^{O} + TMV^{db+}) \tag{4.1}$$

and so

$$tmv(X)^{O} + tmv(X)^{db+} - minShare \times TMV^{db+} < minShare \times TMV^{O} \tag{4.2}$$

By hypothesis, an itemset X is large in both the original database and the newly inserted transactions

it is obtained that

$$tmv(X)^{O} \geq minLMV^{O}(= minShare \times TMV^{O}) \tag{4.3}$$

and

$$tmv(X)^{db+} \geq minLMV^{db+}(= minShare \times TMV^{db+}). \tag{4.4}$$

Since

$$tmv(X)^{O} < tmv(X)^{O} + tmv(X)^{db+} - minShare \times TMV^{db+}. \tag{4.5}$$

From (4.2) and (4.5), we can conclude that

$$tmv(X)^{O} < minShare \times TMV^{O} \tag{4.6}$$

(4.6) contradicts (4.3), hence the proof is complete. □

Theorem 4.2. *If an itemset X is small in both the original database and the newly inserted transactions then the itemset X is small in the entire updated database (Case 4).*

Proof. Assume, by contradiction, that itemset X is not small in the entire updated database, i.e.,

$$tmv(X)^{O} + tmv(X)^{db+} \geq minShare \times (TMV^{O} + TMV^{db+}) \tag{4.7}$$

and so

$$tmv(X)^{O} + tmv(X)^{db+} - minShare \times TMV^{db+} \geq minShare \times TMV^{O}. \tag{4.8}$$

By hypothesis, an itemset X is small in both the original database and the newly inserted transactions

it is obtained that

$$tmv(X)^{O} < minLMV^{O}(= minShare \times TMV^{O}) \tag{4.9}$$

and

$$tmv(X)^{db+} < minLMV^{db+}(= minShare \times TMV^{db+}). \tag{4.10}$$

Since

$$tmv(X)^O > tmv(X)^O + tmv(X)^{db+} - minshare \times TMV^{db+}. \tag{4.11}$$

From (4.8) and (4.11), we can conclude that

$$tmv(X)^O \geq minShare \times TMV^O \tag{4.12}$$

(4.9) contradicts (4.12), hence the proof is complete. □

4.2.2 The IS-FUP Algorithm

The detail of the proposed algorithm are described step by step below.

Input: 1) A PSTable knowledge (O) 2) The newly inserted transactions ($db+$) in which each transaction includes items and their quantities 3) A $Htmv^O$, which is stored a set of high transaction measure value of the original transaction database 4) The total measure value of the original transaction database, TMV^O 5) The minimum share threshold, $minShare$

Output: A $Htmv^U$, that is a set of high transaction measure value of the entire updated database.

Step 1: Calculate the total measure value of newly inserted transactions using Definition 3.6 denoted by TMV^{db+}. After that, TMV^U is calculated ($TMV^U = TMV^O + TMV^{db+}$). Next, the algorithm computes $minLMV^U$ using Definition 3.9 and set $k = 1$, where k is used for recording the number of items in the itemsets currently being processed.

Step 2: The algorithm generates the candidated k-itemsets then computes their transaction measure value $tmv^{db+}(X)$ and local measure value $lmv^{db+}(X)$ from the new transactions using Definition 3.4 and 3.5 respectively. Checks whether the $tmv^{db+}(X)$ of each k-itemset X is larger than or equal to $minLMV^{db+}$. If $tmv^{db+}(X)$ satisfies the above condition, put X in the set of high transaction measure value k-itemset in the newly inserted transactions, $Htmv_k^{db+}$.

Step 3: For each k-itemset X, if it is in the set of high transaction measure value in the new transactions and it also appears in the set of high transaction measure value in the original database , do the following substeps (Case 1):

Step 3.1: The algorithm will keep the itemset X into both the $Htmv_k^U$ and C_k and also removes it from the $Htmv_k^O$.

