International Journal of Innovative
Computing, Information and Control
Volume **9**, Number **1**, January **2013**

ICIC International ©2013 ISSN 1349-4198
pp. **207–222**

AN EFFICIENT MODEL FOR MINING PRECISE QUANTITATIVE ASSOCIATION RULES WITH MULTIPLE MINIMUM SUPPORTS

Shih-Sheng Chen[1] and Tony Cheng-Kui Huang[2,*]

[1]Department of Information Management
National Chin-Yi University of Technology
No. 57, Sec. 2, Zhongshan Rd., Taiping Dist., Taichung 41170, Taiwan
sschen@ncut.edu.tw

[2]Department of Business Administration
National Chung Cheng University
No. 168, Sec. 1, University Rd., Min-Hsiung Township, Chia-yi County 621, Taiwan
*Corresponding author: bmahck@ccu.edu.tw

Received June 2011; revised January 2012

Abstract. *Association rule mining deals with the correlations of items in a transaction. However, this approach raises two problems. First, each item is set with a unified minimum support which cannot be applied in actual applications. The reason is that the frequencies at which items are bought differently. In a shopping case, some items are bought frequently but others are seldom bought because of demand or price. Therefore, setting a unified higher threshold value makes it difficult to find rare items with higher prices, while a unified lower threshold value might lead to a combinational explosion problem. Second, traditional association rules lack quantity-related information and cannot reveal the quantities of different items in an association rule. Therefore, this study proposes quantity-related association rules adopting multiple minimum supports to address these two problems. An efficient algorithm is developed based on a divide-and-conquer idea to find precise quantitative association rules with multiple minimum supports in bag databases. Experiments show the algorithm's computational efficiency and scalability. Our research model can contribute to applications in more realistic circumstances in cases where items occur at various frequencies and have different quantity-related information.*

Keywords: Data mining, Association rules, FP-tree, Multiple minimum supports, Quantitative data

1. **Introduction.** Data are generated and collected so rapidly in databases that data mining techniques are popularly applied in commercial and scientific domains [1-5]. We can use different data mining techniques to extract potential and useful knowledge from volumes of data [6-13]. One important technique is mining association rules. The purpose of this technique is to find associations and correlations of items in transaction databases.

The topic of association rule mining was first discussed in a study by Agrawal et al. [14], which gave the following description. Given a transaction database, an association rule is an implication of the form $X \to Y$, where X and Y are subsets of items and $X \cap Y = \varnothing$. This rule indicates that if we purchase X then it is likely that we will purchase Y as well. To find the association rules from databases, the supports of itemset X and $X \cup Y$ need to be calculated preliminarily. The support of X, support(X), is the number of transactions in the database containing X and the support of $X \cup Y$, support($X \cup Y$), is the number of transactions in the database containing $X \cup Y$. Next, we determine if the itemsets $X \cup Y$ and X are frequent, where an itemset is frequent if its support is greater than or equal to the user-specified minimum support called min_sup. If they are frequent, the confidence

of the rule is calculated by formula support($X \cup Y$)/support(X) . Finally, the $X \rightarrow Y$ holds if its confidence is greater than or equal to the user-specified minimum confidence which is called min_conf.

In business, association rule mining is an approach to implementing bundling strategies [4]. When creating marketing strategies, merchants can employ association rules to bundle some products into a package. The idea of bundling is similar to creating a new product and has three advantages [16,17]. First, it is not necessary to redesign a new product, hence, avoiding the risks of new product development. Second, it helps create new markets. Finally, bundling different products for various packages can attract more different kinds of customers. Therefore, the use of mining association rules for understanding the relationships between items can help merchants develop more bundling strategies for marketing.

