EFFICIENT ALGORITHMS FOR MINING SHARE-FREQUENT ITEMSETS

Yu-Chiang Li[1] Jieh-Shan Yeh[2] Chin-Chen Chang[1, 3]
1.Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi 621, Taiwan
2.Department of Computer Science and Information Management, Providence University, Taichung 433, Taiwan
3.Department of Information Engineering and Computer Science, Feng Chia University, Taichung 407, Taiwan
Email: lyc@cs.ccu.edu.tw, jsyeh@pu.edu.tw, ccc@cs.ccu.edu.tw

ABSTRACT: Itemset share has been proposed to evaluate the significance of itemsets for mining association rules in databases. The Fast Share Measure (FSM) algorithm is one of the best algorithms to discover all share-frequent itemsets efficiently. However, FSM is fast only when dealing with small datasets. In this study, we shall propose a revised version of FSM, called the Enhanced FSM (EFSM) algorithm that speeds up the share-frequent itemsets discovery process. In addition, we shall also present two additional algorithms, SuFSM and ShFSM, developed from EFSM. SuFSM and ShFSM prune the candidates more efficiently than FSM and therefore can improve the performance significantly. Simulation results reveal that the proposed methods perform significantly better than ZSP and FSM, and the performance of ShFSM is the best.

Keywords: Data mining, Knowledge discovery, Association rules, Share measure

1 INTRODUCTION

Recent developments in information science have caused large scale data digitalization, swelling up digital databases and data warehouses. As a result, mechanisms to efficiently handle large quantities of chronological data and to expeditiously extract useful knowledge based on data are essential. Data mining techniques have been developed to find a small set of precious nugget, hidden but potentially valuable information, from reams of data. Nowadays, data mining has become a significant field of research [10].

Mining association rules is particularly useful for discovering relationships among items from large databases. Agrawal *et al*. first defined the mining problem, and developed an Apriori algorithm to generate significant association rules for retail organizations to analyze bar code data [2, 3]. The discovery of frequent itemsets notably controls the overall performance of the mining association rule system. Up to the present time, various algorithms have been proposed to rapidly discover the frequent itemsets, including Apriori and Apriori-like algorithms [2, 3, 7, 8] that follow, and pattern-growth methods [1, 11] as well.

Given a database of sales transactions, data analysis aims to discover all associations among items, such as customers' buying patterns. Shop managers can then arrange the store layout and promote their goods according to the buying patterns. Each product is called an *item*. A group of items bought together in a transaction is called an *itemset*. The *support* value of an itemset measures the itemset's importance in a transaction database. An itemset is said to be frequent when the occurrence count of the itemsets in the database is above a minimum support requirement [2]. However, the support measure does not consider the quantities of items bought in a transaction; that is, each item is a binary variable denoting whether an item was purchased. Therefore, the support count method does not measure the profit or the cost of an itemset. In 1997, Carter *et al*. presented a share-confidence framework to provide valuable information about numerical values associated with transaction items and considered the difficulty of mining characterized association rules from itemsets [9]. Recently, several share measure researches have been proposed to efficiently extract share-frequent itemsets with infrequent subsets [4, 5, 6, 12, 13, 14].

A share-frequent (SH-frequent) itemset usually includes some infrequent subsets. Consequently, the downward closure property cannot be applied to the discovery of all share-frequent itemsets. Several algorithms have been developed to discover share-frequent itemsets. However, some fast algorithms such as SIP, CAC and IAB [4, 5, 6] do not discover complete share-frequent itemsets. Obviously, an exhaustive search can generate all SH-frequents, but it is way too time-consuming and is therefore not applicable in a large dataset environment. Recently, Li *et al*. developed a Fast Share Measure (FSM) algorithm to swiftly discover all SH-frequent itemsets [14]. Although fast on a small dataset, FSM still generates too many useless candidates and is inefficient on a large dataset. To make a difference, this work proposes an Enhanced FSM (EFSM) to improve the performance of FSM. Moreover, to efficiently lower the number of useless candidates and further accelerate the mining process, especially in a large dataset, this study has also developed two new algorithms respectively called Support-counted FSM (SuFSM) and Share-counted FSM (ShFSM). For simplicity and without loss of generality, this study assumes that the measure value of each item is a non-negative integer.