Step 4: For each k-itemset X, if it is in the set of high transaction measure value in the new transactions ($Htmv_k^{db+}$) but it does not appear in the set of high transaction measure value in the original database $Htmv_k^O$, do the following substeps (Case 3):

Step 4.1: Rescans the original database to determine the $tmv^O(X)$ and $lmv^O(X)$ of k-itemset X. After that, set both $tmv^U(X)$ $(= tmv^O(X) + tmv^{db+}(X))$ and $lmv^U(X)$ $(= lmv^O(X) + lmv^{db+}(X))$.

Step 4.2: Checks whether the $tmv^U(X)$ is greater than or equal to $minLMV^U$. If $tmv^U(X)$ satisfies the above condition, stores it to the C_k and also adds into the $(Htmv^U_k)$.

Step 5: For each k-itemset X, if it appears in the set of high transaction measure value in the original database $(Htmv^O_k)$ but does not appear in the set of high share-frequent itemsets from the new transactions $(Htmv^{db+}_k)$, do the following substeps (Case 2):

Step 5.1: The algorithm reads all k-itemsets in the $Htmv^O_k$. After that, compute the tmv^{db+} and the lmv^{db+} from the PSTable knowledge and then calculates both $tmv^U(X)$ $(= tmv^O(X)+ tmv^{db+}(X))$ and $lmv^U(X)$ $(= lmv^O(X) + lmv^{db+}(X))$.

Step 5.2: If the $tmv^U(X)$ is larger than or equal to $minLMV^U$, stores X to the C_k and inserts to the $Htmv^U_k$ and also removes it from the $Htmv^O_k$.

Step 6: Increase k by one and then the algorithm generates the candidate k-itemsets from the variable C_{k-1}.

Step 7: Repeat Steps 2 - 6 until no new candidate itemsets is generated and the $Htmv^O$ is empty.

4.2.3 A Numerical Example

This subsection shows how the IS-FUP algorithm can be used for finding out the share-frequent itemsets from the new transactions. Assume that the transactions $T_1 - T_5$ shown in Table 1 are the initial database and the transaction T_6 - T_9 are newly inserted transactions. Also assume that the minimum share threshold ($minShare$) is set at 0.25 which is 25% of total quantity. The set of high transaction measure value is built in advance from the original database as shown in Table 4.

Step 1: : The IS-FUP algorithm calculates the initial variables at first i.e. TMV^{db+}, $minLMV^{db+}$, TMV^U and $minLMV^U$ respectively, as shown in Table 5. Then, k is set at 1.

Step 2: The k-itemsets in the newly inserted transactions for $\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{F\} and \{H\}$ are read into the algorithm and then calculate their transaction measure value (tmv^{db+}) and the local measure value (lmv^{db+}) from the new transactions. Take k-itemset $\{A\}$ as an example to illustrate the process. The k-itemset $\{A\}$ appears in the new transactions

Table 4: The $Htmv^O$

Itemset X	$tmv^O(X)$	$lmv^O(X)$
$\{A\}$	19	7
$\{B\}$	15	4
$\{C\}$	30	13
$\{E\}$	18	4
$\{F\}$	15	4
$\{G\}$	15	2
$\{H\}$	15	4
$\{A, C\}$	11	8
$\{A, E\}$	11	8
$\{B, H\}$	15	8
$\{C, E\}$	18	10
$\{F, G\}$	15	6
$\{A, C, E\}$	11	11

T_6, T_7 and T_9 and then the algorithm calculates the $tmv^{db+}(\{A\})$ as (13+3+4), which is 20 and computes the $lmv^{db+}(\{A\})$ as (4+1+2), which is 7 respectively. The other k-itemsets are calculated in the same way and the result is shown in Table 6. After that, checks whether the $tmv^{db+}(X)$ of each k-itemset is larger than or equal to $minLMV^{db+}$. In this example, all k-itemsets satisfy the condition and then put them in the set of high transaction measure value k-itemsets in the new transaction, ($Htmv_k^{db+}$).