Although mining association rules can be applied to business, we still lack quantity-related information to deal with more precise bundling strategies. Unfortunately, this requirement is not satisfied in traditional models. Srikant and Agrawal [18], therefore, proposed a new way to mine association rules with quantitative attributes (e.g., purchased quantities of items). For example, we may have a traditional association rule like "milk→red wine"; however, a quantitative association rule may look like "milk = 4→bread = 2". The latter allows us to comprehend quantitative information to make more suitable strategies. Later, Hsu et al. [19] extended the idea to mine quantitative association rules in bag databases concerned with the number of units purchased in each transaction. They proposed simple rules to discover quantitative association rules in bag databases. In a simple rule, the comparison operators, "≥" or "=", must be of the same kind. The form of simple rules looks like "(milk = 8)→(red wine = 1)", which means that if customers buy 8 units of milk, then it is likely that they will buy 1 unit of red wine. To develop bundling strategies for precise quantitative information, simple rules are available for marketing. This means that we need to know the exact quantities of items which are bundled, rather than their rough quantities. We therefore remove the two operators, ≤ and ≥, as well as the semantic rules, and only maintain one operator, "=", to mine quantitative association rules. For the reason given above, we redefine our problem as a *precise quantitative association rules* mining problem.

To pinpoint the best bundling strategies, we still need to conquer a problem, i.e., there is only a single min_sup for all items. According to the problem of mining association rules, using a single min_sup implies that all items in the data are of the same nature or occur at similar frequencies in the database. However, some items are bought frequently while others are seldom bought because of demand or price. For example, milk is bought often but red wine is seldom bought. Therefore, setting a higher min_sup makes it difficult for merchants to find associations of rare items with higher prices, such as the associations among red wine and other products. However, setting a lower min_sup leads to a combinational explosion problem, such as miscellaneous or trivial associations among milk and other products. Therefore, we adopt the idea of multiple minimum supports introduced by Liu et al. [15] to solve this problem. Their method is to allow users to specify different minimum supports for each item according to the nature of the item. For example, the min_sup of milk could be set higher, yet that of red wine could be set lower. As we know, their proposed algorithm, the MSapriori algorithm, is developed by an Apriori-like algorithm and suffers from the problem of inferior performance problem. To overcome the problem, we modify an efficient method and the frequent pattern growth (FP-growth) algorithm, with the frequent pattern tree (FP-tree) [20]. We then propose a novel algorithm, called the QFP-growth algorithm, with a quantitative tree structure

and a QFP-tree, to discover precise quantitative association rules with multiple minimum supports (PQAR-MMS) in bag databases.

The proposed algorithm is applied to the following two conditions. First, the spectrum of prices for products has to be wider under the same circumstance; that is, the prices of some products are lower while those of some products are higher. Based on this condition, we can simply specify different minimum supports for different products (items). The cases of convenience, retail, or wholesale stores can satisfy this condition. Second, in the data-collection process, the quantities of products purchased from customers have to be stored in databases. Therefore, we can find the associations in products based on different natures and frequencies and the quantitative information.

The remainder of this paper is organized as follows. Section 2 reviews related works. Section 3 formally defines the problem and QFP-tree construction. Section 4 presents the QFP-growth algorithm. Section 5 then compares the performances of the algorithms. Finally, conclusions and suggestions for future work are given in Section 6.

2. **Related Works.**

Since the study of mining precise quantitative association rules with multiple minimum supports stems from previous research [15,19,20], we briefly review three algorithms in this section. In the beginning, we are only concerned with mining simple rules and discuss the development of mining quantitative association rules with one comparison operator (MQA-1) algorithm. This algorithm not only considers the items bought but also the quantities of items. However, it has two drawbacks which need to be solved; i.e., it only specifies one minimum support and it has inferior performance. Therefore, we review studies related to the MSapriori algorithm to address the first drawback and studies related to the FP-growth algorithm to address the second drawback.

2.1. **The MQA-1 algorithm.**

For mining simple rules, Hsu et al. [19] proposed the MQA-1 algorithm. It consists of the following two steps: (1) finding all frequent itembags; (2) using the frequent itembags to generate quantitative association rules.

The MQA-1 algorithm, however, has two drawbacks. First, it adopts an Apriori-like candidate set generation-and-test approach, and is costly and time-consuming, especially when there are long patterns in the database. Second, it uses only one single minimum support to assume that all items are of the same nature or occur at similar frequencies in the database [15]. Our study, therefore, addresses these two issues.