The rest of this paper is organized as follows. Section 2 reviews the support-confidence and the share-confidence frameworks. Section 3 explains the proposed Enhanced Fast Share Measure (EFSM) algorithm, followed by the introduction of the proposed Support-counted FSM (SuFSM) algorithm in Section 4 and the proposed Share-counted FSM (ShFSM) algorithm in Section 5. Then, Section 6 provides experimental results and evaluates the performance of the proposed algorithms. Finally, we summarize our work with Section 7.

2 REVIEW OF SUPPORT AND SHARE MEASURES

2.1 Support-confidence framework

Agrawal *et al*. first introduced a model to define the problem of mining association rules [2, 3]. Given a transaction database, the mining of association rules is to discover the interesting rules, in which some items often appear at the same transactions. Let $I=\{i_1, i_2, \ldots, i_n\}$ be a set of literals, called items. A set of items X is called an itemset, where $X \subseteq I$. Let $DB=\{T_1, T_2, ..., T_z\}$ be the transaction database, where each transaction T_q satisfies $T_q \in DB$, $T_q \subseteq I$, $1 \le q \le z$. Each transaction is assigned a unique identifier *TID* so that it can easily be distinguish from others. An itemset X is contained in T_q if and only if $X \subseteq T_q$. The form $X \Rightarrow Y$ presents an association rule, where $X \subseteq I$, $Y \subseteq I$ and $X \bigcap Y = \phi$ (For example, $I = \{A, B, C, D, E\}$, $X = \{A, C\}$, $Y = \{B, E\}$). The rule $X \Rightarrow Y$ involves two characteristic values *support* and *confidence*. If the itemset $X \bigcup Y$ appears in s% of transactions in *DB*, the support of the rule $X \Rightarrow Y$ is s%, denoted as $Sup(X \bigcup Y)$. If the set of transactions contains X in *DB* and it also has c% transactions containing Y, the confidence of the rule $X \Rightarrow Y$ is c%, denoted as $Conf(X \Rightarrow Y)$. Thus, $Conf(X \Rightarrow Y)=Sup(X \bigcup Y)/Sup(X)$. The idea of mining association rules is to generate all rules whose support and confidence exceed the pre-defined minimum support and minimum confidence thresholds, respectively. An itemset, whose support is not less than the minimum support (*minSup*) threshold, is called a frequent itemset.

Apriori prunes some infrequent itemsets, subsequently violating the downward closure feature to efficiently discover the frequent itemsets. The downward closure property is characterized as follows. An arbitrary subset of a frequent itemset is also a frequent itemset; otherwise the itemset is infrequent.

2.2 Share-confidence framework

In 1997, Hilderman *et al*. first introduced the share-confidence framework, which is an alternative method to address the importance of itemsets [9]. Instead of a binary attribute, each item i_p involves a numerical value of measure attribute in each transaction T_q. The term $mv(i_p, T_q)$ represents the measure attribute value of i_p in a transaction T_q, called the *measure value*. For example, in Table 1, $mv(D, T01) = 1$ and $mv(C, T03) = 3$. The numerical value can be an integer as in sales statistics, or a real number as in profit or revenue accounts. The total measure value of a transaction T_p is called the *transaction measure value*, denoted as $tmv(T_p)$, where $tmv(T_p) = \sum_{i_p \in T_q} mv(i_p, T_q)$. Let *Tmv* be the total of measure values in *DB*, where $Tmv = \sum_{T_q \in DB} \sum_{i_p \in T_q} mv(i_p, T_q)$. Similarly, $Tmv(db)$ denotes the total of measure values in *db*, where $db \subseteq DB$ and $Tmv(db) = \sum_{T_q \in db} \sum_{i_p \in T_q} mv(i_p, T_q)$. The other notations and their definitions are as follows [6, 14].

Definition 2.1 Each k-itemset $X \subseteq I$ has an associated set of transactions $db_X = \{T_q \in DB \mid X \subseteq T_q\}$. Therefore, db_X is the set of transactions that contain itemset X.

Definition 2.2 The measure value of an itemset X in a transaction T_q is called the *itemset measure value*, denoted as $imv(X, T_q)$, where $imv(X, T_q) = \sum_{X \subseteq T_q, i_p \in X} mv(i_p, T_q)$.

Definition 2.3 The *local measure value* of an itemset X, denoted as $lmv(X)$, is the sum of the itemset measure values in db_x. In other words, $lmv(X) = \sum_{T_q \in db_x} imv(X, T_q)$ $= \sum_{T_q \in db_X} \sum_{X \subseteq T_q, i_p \in X} mv(i_p, T_q)$.