Table 5: The initial variables for the algorithm

	TMV	$minLMV$
O	38	10
$db+$	29	7
U	67	17

Step 3: For each k-itemset X, it is in both the set of high transaction measure value in the new transactions ($Htmv_k^{db+}$) and the set of high transaction measure value in the original database ($Htmv_k^O$), do the following substeps (Case 1):

Step 3.1: The k-itemsets are in the $Htmv_k^O$ and the $Htmv_k^{db+}$ which are as follows $\{A\}, \{B\}, \{C\}, \{E\}, \{F\} and \{H\}$. These itemsets are recorded to the $Htmv_k^U$ and are removed from the $Htmv_k^O$ and also kept as candidate share-frequent itemsets, C_k. The result is shown in Table 7 and 8.

Table 6: all k-itemsets are calculated from the new transactions

k-itemset X	$tmv^{db+}(X)$	$lmv^{db+}(X)$
$\{A\}$	20	7
$\{B\}$	9	4
$\{C\}$	26	6
$\{D\}$	16	3
$\{E\}$	17	2
$\{F\}$	13	5
$\{H\}$	9	2

Table 7: After k-itemsets (Case 1) are added in the $Htmv^U$

k-itemset X	$tmv^U(X)$	$lmv^U(X)$
$\{A\}$	39	14
$\{B\}$	24	8
$\{C\}$	56	19
$\{E\}$	35	6
$\{F\}$	28	9
$\{H\}$	24	6

Table 8: After k-itemsets (Case 1) are deleted from the $Htmv^O$

k-itemset X	$tmv^O(X)$	$lmv^O(X)$
$\{G\}$	15	2
$\{A, C\}$	11	8
$\{A, E\}$	11	8
$\{B, H\}$	15	8
$\{C, E\}$	18	10
$\{F, G\}$	15	6
$\{A, C, E\}$	11	11

Step 4: For each k-itemset X, it is in the set of high transaction measure value in the newly inserted transactions, $Htmv_k^{db+}$, but it does not in the set of high transaction measure value in the original database, $Htmv_k^O$, do the following substeps (Case 3):

Step 4.1: The k-itemset X appears in the $Htmv_k^{db+}$ but it does not appear in the $Htmv_k^O$ which is $\{D\}$. The IS-FUP algorithm rescans to calculate the $tmv^O(X)$ and the $lmv^O(X)$ from the PSTable knowledge. Then, the algorithm computes the $tmv^U(X)$ and the $lmv^U(X)$ respectively. Let takes the k-itemset $\{D\}$ as an example to illustrate the process. The tmv^{db+} of $\{D\}$ and the lmv^{db+} of $\{D\}$ in the new transactions are 16 and 3 respectively as shown in Table 6. The tmv^O of $\{D\}$ and the lmv^O of $\{D\}$ in the original database are both 0, which are shown in Table 9. The updated transaction measure value and The updated local measure value for the updated database of $\{D\}$ are calculated as $tmv^U(\{D\})$ (=0+16), which is 16 and $lmv^U(\{D\})$ (=0+3), which is 3 respectively.

Step 4.2: Check whether the tmv^U of k-itemset X is larger than or equal to the $minLMV^U$. If it satisfies the condition, inserts X into C_k and also stores it to the $Htmv_k^U$. For instance, the k-itemset $\{D\}$ does not satisfy the both of above conditions.

Step 5: For each k-itemset X, it does not in the set of high transaction measure value in the new transactions ($Htmv_k^{db+}$), but it appears in the set of high transaction measure value in the original database ($Htmv_k^O$), do the following substeps (Case 2):

Step 5.1: The IS-FUP algorithm reads all k-itemsets from the $Htmv_k^O$ in Table 8, which is $\{G\}$. The tmv^{db+} and the lmv^{db+} of k-itemset $\{G\}$ in the new transactions are calculated, which are both 0. Then, the updated transaction measure value and the updated local measure value for the updated database of $\{G\}$ are computed as $tmv^U(\{G\})$ (=0+15), which is 15 and $lmv^U(\{G\})$ (=0+2), which is 2 respectively. The result is shown in Table 10.