2.2. **The MSapriori algorithm.**

The MSapriori algorithm was proposed in Liu et al. [15]. This model extends the existing association rule model to allow users to specify multiple minimum supports to reflect the different natures and frequencies of items. Moreover, it enables users to find rare item rules without producing a huge number of meaningless rules. In this model, the definition of minimum support is changed. Each item in the database has its minimum support threshold, which is expressed in terms of *minimum item supports* (MIS). By providing different MIS values for different items, users can effectively express different support requirements for different rules.

Mining association rules typically consists of two steps: (1) finding all frequent itemsets and (2) generating association rules using the frequent itemsets. When there is only one minimum support, the above two steps satisfy the *downward closure property*. That is, if a set of items satisfies the minimum support, all its subsets also satisfy the minimum support.

Liu et al. [15] modified a well-known Apriori algorithm so that the MSapriori algorithm can be used to find all frequent itemsets with multiple minimum supports. The details can be referred to in Liu et al. [15].

2.3. **The FP-tree and the FP-growth algorithm.** Han et al. [20] proposed a frequent pattern tree structure, called FP-tree, which is an extended prefix-tree structure for sorting compressed and crucial information. Consequently, the FP-growth is an FP-tree-based mining algorithm for mining complete sets of frequent patterns. The frequent items only play a role before the construction of the FP-tree. All frequent items are sorted in descending order of their support counts. An FP-tree consists of one root labeled as "*null*", a set of item prefix subtrees as the children of the root, and a frequent-item header table.

An FP-tree is constructed as follows. Scan the database once to collect all frequent items and their support counts. All frequent items are sorted in descending order of support and denoted as L. After that, create the root of an FP-tree and label it as "null". Scan the database a second time. The items of each transaction in the database are sorted according to the order of L. On inserting a transaction, if the tree has the same path, then the count of each node in the path increases by 1. If the path is incomplete in the tree, then a new branch and new nodes are created. Moreover, the *node-links* of these new nodes are linked to the nodes with the same *item-name* via the node-link structure. After constructing the FP-tree, the FP-growth algorithm recursively builds a "conditional pattern base" and "conditional FP-tree". Then, we use them to generate all frequent patterns.

The following two ideas emerge from the above review. First, we can specify multiple minimum supports to remedy the first drawback of the MQA-1 algorithm. Second, we can design an FP-growth-like algorithm to conquer the second drawback. As such, we propose an FP-growth-based algorithm, called the QFP-growth algorithm, with multiple minimum supports to discover precise quantitative association rules.

3. **Problem Definition.** In this section, a new tree structure, named the QFP-tree, is proposed for mining precise quantitative association rules with multiple minimum supports (PQAR-MMS). It is designed by extending the FP-tree structure to find extra information.

Let $I = \{i_1, i_2, \ldots, i_m\}$ denote all items in bag database *DB* (call database afterward). Each transaction $T = \{b_1, b_2, \ldots, b_n\}$ is a bag of items (itembag), where $b_i \in I$ for all $i = 1, 2, \ldots, n$. An item b_i in T can be identical to an item b_j in T for $i \neq j$. Each transaction with a unique identifier is called *TID*. Let X be an itembag. For the sake of brevity, we denote X as $\{p_1q_1, p_2q_2, \ldots, p_kq_k\}$, where p_i (a.k.a $p_iq_i.item$) is a distinct item and q_i is the number of occurrences of p_i (a.k.a $p_iq_i.qty$) in X for all $i = 1, 2, \ldots, k$. T precisely contains X ($T \supseteq X$) if the following two conditions are satisfied. (1) Every item p_i in X also appears in T; (2) The number of occurrences of p_i in T is equal to q_i.

To mine PQAR with multiple minimum supports, we need to redefine the traditional support threshold approach. Here, according to our definition, each item in database *DB* has its minimum support, which is expressed in terms of minimum item support (MIS). That is to say, users can specify different MIS values for different items. Let MIS(a_i) denote the MIS value of item a_i. The percentage of transactions containing precisely X in *DB* is called the support of itembag X, which is denoted by support(X). If support(X) is no less than MIS(X), then itembag X is frequent. Otherwise, it is referred to as infrequent.