Definition 2.4 The *itemset share value* of an itemset X, denoted as $SH(X)$, is the ratio of the local measure value of X to the total measure value. That is, $SH(X) = \frac{lmv(X)}{Tmv}$.

Definition 2.5 A k-itemset X is *share-frequent* (SH-frequent) if $SH(X)$ is exceeds a pre-defined minimum share threshold (*minShare*) s%.

Example 2.1 Consider the transaction database with eight transactions as shown in Table 1 and *minShare* = 35%. The column "Count" lists the corresponding measure value of each item in a transaction. Table 2 lists the local measure value and the share value of each 1-itemset, where $Tmv = 57$. Let $X = \{B, C, D\}$, and the local measure value of $\{B, C, D\}$ is $lmv(X) = imv(X, T01) + imv(X, T03) + imv(X, T06) + imv(X, T07) = 3 + 10 + 6 + 8 = 27$. $SH(X) = lmv(X)/Tmv = 27/57 = 0.474 > 35\%$. Therefore, $\{B, C, D\}$ is an SH-frequent itemset. Table 3 enumerates all SH-frequent itemsets.

Table 1: Example of a transaction database with counting

TID	Transaction	Count
T01	{A, B, C, D, E, G, H}	{1, 1, 1, 1, 1, 1, 1}
T02	{F, H}	{4, 3}
T03	{B, C, D}	{4, 3, 3}
T04	{C, E}	{4, 1}
T05	{B, D}	{3, 2}
T06	{B, C, D}	{3, 2, 1}
T07	{B, C, D, E}	{3, 4, 1, 2}
T08	{A, F, G}	{4, 2, 1}

Table 2: Local measure value and itemset share value of each 1-itemset

Item	A	B	C	D	E	F	G	H
$lmv(i_p)$	5	14	14	8	4	6	2	4
$SH(i_p)$	8.8%	24.6%	24.6%	14%	7%	10.5%	3.5%	7%

Table 3: All SH-frequent itemsets of the sample database

SH-frequent itemset	BC	BD	BCD
$lmv(X)$	21	22	27
$SH(X)$	36.8%	38.6%	47.4%

2.3 Fast share measure (FSM) algorithm

For the support measure, Apriori-like algorithms employ the downward closure property to reduce the number of candidates and speed up the mining process efficiently. However, some, or even all, of the subsets in an SH-frequent itemset could be infrequent. In other words, the characteristic of downward closure does not work on share measure. Obviously, the exhaustive search method can find all SH-frequent itemsets, but the running time is exponential. The ZSP algorithm, a variant of exhaustive search, only prunes the candidate itemsets whose local measure values are exactly zero [6]. Therefore, in our previous work, we first proposed a non-exhaustive search method called Fast Share Measure (FSM) to discover all SH-frequent itemsets efficiently [14]. Instead of the downward closure property, FSM takes advantage of the level closure property to rapidly decrease the number of candidates. Suppose we have $min\_lmv = minShare \times Tmv$. The level closure goes as follows. Given a *minShare* and a candidate k-itemset X, if $lmv(X) + (lmv(X)/k) \times MV \times Level < min\_lmv$, no superset of X with length $\leq k + Level$ is SH-frequent, where MV is the maximum measure value of all. The left hand side of the inequality is called the critical function $CF(X)$. Set $Level = ML - k$, the level closure property's inequality guarantees that no superset of SH-infrequent itemset X is SH-frequent if $CF(X) < min\_lmv$, where ML is the maximum length among all transactions.

FSM is also a level-wise, multiple passes algorithm. In each pass, FSM scans the entire database to count the local measure value of each itemset. Each candidate itemset X is pruned when $CF(X) < min\_lmv$. After the (k-1)-th pass, some candidates of C_{k-1} are pruned, and the remained candidate set is called RC_{k-1}. In the k-th pass, FSM joins two arbitrary RC_{k-1} candidates, whose first k-2 items are identical. The k subsets with the length of (k - 1) of each candidate k-itemset are all in RC_{k-1}; otherwise, the k-itemset has no SH-frequent superset and can be pruned. After C_k is produced, RC_{k-1} is deleted. Next, FSM scans the database to detect the SH-frequent itemsets. If no candidates can be generated, then the process is terminated. The performance of FSM overcomes ZSP two or three orders of magnitude [14].