Step 5.2: Check whether the tmv^U of $\{G\}$ is larger than or equal to the $minLMV^U$. If it satisfies the condition, stores it into both C_k and the $Htmv_k^U$ and also removes it from the $Htmv_k^O$. For instance, the k-itemset $\{G\}$ does not satisfy both of the above condition.

Table 9: After all k-itemsets in case 3 are re-calculated.

k-Itemset X	$tmv^O(X)$	$lmv^O(X)$	$tmv^U(X)$	$lmv^U(X)$
$\{D\}$	0	0	16	3

Step 6: Set $k = 2$ $(k+1)$. The candidate k-itemsets are generated from C_{k-1}, which are $\{A,B\}$, $\{A,C\}$, $\{A,E\}$, $\{A,F\}$, $\{A,H\}$, $\{B,C\}$, $\{B,E\}$, $\{B,F\}$, $\{B,H\}$, $\{C,E\}$, $\{C,F\}$, $\{C,H\}$, $\{E,F\}$, $\{E,H\}$, $\{F,G\}$ and $\{F,H\}$.

Table 10: After all k-itemsets in case 2 are re-calculated.

k-Itemset X	$tmv^{db+}(X)$	$lmv^{db+}(X)$	$tmv^{U}(X)$	$lmv^{U}(X)$
$\{G\}$	0	0	15	2

Step 7: Steps 2 - 6 are then repeated until no candidate itemset is generated and the $Htmv^{O}$ is empty as shown in Table 12. The IS-FUP algorithm returns a set of high transaction measure value in the entire updated database, $Htmv^{U}$, that contains 3 share-frequent itemsets i.e., $\{C\}$, $\{A,C\}$ and $\{A,C,E\}$ that shown in Table 11.

Table 11: The $Htmv^{U}$ after the IS-FUP completed

Itemset X	$tmv^{U}(X)$	$lmv^{U}(X)$	$SH(X)$
$\{A\}$	39	14	0.2090
$\{B\}$	24	8	0.1194
{C}	**56**	**19**	**0.2836**
$\{E\}$	35	6	0.0896
$\{F\}$	28	9	0.1343
$\{H\}$	24	6	0.0896
{A,C}	**28**	**17**	**0.2537**
$\{A,E\}$	28	16	0.2388
$\{A,F\}$	21	13	0.1940
$\{B,H\}$	24	14	0.2090
$\{C,E\}$	35	15	0.2239
$\{C,F\}$	20	12	0.1791
$\{E,F\}$	20	9	0.1343
{A,C,E}	**28**	**22**	**0.3284**
$\{C,E,F\}$	20	14	0.2090

Table 12: The $Htmv^{O}$ after the proposed algorithm completed

Itemset X	$tmv^{O}(X)$	$lmv^{O}(X)$
– Empty –		

5 Algorithm Efficiency

In this section, we demonstrate the efficiency of the algorithm by applying the IS-FUP to two IBM synthetic datasets (T10I4D100K and T40I10D100K) and a real-life dataset (pumsb*). They are acquired from frequent itemset mining dataset repository page (http://fimi.ua.ac.be/data/). As like the performance evaluation of the previous share-frequent itemset mining algorithm [17, 13], we modified these datasets by creating random numbers for the quantity of each item in each transaction, ranging from 1 to 10. The characteristics of the datasets are shown in Table 13, $|D|$ is the total number of transaction in a dataset, $|N|$ is the number of distinct pattern in the dataset and T_{avg} is the average size of transaction. Our algorithms were written in Microsoft C++ 2010 and carried out on a PC with 1.40 GHz CPU with 4 GB main memory. The algorithm efficiency is determined by the processing time and compared to MCShFI [13] in the batch way.