Example 3.1. *Consider the database DB in Table 1. To take an example of TID 100, there are four itembags and their MIS values are shown in Table 2. Let itembag* $X = \{A2, B1, C3, E2\}$ *and support*(X) *be equal to 1. Then, itembag X is infrequent because MIS*(X) $= \min[MIS(A2.item), MIS(B1.item), MIS(C3.item), MIS(E2.item)] = \min[5, 3, 4, 2] = 2$, *which is greater than support*(X).

TABLE 1. An example of database *DB*

TID	*Itembags Bought*
100	$A2$, $B1$, $C3$, $E2$
200	$A2$, $B2$, $C3$
300	$B2$, $D1$, $F1$
400	$A1$, $B1$, $D1$, $F2$
500	$A2$, $B2$, $C3$, $E1$
600	$A2$, $C3$, $E1$

TABLE 2. The MIS value of each item in *DB*

Item	A	B	C	D	E	F
MIS value	5	3	4	3	2	2

TABLE 3. The support count of each itembag in *DB*

Itembag	$A1$	$A2$	$B1$	$B2$	$C3$	$D1$	$E1$
Support	1	4	2	3	4	2	2
Itembag	$E2$	$F1$	$F2$				
Support	1	1	1				

Let *MIN* denote the smallest MIS value of all items ($MIN = \min[\text{MIS}(i_1), \text{MIS}(i_2), \ldots, \text{MIS}(i_m)]$), and let MIN_F denote the set of those itembags with supports no less than MIN.

Following Example 3.1, the supports of the four itembags are shown in Table 3.

Lemma 3.1. *Let L_k denote the set of all frequent itembags with k distinct items. Then each itembag in L_k ($k \geq 2$) must be in MIN_F.*

Rationale *Let $X = \{p_1q_1, p_2q_2, \ldots, p_kq_k\}$ be an itembag in L_k, then $MIS(X) = \min [MIS(p_1), MIS(p_2), \ldots, MIS(p_k)]$ which is no less than MIN. If any p_iq_i ($i = 1, 2, \ldots, k$) does not belong to MIN_F, then the support(X) must be less than MIS(X).*

In the traditional FP-tree, only the frequent items play a role in the candidate patterns of the next step. However, according to Lemma 3.1, the QFP-tree not only consists of all frequent itembags, but also those itembags with supports no less than *MIN* because their supersets might be frequent in the next phases. As follows, we define the QFP-tree.

Definition 3.1. (QFP-tree) *A QFP-tree is a tree structure defined as follows.*

1. *A QFP-tree consists of one root labeled as "null", a set of itembag prefix subtrees as the children of the root, and a frequent-itembag-header table which contains all itembags in MIN_F.*
2. *Each node in the itembag prefix subtree consists of three fields: itembag-name, count, and node-link, where the itembag-name registers which itembag the node represents, the count registers the number of transactions represented by the portion of the path reaching the node, and the node-link links to the next node in the QFP-tree carrying the same itembag-name, or null if there is none.*
3. *Each entry in the frequent-itembag-header table consists of three fields, (1) itembag-name, (2) MIS, and (3) head of node-link, which points to the first node in the QFP-tree carrying the itembag-name.*
4. *All the itembags in the table are sorted in nonincreasing order in terms of their MIS values.*

Algorithm 1 (The QFP-tree Construction Algorithm)
Input: a database *DB* and a minimum support threshold of each item, MIS.
Output: QFP-tree.

1 Scan the database *DB* once.
  1.1 Count the support value of each itembag in *DB*.
  1.2 Collect *MIN_F* the set of those itembags with supports no less than *MIN*.
2 For each transaction *Trans* in *DB* do the following:
  2.1 *f* denotes an itembag in *MIN_F*. For each *f* in *MIN_F* do the following:
    2.1.1 Delete the itembag in *Trans* with itembag $\notin$ *MIN_F*.
  2.2 Sort all of the itembags in *Trans* according to their MIS values in nonincreasing order.
3 Create the root of a tree, *T*, and label it as "null". For each transaction *Trans* in *DB* do the following:
  3.1 Let the sorted itembags in *Trans* be [*p*|*P*], where *p* is the first element and *P* is the remaining list.
  3.2 Call *insert_tree*([*p*|*P*], *T*)