3 ENHANCED FAST SHARE MEASURE (EFSM) ALGORITHM

In ZSP or FSM, the generation of C_k requires determining whether two arbitrary k-itemsets in RC_{k-1} can be joined. The first k-2 items of two arbitrary (k-1)-itemsets in RC_{k-1} must be compared. Therefore, the time complexity of C_k generation is $O(n^{2k-2})$, where n is the number of distinct items. The time complexity is the quadratic of the number of RC_{k-1} elements. However, the number of generated candidates is usually less than the number of combinations of arbitrary pairs of itemsets by several orders of magnitude. The process wastes too much computation time to execute the joining procedure. To reduce the time complexity of candidate generation, this study proposes the Enhanced FSM (EFSM) algorithm to discover SH-frequent itemsets rapidly.

When discovering long SH-frequent itemsets, the number of candidate itemsets C_k increases quickly even when the number of distinct items is small. The Apriori-gen function [2] is time-consuming when there are many RC_{k-1} candidates. Therefore, instead of joining arbitrary two itemsets in RC_{k-1}, EFSM joins arbitrary itemset of RC_{k-1} with a single item in RC_1 to generate C_k efficiently. Let candidate (k-1)-itemset X_p be $\{i_1, i_2\ldots, i_{k-1}\}$ in the order of literals. Let the candidate 1-itemset X_q be $i_q \in I$. If $i_{k-1} < i_q$, then $X = \{i_1, i_2\ldots, i_{k-1}, i_q\}$ is a candidate k-itemset. Next, in the prune step, the procedure deletes all itemsets $X \in C_k$ in which at least one (k-1)-subset of X is not in RC_k. The pseudo-codes of the Join and Prune functions are as follows.

```
function Join($RC_{k-1}$, $RC_1$)
1.    $C_k := \phi$;
2.    foreach $X_p = \{i_1, i_2, \ldots, i_{k-1}\} \in RC_{k-1}$ {
3.        foreach $i_q \in RC_1$ {
4.            if ($i_q > i_{k-1}$) {
5.                $X = X_p \cup i_q$;
6.                $C_k = C_k + X$;  }    }    }
```

```
function Prune($C_k$)
1.    foreach $X \in C_k$ {
2.        if there is a ($k$-1)-subset of $X \notin RC_{k-1}$ {
3.            $C_k = C_k - X$;  }    }
```

EFSM joins an arbitrary 1-itemset of RC_1 with each (k-1)-itemset of RC_{k-1} when $i_{k-1} < i_q$. Therefore, the time complexity of the join step is reduced to $O(n^k)$. Furthermore, EFSM and FSM generate the identical set of candidate k-itemsets. EFSM does not require other extra computation overhead to reduce the time complexity of the join step. The strategy also can be applied to the ZSP algorithm to improve its performance. The improved ZSP algorithm is called the Enhanced ZSP (EZSP) algorithm.

4 SUPPORT-COUNTED FAST SHARE MEASURE (SuFSM) ALGORITHM

Although EFSM reduces the time complexity of the join step, it does not reduce the number of candidates, which always limits the performance of discovering share-frequent itemsets. FSM employs the level closure property and has an inequality to limit candidate generation. To further reduce the number of RC_k candidates and improve the performance, this study also proposes the Support-counted FSM (SuFSM) and Share-counted FSM (ShFSM) algorithms (ShFSM is presented in Section 5). The performance of EFSM is significantly better than that of FSM, which has been demonostrates by our experiments (see Section 6 later). Therefore, the development of SuFSM and ShFSM is based on the EFSM algorithm.

Definition 4.1 Given a k-itemset X and a database DB, an arbitrary superset of X with length $k + 1$ in T_q is denoted as X^{k+1}, where $X^{k+1} \subseteq T_q \in DB$. For example, in Table 1, Let X

= $\{F\}$, X^{k+1} = $\{AF\}$, $\{FG\}$, or $\{FH\}$.

Definition 4.2 Given a k-itemset X and a database DB, the set which contains all $(k+1)$-supersets of X in DB is denoted as $S(X^{k+1})$, where $X^{k+1} \in S(X^{k+1})$. For example Table 1, Let $X = \{F\}$, $S(X^{k+1}) = \{\{A, F\}, \{F, G\}, \{F, H\}\}$.