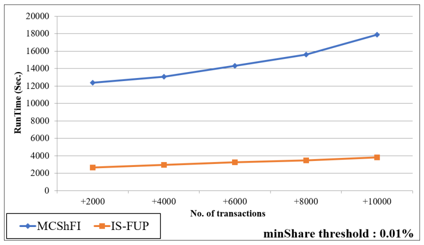
Table 13: Characteristics of the experiment datasets

Dataset	Size (MB)	$\|D\|$	$\|N\|$	T_{avg}
T10I4D100K	3.83	100,000	870	10.1
T40I10D100K	14.7	100,000	942	40.5
pumsb*	10.7	49,046	2,088	50.5

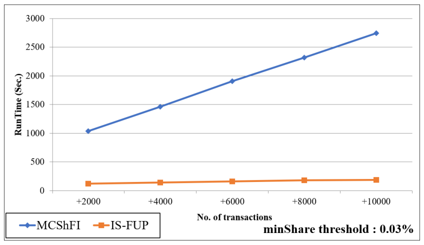
5.1 The IS-FUP Algorithm on Synthetic Datasets

In the first experiment, we have tested the effectiveness of the IS-FUP algorithm in incremental mining and the batch MCShFI with T10I4D100K. The minimum share threshold was set at 0.01% and 0.03% respectively. A number of 70,000 transactions were created to initially mine the share-frequent itemsets with their quantities. Next, the numbers of inserted transactions were varied from 2,000 to 10,000 and the results are shown in Figures 2 (a)-(b). For T40I10D100K, experiments were conducted in similar to the previous experiment but the minimum share threshold was set at 0.1% and 0.3% respectively. The experimental results are illustrated in Figures 3 (a)-(b).

It can be easily observed from these figures that the both algorithms required more execution time when the certain threshold lowers. This is reasonable because when the minimum share value becomes minor, the algorithms have to generate more candidates in the mining process. Moreover, the runtime of the new solution is smaller than the MCShFI method in a batch way. This is because the MCShFI needs to update information on its data structure to meet updated transactions. After that, it has to mine the last updated database to get the final share-frequent itemsets. Therefore, this process must be performed every time whenever transactions are inserted. For the new solution, it takes advantages of the earlier result of share-frequent itemsets and does not re-mine the whole updated database. The

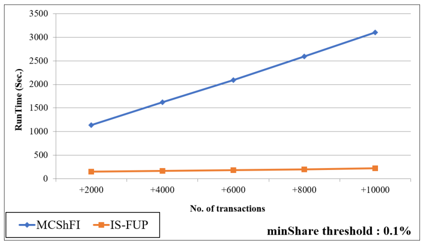


(a)

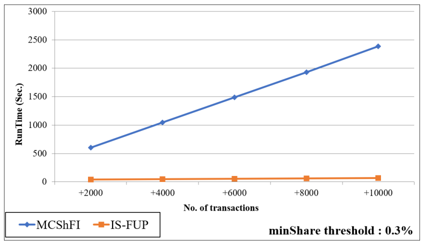


(b)

Figure 2: Effects of numbers of inserted transactions on runtime for the T10I4D100K dataset



(a)



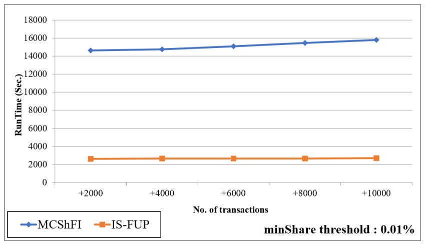
(b)

Figure 3: Effects of numbers of inserted transactions on runtime for the T40I10D100K dataset

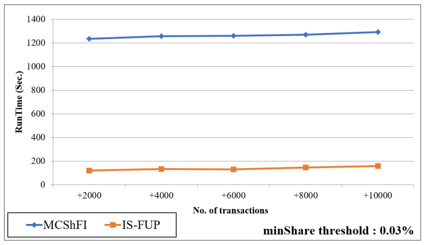
proposed method mines only the newly inserted transactions and rescans as necessary to re-calculate itemsets from the original database only if it is in Case 3 as mentioned in Section 4.2.1.