Procedure insert_tree([*p*|*P*], *T*)
1 While (*P* is nonempty) {
2 If *T* has a child *N* such that *N*.itembag-name=*p*.itembag-name then
  N.count++;
3 Else
4 Create a new node *N*;
5 *N*.count=1;
6 Let its parent link be linked to *T*;
7 Let its node-link be linked to the nodes with the same itembag-name via the node-link structure;
8 }

FIGURE 1. The QFP-tree construction algorithm

Based on Definition 3.1, we have the following QFP-tree construction algorithm and each function used in Algorithm 1 is shown in Figure 1.

Figure 1 gives the QFP-tree construction algorithm. To construct the QFP-tree, our algorithm needs to scan the database. First, we count all supports of the itembags in *DB* by removing those supports of itembags less than MIN. Second, we scan all transactions in *DB*. For each transaction, we add each itembag into the tree. Next, we will show that the QFP-tree contains complete information for mining PQAR-MMS. Lemma 3.2 shows that the QFP-tree construction algorithm could contain complete information related to frequent patterns in *DB*.

Lemma 3.2. *Give a database DB and a support threshold MIS of each item. Then the QFP-tree contains complete information related to frequent patterns in DB.*

Rationale. *Based on the QFP-tree construction process, each transaction in the DB is mapped to one path in the QFP-tree, and all the information of the MIN_F itembags in each transaction is completely stored in the QFP-tree. According to Lemma 3.1, we also retain those itembags with supports no less than MIN in the QFP-tree.*

Example 3.2. *Let database DB be in Table 1. The MIS value of each item is shown in Table 2. According to Lemma 3.1, only those itembags with supports no less than MIN will play a role in the mining process. First, a scan of DB derives the support of each itembag, as shown in Table 3. Second, we remove those itembags with supports less than MIN for each transaction. According to Algorithm 1, itembags are arranged according to their MIS values in nonincreasing order. To simplify the discussion, the result of each transaction is listed in this order in the rightmost column of Table 4.*

Table 3 shows the actual occurrences of the itembags in the *DB*. We use itembag $A2$ as an example. We count itembag $A2$ if it appears in the second column of Table 1. After traversing Table 1, we know that itembag $A2$ appears in *TID*s 100, 200, 500 and 600. Then, we get support$(A2) = 4$. The remaining itembags in Table 3 can be handled in the same way. Table 3 shows the result. Table 2 shows the MIS values for each item, which are specified by users. However, in the following experimental study, we use another method proposed by Liu et al. [15] to assign MIS values to items. The formula can be defined as follows:

$$\text{MIS}(a_i) = \begin{cases} M(a_i) & M(a_i) \geq MIN \\ MIN & \text{Otherwise} \end{cases}, \quad M(a_i) = \sigma f(a_i),$$

$f(a_i)$ is the actual occurrence of item a_i and *MIN* is the smallest MIS value of all items. σ is a parameter that controls how the MIS values for items should be related to their frequencies. If $\sigma = 0$, we have only one minimum support, *MIN*. If $\sigma = 1$ and $f(a_i) \geq MIN$, $f(a_i)$ is the MIS value for item a_i. Consider item A shown in Table 2. Let $\sigma = 0.6$ and $MIN = 20\%$. The actual occurrence of item A is shown in Table 3. There are two itembags with the same item name but different quantities. We add their support counts to compute item A's MIS value. Then, $f(A) = 5/6 = 83\%$ and $M(A) = 0.6 \times 83\% = 49\%$. Because $M(A)$ is larger than *MIN*, we can get $\text{MIS}(A) = 49\%$. Otherwise, $\text{MIS}(A)$ is equal to *MIN*.