Definition 4.3 Given a DB and a k-itemset X, the set of transactions of which each transaction contains at least one X^{k+1} is denoted as $db_{S(X^{k+1})}$. Therefore, the support value of $S(X^{k+1})$ denoted as $Sup(S(X^{k+1}))$ is equal to $|db_{S(X^{k+1})}|$. For example, in Table 1, Let $X = \{F\}$, $Sup(S(X^{k+1})) = 2$. Let $X = \{FH\}$, $Sup(S(X^{k+1})) = 0$.

Let $maxSup(X^{k+1})$ denote the maximum spport value among all $Sup(X^{k+1})$ in DB. Let X be a candidate k-itemset. FSM and EFSM obtain the upper bound of the maximum distribution number, $lmv(X)/k$, of transactions, containing the arbitrary superset of X. According to the bound value, the MV value and the ML value, FSM can determine whether X has any superset that could be an SH-frequent itemset. However, the value $lmv(X)/k$ is not tight to the $maxSup(X^{k+1})$. The following equation holds.

$$lmv(X)/k \geq Sup(X) \geq Sup(S(X^{k+1})) \geq maxSup(X^{k+1}). \qquad \textbf{(1)}$$

If no superset of X is an SH-frequent itemset, then the following four equations hold.

$$lmv(X) + (lmv(X)/k) \times MV \times (ML - k) < min\_lmv, \qquad \textbf{(2)}$$

$$lmv(X)+Sup(X) \times MV \times (ML - k) < min\_lmv, \qquad \textbf{(3)}$$

$$lmv(X)+Sup(S(X^{k+1})) \times MV \times (ML - k)<min\_lmv, \qquad \textbf{(4)}$$

and

$$lmv(X)+maxSup(X^{k+1}) \times MV \times (ML - k) < min\_lmv. \qquad \textbf{(5)}$$

The left hand side of each of the four equations is called a critical function $CF(X)$ of the itemset X. Equation 5 can be used to prune the most useless candidates among the four equations, but wastes much memory space to count the support value so as to obtain the value of $maxSup(X^{k+1})$. The number of combinations of all $(k+1)$-supersets reaches $O(n^{2k-2})$, where n is the number of distinct items. By contrast to obtaining the value of $Sup(S(X^{k+1}))$, the cost of finding the value of $maxSup(X^{k+1})$ is too high. Calculating the value of $Sup(S(X^{k+1}))$ only requires a variable to accumulate the support value for each candidate itemset when DB is scanned. Therefore, Support-counted FSM (SuFSM) modifies the inequality Eq. 2 used in FSM to Eq. 4. SuFSM employs the support value as a parameter to prune more candidates than FSM and EFSM do.

5 SHARE-COUNTED FAST SHARE MEASURE (ShFSM) ALGORITHM

SuFSM calculates the value of $Sup(S(X^{k+1}))$ for each candidate in C_k being applied to determine RC_k. For EFSM and SuFSM, a lower critical function value generates a smaller set RC_k. Although the two parameters MV and $(ML - k)$ limit the upper bound of the $CF(X)$ value, they relax the real upper bound. Therefore, this section presents the Share-counted FSM (ShFSM) algorithm to compact the interval between the critical function value and the maximum local measure value of X^{k+1}. Let X be a k-itemset, which is a subset of $(k+1)$-itemset X^{k+1} in DB. Other notations and characteristics of ShFSM are as follows.

Definition 5.1 Given a DB, let X and X' be two itemsets, where $X \subseteq X'$. The local measure value of X on X', denoted as $lmv(X, X')$, is the sum of the itemset measure values in $db_{X'}$. In other words, $lmv(X, X') = \sum_{T_q \in db_{X'}} imv(X, T_q)$. If $X = X'$, then $lmv(X, X') = lmv(X)$.

Lemma 5.1 Let X, X' and X'' be three itemsets, $X \subseteq X' \subseteq X''$, then

(1) $lmv(X, X'') \leq lmv(X', X'')$, especially when $X' = X''$, $lmv(X, X') \leq lmv(X')$.
(2) $lmv(X, X') \geq lmv(X, X'')$, especially when $X = X'$, $lmv(X) \geq lmv(X, X'')$.

Proof. The proof of this lemma may be found in [14].