In the next experiments, the first 90,000 transactions were discovered from the T10I4D100K dataset to construct initial share-frequent itemsets with their quantities. The minimum share threshold was set at 0.01% and 0.03% respectively. After that, each 2,000 transactions, which were used as new inserted transactions, were performed again with the same thresholds. The results are shown in Figures 4 (a)-(b). Similar to above experiments for T40I10D100K, the minimum share threshold was set at 0.1% and 0.3% respectively. The experimental results are illustrated in Figures 5 (a)-(b).

Figure 6 (a) is then made to show the runtimes of the two algorithms on the T10I4D100K dataset for different minimum share threshold. The thresholds range of 0.005 to 0.05 percent were used here. The first 90,000 transactions were mined from the T10I4D100K dataset to create initial share-frequent itemsets with their quantities. After that, the next 10,000 transactions, which were used as new inserted transactions, were generated with the same thresholds. In the similar way



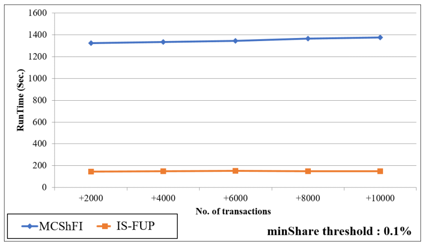
(a)



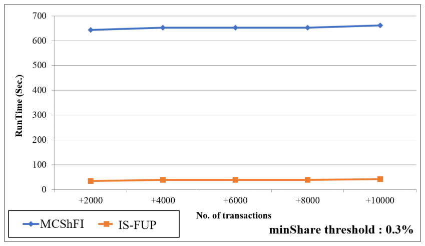
(b)

Figure 4: Effects of numbers of inserted transactions on runtime for the T10I4D100K dataset

for T40I10D100K, the minimum share thresholds were set from 0.03% to 0.3%. The experimental results illustrated in Figure 6 (b).

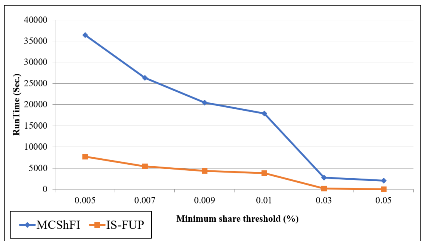


(a)

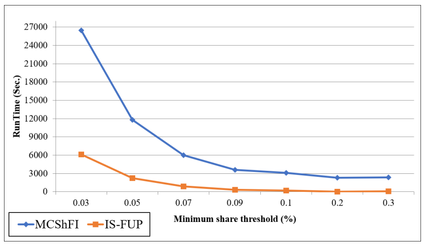


(b)

Figure 5: Effects of numbers of inserted transactions on runtime for the T40I10D100K dataset



(a)

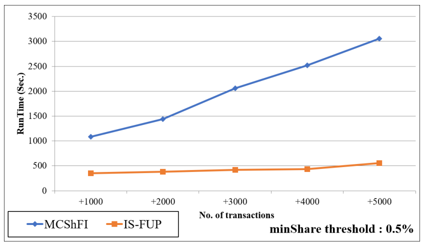


(b)

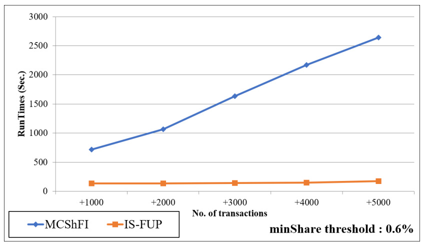
Figure 6: Effects of different minimum share thresholds on runtime for the T10I4D100K and the T40I10D100K datasets

5.2 The IS-FUP Algorithm on Real-Life Dataset

At first, we constructed to initially mine the share-frequent itemsets with their quantities from the pumsb* dataset for 34,046 transactions with three different minimum share thresholds. The thresholds were set at 0.5% and 0.6% respectively. In addition, various number of transactions of 1,000, 2,000, 3,000, 4,000 and 5,000 were added and the results are shown in Figures 7 (a)-(b). As shown in these figures, the batch MCShFI algorithm wasted a lot of time to mine all of the whole updated transactions whenever transactions were inserted. The new solution does not re-mine the last updated database and hence takes advantages of the previous results of share-frequent itemsets.