Finally, we describe the construction process in Table 4. To create the QFP-tree, we first create the root of the tree, labeled as "*null*". Scan the *DB* a second time. The scan of the first transaction leads to the construction of the first branch of the QFP-tree: $((A2 : 1), (C3 : 1), (B1 : 1))$. Notice that the itembags in the transaction are ordered according to their MIS values in nonincreasing order. The second transaction $(A2, C3, B2)$ shares a common prefix $(A2, C3)$ with the existing path $(A2, C3, B1)$. The count of each node along the prefix is incremented by 1, and one new node $(B2 : 1)$ is

TABLE 4. A database *DB*

TID	*Itembags Bought*	*Itembags Bought(Ordered)*
100	$A2$, $B1$, $C3$, $E2$	$A2$, $C3$, $B1$
200	$A2$, $B2$, $C3$	$A2$, $C3$, $B2$
300	$B2$, $D1$, $F1$	$B2$, $D1$
400	$A1$, $B1$, $D1$, $F2$	$B1$, $D1$
500	$A2$, $B2$, $C3$, $E1$	$A2$, $C3$, $B2$, $E1$
600	$A2$, $C3$, $E1$	$A2$, $C3$, $E1$

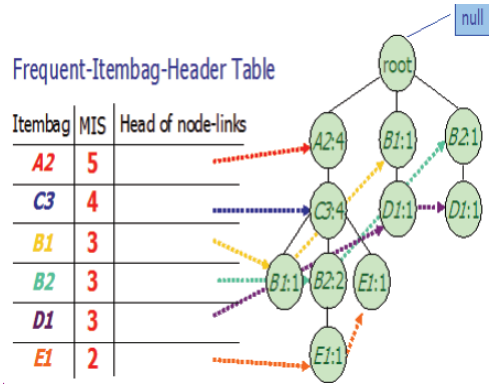

FIGURE 2. The QFP-tree

created and linked as a child of $(C3:2)$. The third transaction leads to the construction of the second branch of the QFP-tree. The remaining transactions in *DB* can be done in the same way.

To facilitate tree traversal, a frequent-itembag-header table is built. Each itembag points to its occurrence in the tree via a head of a node-link. Nodes with the same *itembag-name* are linked in sequence via such *node-links*. After the algorithm scans all the transactions, the tree with the associated *node-links* is shown in Figure 2.

Lemma 3.3. *(The space-complexity of Algorithm 1) Given a database DB and a minimum support threshold of each item, MIS, the number of nodes in a QFP-tree is no more than* $\sum_{t\in DB}|freq(t)|+1$*, where* $freq(t)$ *is the set of frequent itembags in transaction t. Moreover, the number of nodes in the longest path from the root is* $\max_{t\in DB}\{|freq\,(t)|\}$.

Rationale. *According to the QFP-tree construction process, for any transaction t in DB, let* $freq(t) = p_1q_1, p_2q_2, \ldots, p_nq_n$*. A path,* $root - p_1q_1 - p_2q_2 - \ldots - p_nq_n$*, exists in the QFP-tree. Except for when the root node is empty and is eliminated from the tree, all other nodes correspond to at least one frequent itembag occurring in the DB. In the worst case, there is no overlap among frequent itembag projections of transactions; and thus, all paths from the root to the leaves share only the root node. Therefore, the number of nodes in the tree is no more than* $\sum_{t\in DB}|freq(t)|+1$*. In the longest path from the root in the tree, there are* $\max_{t\in DB}\{|freq\,(t)|\}$ *nodes.*

4. Mining PQAR-MMS Using the QFP-Tree.

In this section, we will develop the QFP-growth algorithm for mining the complete set of PQAR-MMS. We also observe two properties of the QFP-tree structure in Section 4.1 and introduce the QFP-growth algorithm in Section 4.2.

4.1. The QFP-tree: properties.

We introduce the two properties of the QFP-tree structure as follows.

Property 1. *(Node-link property) For any frequent itembag* a_i*, all the possible frequent patterns of* a_i *can be obtained by following* a_i*'s node-links, initiating from* a_i*'s head in the QFP-tree header.*

This property is based on the QFP-tree construction process. It facilitates the process of finding all of a_i's patterns by traversing the QFP-tree and following a_i's node-links.

Property 2. *(Prefix path property) To compute* a_i*'s frequent patterns in a path P, only* a_i*'s prefix subpath in P needs to be accumulated and the frequency count of each node in the prefix path should hold the same count as node* a_i.