Theorem 5.1 Given a DB and a k-itemset X, we have $lmv(X) + Sup(S(X^{k+1})) \times MV \times (ML - k) \geq Tmv(db_{S(X^{k+1})})$.
Proof. Let $(k+1)$-superset of X be $X^{k+1} \subseteq T_p$. By Lemma 5.1, $lmv(X) \geq lmv(X, X^{k+1})$. So,

$$\begin{aligned} &lmv(X) + Sup(S(X^{k+1})) \times MV \times (ML\text{-}k) \\ &\quad \geq lmv(X, X^{k+1}) + Sup(S(X^{k+1})) \times MV \times (ML\text{-}k). \end{aligned} \qquad \textbf{(6)}$$

For each transaction $T_q \in DB$, since $MV \times (ML - k) \geq tmv(T_q) - imv(X, T_q)$, $Sup(S(X^{k+1})) \times MV \times (ML - k) \geq \sum_{T_q \in db_{S(X^{k+1})}} tmv(T_q) - \sum_{T_q \in db_{S(X^{k+1})}} imv(X, T_q)$ in $db_{S(X^{k+1})}$. So,

$$\begin{aligned} &Sup(S(X^{k+1})) \times MV \times (ML - k) \\ &\qquad \geq Tmv(db_{S(X^{k+1})})\text{-}lmv(X, X^{k+1}). \end{aligned} \qquad \textbf{(7)}$$

By Eqs. 6 and 7, we have

$lmv(X)+Sup(S(X^{k+1})) \times MV \times (ML\text{-}k) \geq Tmv(db_{S(X^{k+1})})$.
Q.E.D

Theorem 5.2 Given a $minShare$, if $Tmv(db_{S(X^{k+1})}) < min\_lmv$, all supersets of X are infrequent.
Proof. For arbitrary superset X' of X with length $k + i$, where $0 < i \leq (ML - k)$, since $db_{X'} \subseteq db_{S(X^{k+1})}$, $Tmv(db_{X'}) \leq Tmv(db_{S(X^{k+1})})$. In a DB, $lmv(X') \leq Tmv(db_{X'})$. So, if the inequality $Tmv(db_{S(X^{k+1})}) < min\_lmv$ holds, $lmv(X') < min\_lmv = minShare \times Tmv$. That is, $SH(X') = lmv(X') / Tmv < minShare$. X' is infrequent. Q.E.D

Theorem 5.2 guarantees that if $Tmv(db_{S(X^{k+1})}) < minShare \times Tmv$, no superset of X is SH-frequent in DB. The characteristic can be applied to prune candidates whose supersets are not SH-frequent. The ShFSM algorithm based on EFSM is a level-wise, multiple passes algorithm. Among the three algorithms, EFSM, SuFSM and ShFSM, ShFSM obtains the smallest critical function value for a candidate itemset X. Therefore, ShFSM can prune more candidates than SuFSM and EFSM.

6 EXPERIMENTAL RESULTS

The performance of the proposed algorithms was compared with that of ZSP and FSM using a 1.5 GHz Pentium IV PC with 1.5 GB of main memory, running Windows XP Professional. All algorithms were coded in Visual C++ 6.0, and applied to several synthetic datasets. Each algorithm employed the trie structure to count the local measure value. The complete SH-frequent itemsets were outputted to main memory to avoid disk writing.

This study uses the IBM synthetic data generator [15] to generate several synthetic datasets. The VC++ version of the data generator was obtained from [16]. To simulate the characteristic of the measure value in each item which appears in each transaction, the measure value was randomly generated between 1 and *m*, with 50% of measure value set as 1. Table 4 summarizes the parameters used in these experiments. The notation T*x*.I*y*.D*z*.N*n*.S*m* denotes a dataset with given parameters *x*, *y*, *z*, *n* and *m*.

Table 4. Parameters

x	Mean size of the transactions
y	Mean size of potentially frequent itemsets
z	Number of transactions in *DB*
n	Number of items
m	Maximum measure value

Figures 1, 2 and 3 plot the performance curves of running time with the six algorithms applied to T4.I2.D100k.N50.S10, T6.I4.D100k.N200.S10 and T10.I6.D100k.N500.S20, respectively. The main memory was not enough to run ZSP and EZSP using the last two datasets. As Fig. 3 shows, FSM and EFSM generated too many candidates to run in main memory when *minShare* $\leq$ 0.4%. The *x*-axis represents several distinct *minShare* thresholds between 0.1% and 1.2%, and the *y*-axis stands for the running time. The three figures use a logarithmic scale for the *y*-axis. Figure 2 demonstrates that EZSP performed better than ZSP by more than two orders of magnitude. EFSM always significantly outperforms FSM. The three figures show that ShFSM had the best performance, followed by SuFSM. For example, as indicated in Fig. 3, ShFSM's running time was only 13%, 2% and 0.03% of those of SuFSM, EFSM and FSM, respectively, when *minShare* = 0.6%.