(a)

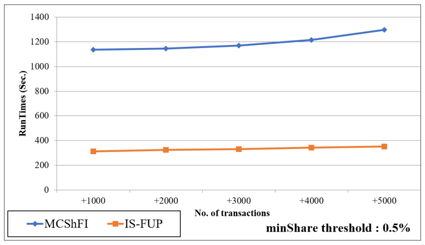


(b)

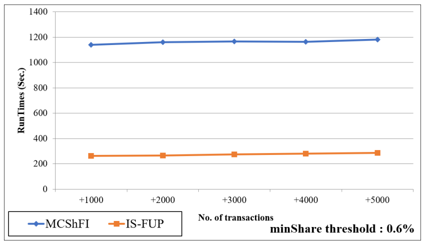
Figure 7: Effects of numbers of inserted transactions on runtime for the pumsb* dataset

In the next experiments, the first 44,046 transactions were searched from the pumsb* to create initial share-frequent itemsets with their quantities. The minimum share threshold was set at 0.5% and 0.6% respectively. After that, each 1,000 transactions, which were used as new inserted transactions were generated again with the same threshold. The results are shown in Figures 8 (a)-(b).

The last experiment was then conducted to compare the runtimes of both algorithms on the pumsb* real-life dataset for different numbers of minimum share threshold. Additionally, the minimum share threshold was considered to be varied from 0.45% to 0.6%. A number of 44,046 transactions were extracted to construct initial share-frequent itemsets with their quantities. After that, the next 5,000 transactions were added and, with the same threshold value, the runtime was measured. The result is as shown in Figure 9. The computation time of the new solution is lower than the MCShFI in batch mode for handling transaction insertion. The reason is as same as already mentioned in the previous experiment. In conclusion, our proposed algorithm in incremental share-frequent itemset mining (the IS-FUP algorithm) outperforms the batch MCShFI method to maintain the updated database.



(a)



(b)

Figure 8: Effects of numbers of inserted transactions on runtime for the pumsb* dataset

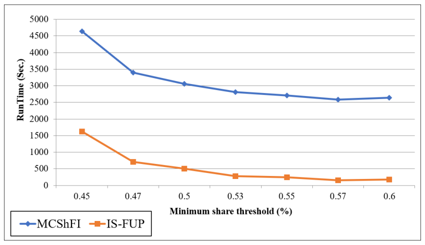


Figure 9: Effects of different minimum share thresholds on runtime for the pumsb* dataset

6 Conclusions

In the past, the efficient algorithm for mining complete share-frequent itemsets (called MCShFI) was proposed to discover complete share-frequent itemsets based on a batch approach. In this paper, an incremental approach to the problem is proposed with the fast update concept to improve mining execution time. Among the four updated cases under consideration of the fast update concept, we mathematically show that two of them do not affect mining result. This even more helps in improving execution time of the proposed algorithm. Extensive performance analyses show that execution time of the incremental algorithm outperforms of the batch one on both synthetic datasets and real-life datasets.

References

[1] R. Agrawal, T. Imielinski, A. Swami, Mining association rule between sets of items in large database, In Proceedings of ACM SIGMOD on Management

of Data (1993), 207-216.

[2] R. Agarwal, R. Srikant, Fast algorithms for mining association rules in large databases, In Proceedings of 20th International Conference on Very Large Data Bases (1994), 487-499.

[3] J.S. Park, M.S. Chen, P.S. Yu, Using a Hash-Based method with transaction trimming for mining association rules, IEEE Transactions on Knowledge and Data Engineering 9 (1997), 813-825.

[4] S. Brin, R. Motwani, J.D. Ullman, S. Tsur, Dynamic itemset counting and implication rules for market basket data, In Proceedings of the ACM SIGMOD on Management of Data (1997), 255-264.