Rationale. *Let* $a_1, a_2, \ldots, a_n$ *be the nodes along the path P.* a_1 *is the root of the prefix subpath,* a_n *is the leaf of the subpath in P, and* a_i $(1 \le i \le n)$ *is the node being referenced. Based on the construction of the QFP-tree presented in Algorithm 1, for each prefix node* a_k $(1 \le k < i)$*, the prefix subpath of the node* a_i *in P occurs together with* a_k *exactly* $a_i.count$ *times. Thus, every such prefix node should hold the same count as node* a_i*. Notice that a postfix node* a_l $(i < l \le n)$ *along the same path also co-occurs with node* a_i*. However, all of* a_l*'s patterns will be generated upon the examination of postfix node* a_l*. Examining* a_l *here will lead to redundant generation. Therefore, we only need to examine the* a_i*'s prefix subpath.*

4.2. The QFP-growth algorithm.

In the QFP-growth algorithm, we recursively build "conditional QFP-trees" from the QFP-tree (denoted as "QFP-tree | itembag") and "conditional pattern bases" for mining frequent patterns. Below, we illustrate how to build the conditional QFP-trees and the conditional pattern bases.

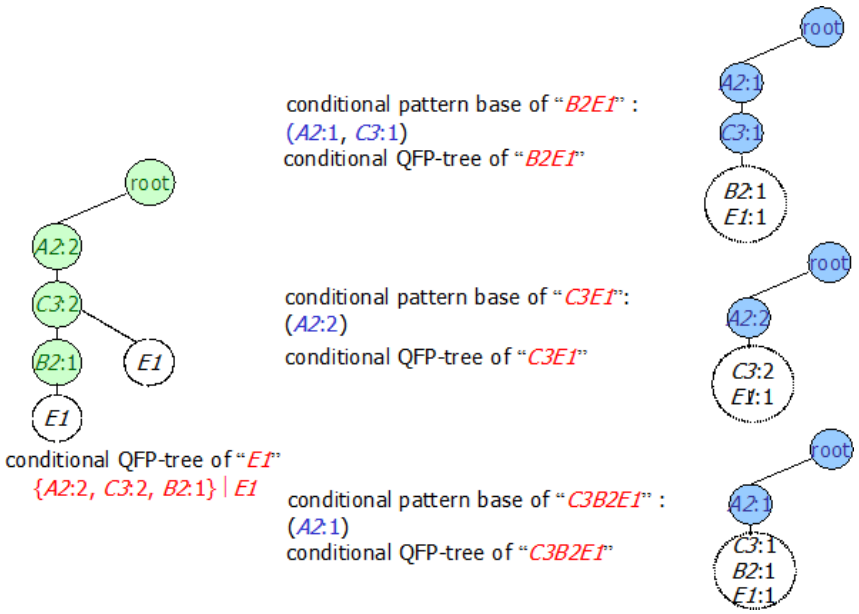

FIGURE 3. $E1$'s conditional QFP-tree

Example 4.1. *Let us examine the mining process using the QFP-tree shown in Figure 2. According to Property 1, we collect all patterns that node a_i participates in by starting from a_i's head (in the frequent-itembag-header table) and following a_i's node-links. Let us start from the bottom of the header table.*