Table 5 presents the numbers of C_k and RC_k in each pass using the dataset T6.I4.D100k.N200.S10 when *minShare* = 0.1%. ShFSM generated the fewest candidates among the four algorithms. ShFSM terminated the process at pass 10 and performed best in that aspect, while the others went up to pass 13. FSM and EFSM produced the identical C_k and RC_k sets. EFSM significantly reduced the time complexity of the join step. Therefore, EFSM is faster than FSM. All the four algorithms can discover all SH-frequent itemsets even when no (*k*-1)-subset is SH-frequent. For example, in the pass 5, the five SH-frequent 5-itemsets have no SH-frequent 4-subset.

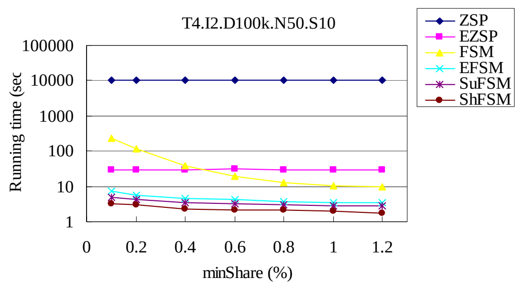

Figure 1: Running time comparison

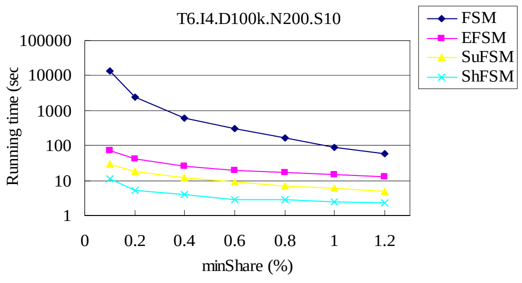

Figure 2: Running time comparison

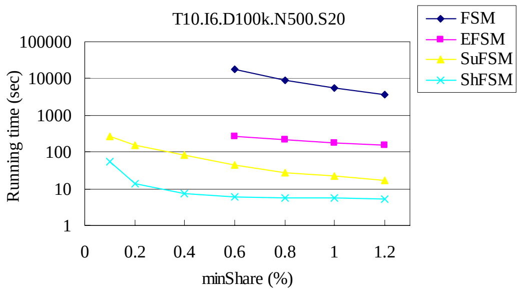

Figure 3: Running time comparison

Table 5: Comparison on the number of candidates in each pass using T6.I4.D100k.N200.S10 (*ML*=20)

Method / Pass (*k*)		FSM	EFSM	SuFSM	ShFSM	F_k
k=1	C_k	200	200	200	200	159
	RC_k	200	200	199	197	
k=2	C_k	19900	19900	19701	19306	1844
	RC_k	16214	16214	13312	7199	
k=3	C_k	829547	829547	564324	190607	101
	RC_k	251877	251877	99765	9792	
k=4	C_k	3290296	3290296	793042	20913	0
	RC_k	332877	332877	41057	1420	
k=5	C_k	393833	393833	25003	1050	5
	RC_k	71420	71420	19720	959	
k=6	C_k	26137	26137	11582	518	8
	RC_k	25562	25562	11045	506	
k=7	C_k	11141	11141	5940	204	7
	RC_k	11099	11099	5827	196	
k=8	C_k	4426	4426	2797	58	1
	RC_k	4423	4423	2750	54	
$k \geq 9$	C_k	2036	2036	1567	12	0
	RC_k	2030	2030	1513	10	
Time(sec)		13610.4	71.55	29.67	10.95	

Figure 4 presents the scalability with the transaction number of *DB* using T6.I4.Dz.N200.S10, where *minShare* = 0.3%. The *x*-axis represents several distinct *DB* sizes between 100k and 1000k, and the *y*-axis represents the running time. Figure 4 uses a logarithmic scale for the *y*-axis. The running times of EFSM, SuFSM and ShFSM linearly increased with the growth of the *DB* size. The running time of FSM only increased by 50% from |*DB*|=100k to |*DB*|=1000k, because the join step dominated the performance of FSM. The size of *DB* dominated the running times of EFSM, SuFSM and ShFSM.