[5] E. Houda, E.F. Mohamed, E.M. Mohammed, A novel approach for mining frequent itemsets: AprioriMin, In Proceedings of 4th IEEE International Colloquium on Information Science and Technology (2016), 286-289.

[6] D.P. Shubhangi, R.D. Ratnadeep, K.D. Kirange, Adaptive Apriori Algorithm for frequent itemset mining, In Proceedings of International Conference System Modeling & Advancement in Research Trends (2016), 7-13.

[7] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, In Proceedings of the ACM SIGMOD on Management of Data (2000), 1-12.

[8] J. Pei, J. Han, H. Lu, Hmine: Hyper-structure mining of frequent patterns in large database, In Proceedings of International Conference on Data Mining (2001), 441-448.

[9] R. Agarwal, C. Aggarwal, V.V.V. Prasad, A tree projection algorithm for generation of frequent itemsets, Journal of Parallel and Distributed 61 (2001) 350-371.

[10] M. El-Hajj, O.R. Zaiane, COFI approach for mining frequent itemsets revisited, In Proceeding of the ACM SIGMOD on Data Mining and Knowledge Discovery (2004), 70-75.

[11] B. Barber, H.J. Hamilton, Extracting share frequent itemset with infrequent subsets, Data mining and Knowledge Discovery 7 (2003) 153-185.

[12] C.L. Carter, H.J. Hamilton, N. Cercone, Share based measures for itemsets, In Proceedings of the 1st European Symposium on Principles of Data Mining and Knowledge Discovery (1997), 14-24.

[13] C. Nawapornanan, V. Boonjing, An efficient algorithm for mining complete share-frequent itemsets using BitTable and heuristics, In Proceedings of International Conference on Machine Learning and Cybernetics (2012), 96-101.

[14] D.W. Cheung, J. Han, V. Ng, C.Y. Wong, Maintenance of discovered association rules in large databases: An incremental updating technique, In Proceedings of International Conference on Data Engineering (1996), 106-114.

[15] B. Barber, H.J. Hamilton, Algorithms for mining share frequent itemsets containing infrequent subsets, In Proceedings of Fourth European Conference on Principles of Knowledge Discovery in Database (2000), 316-324.

[16] B. Barber, H.J. Hamilton, Parametric algorithm for mining share frequent itemsets, Journal of Intelligent Information Systems 16 (2001) 277-293.

[17] Y.-C. Li, J.-S. Yeh, C.-C. Chang, Direct candidates generation: a novel algorithm for discovering complete share-frequent itemsets, In Proceedings of International Conference of Fuzzy Systems and Knowledge Discovery (2005), 551-560.

[18] T.P. Hong, C.W. Lin, Y.L. Wu, Incrementally fast updated frequent pattern trees, Expert Systems with Applications 34 (2008) 2424-2435.

[19] Y.-C. Li, J.-S. Yeh, C.-C. Chang, A fast algorithm for mining share-frequent itemsets, In Proceedings of International Conference on Asia-Pacific Web (2005), 417-428.

[20] C. Nawapornanan, V. Boonjing, A new share frequent itemsets mining using incremental BitTable knowledge, In Proceedings of 5th International Conference on Computer Sciences and Convergence Information Technology (2011), 358-362.

[21] J. Dong, M. Han, An efficient mining frequent itemsets algorithm, Knowledge-Based Systems. 20 (2007) 329-335.

[22] H. Jin, A counting mining algorithm of maximum frequent itemset based on matrix, In Proceedings of the 7th International Conference on Fuzzy Systems and Knowledge Discovery (2010), 1418-1422.

[23] N. Khare, N. Adlakha, K.R. Paradasani, An algorithm for mining multidimensional association rules using boolean matrix, in Proceedings of International Conference on Recent Trends In Information, Telecommunication and Computing (2010), 95-99.

(Received 28 September 2017)
(Accepted 26 April 2018)