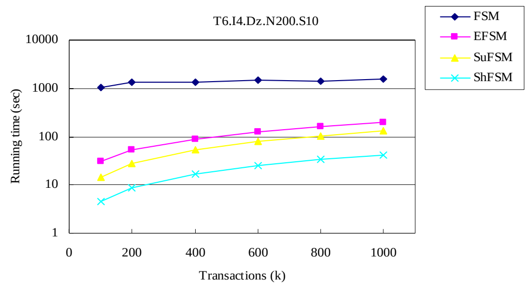

Figure 4: Scalability with the transaction number of *DB*,

7 CONCLUSIONS

The value of the itemset share can provide useful information, such as the total profit or total sales of an itemset in the database. Therefore, itemset share can overcome the drawbacks of the support measure. The development of an efficient way to discover complete SH-frequent itemsets is an important solution to various mining problems. However, the downward closure property fails to discover all share-frequent itemsets. To solve such problem and develop an efficient method for rapidly generating all SH-frequent itemsets, in this study, we have proposed the Enhanced FSM (EFSM) algorithm to efficiently reduce the time complexity of the join step. In addition, we have also developed SuFSM and ShFSM from EFSM. SuFSM and ShFSM can efficiently prune the candidates, and significantly improve the performance. The experimental results have indicated that ShFSM has the best performance. In the future, the authors plan to develop even more advanced algorithms to accelerate the process of identifying all SH-frequent itemsets.

REFERENCES

[1] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad (2001) “A tree projection algorithm for generation of frequent itemsets”, Journal of Parallel and Distributed Computing, 61:350-361.

[2] R. Agrawal, T. Imielinski, and A. Swami (1993) “Mining association rules between sets of items in large databases”, in Proc. 1993 ACM SIGMOD Intl. Conf. on Management of Data, Washington, D.C., 207-216.

[3] R. Agrawal and R. Srikant (1994) “Fast algorithms for mining association rules”, in Proc. 20th Intl. Conf. on Very Large Data Bases, Santiago, Chile, 487-499.

[4] B. Barber and H. J. Hamilton (2000) “Algorithms for mining share frequent itemsets containing infrequent subsets”, Lecture Notes in Computer Sciences, Springer-Verlag, Germany, 1910:316-324.

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

[6] B. Barber and H. J. Hamilton (2003) “Extracting share frequent itemsets with infrequent subsets”, Data Mining and Knowledge Discovery, 7:153-185.

[7] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur (1997) “Dynamic itemset counting and implication rules for market basket data”, in Proc. 1997 ACM SIGMOD Intl. Conf. on Management of Data, Tucson, AZ, 255-264.

[8] F. Berzal, J. C. Cubero, N. Marín, and J. M. Serrano (2001) “TBAR: An efficient method for association rule mining in relational databases”, Data & Knowledge Engineering, 37:47-64.

[9] C. L. Carter, H. J. Hamilton, and N. Cercone (1997) “Share based measures for itemsets”, Lecture Notes in Computer Science, Springer-Verlag, Germany, 1263:14-24.

[10] M. S. Chen, J. Han, and P. S. Yu (1996) “Data mining: An overview from a database perspective”, IEEE Trans. Knowledge Data Engineering, 8:866-883.

[11] J. Han, J. Pei, Y. Yin, and R. Mao (2004) “Mining frequent pattern without candidate generation: A frequent pattern tree approach”, Data Mining and Knowledge Discovery, 8:53-87.

[12] R. J. Hilderman (2003) “Predicting itemset sales profiles with share measures and repeat-buying theory”, Lecture Notes in Computer Science, Springer-Verlag, Germany, 2690:789-795.

[13] R. J. Hilderman, C. L. Carter, H. J. Hamilton, and N. Cercone (1998) “Mining association rules from market basket data using share measures and characterized itemsets”, Intl. Journal of Artificial Intelligence Tools, 7:189-220.

[14] Y. C. Li, J. S. Yeh, and C. C. Chang (2005) “A fast algorithm for mining share-frequent itemsets”, Lecture Notes in Computer Science, Springer-Verlag, Germany, 3399:417-428.

[15] http://alme1.almaden.ibm.com/software/quest/Resources/datasets/syndata.html

[16] http://www.cse.cuhk.edu.hk/~kdd/data/IBM_VC++.zip