For node $E1$, following the node-link, we get two paths in the QFP-tree: $(A2:4, C3:4, B2:2, E1:1)$ and $(A2:4, C3:4, E1:1)$. According to Property 2, only the prefix subpath of node $E1$ needs to be extracted. And the frequency count of each node should hold the same count as node $E1$. These two paths, $(A2:1, C3:1, B2:1)$ and $(A2:1, C3:1)$, with the exception of $E1$ from "$(A2:1, C3:1, B2:1, E1:1)$" and "$(A2:1, C3:1, E1:1)$", form the conditional pattern bases of $E1$. We use these conditional pattern bases to construct a conditional QFP-tree. Notice that the path $(A2, C3)$ appears twice together with $E1$. Similarly, the path $(B2)$ only appears once with $E1$. We derive the single frequent pattern path, $(A2:2, C3:2, B2:1)$, for the conditional QFP-tree of $E1$. The conditional QFP-tree for $E1$ is shown in Figure 3. After building the conditional QFP-tree, we get three itembags in the conditional QFP-tree of $E1$. To check whether the itembags are frequent or not, we follow the node-link of each itembag and sum up the counts. If the count of the itembag is greater than the MIS value of itembag $E1$, then it is frequent. Otherwise, it is infrequent. In the conditional QFP-tree for $E1$, the support count of $A2$ is 2, the support count of $C3$ is 2, and the support count of $B1$ is 1. Since the MIS value of $E1$ is 2, there are no frequent itembags in $E1$'s conditional QFP-tree. Therefore, we do not find any conditional frequent pattern for $E1$. After finding all the conditional patterns of $(E1)$, $(A2E1)$, $(C3E1)$, and $(B2E1)$, we recursively build the conditional pattern bases and the conditional QFP-trees of $A2E1$, $C3E1$, and $B2E1$. The conditional pattern base of $(A2E1)$ contains no itembags and the building process is thus terminated. For the conditional pattern base of $(B2E1)$, we find two patterns, $(A2B2E1:1)$ and $(C3B2E1:1)$. Therefore, the conditional patterns of $E1$ include $(A2E1)$, $(C3E1)$, $(B2E1)$, $(A2C3E1)$, $(A2B2E1)$, $(C3B2E1)$, and $(A2C3B2E1)$. The QFP-growth algorithm for itembag $E1$ is terminated until all the conditional pattern bases of $E1$ contain no itembags.

TABLE 5. Conditional pattern bases and conditional QFP-trees

Itembag	*MIS*	*Conditional pattern base*	*Conditional QFP-tree*
$E1$	2	$\{(A2:1, C3:1, B2:1), (A2:1, C3:1)\}$	$\{(A2:2, C3:2, B2:1)\}\|E1$
$D1$	3	$\{(B1:1), (B2:1)\}$	$\{(B1:1, B2:1)\}\|D1$
$B2$	3	$\{(A2:2, C3:2)\}$	$\{(A2:2, C3:2)\}\|B2$
$B1$	3	$\{(A2:1, C3:1)\}$	$\{(A2:1, C3:1)\}\|B1$
$C3$	4	$\{(A2:4)\}$	$\{(A2:4)\}\ \|C3$
$A2$	5	$\varnothing$	$\varnothing$

TABLE 6. Conditional patterns and conditional frequent patterns

Itembag	*Conditional patterns*	*Conditional frequent patterns*
$E1$	$A2E1$, $C3E1$, $B2E1$, $A2C3E1$, $A2B2E1$, $C3B2E1$, $A2C3B2E1$	$\varnothing$
$D1$	$B1D1$, $B2D1$	$\varnothing$
$B2$	$A2B2$, $C3B2$	$\varnothing$
$B1$	$A2B1$, $C3B1$	$\varnothing$
$C3$	$A2C3$	$A2C3$

All of the conditional pattern bases and conditional QFP-trees and conditional patterns and conditional frequent patterns are summarized in Table 5 and Table 6, respectively.

The FP-growth algorithm only finds frequent patterns in the conditional FP-tree. In our algorithm, we find all the conditional patterns. When we use multiple minimum supports, the *downward closure property* is no longer held. That is, if a set of itembags is frequent, then we are not sure whether all its subsets are also frequent or not. For this reason, the QFP-growth algorithm finds all conditional patterns.

The following lemma and corollary are related to the mining process.

Lemma 4.1. *(Fragment growth) Let α be an itembag in DB, B be α's conditional pattern base and β be an itembag in B. Then the support of $\alpha \cup \beta$ in DB is equivalent to the support of β in B.*

Rationale. *According to Property 2, each transaction in B which appears together with α should carry the same count. That is, if an itembag β appears at t times in B, it appears with α at t times in DB as well. Since all such itembags are collected in α's conditional pattern base, $\alpha \cup \beta$ appears at t times in DB as well.*

Corollary 4.1. *(Pattern growth) Let α be a frequent itembag in DB, B be α's conditional pattern base and β be an itembag in B. Then $\alpha \cup \beta$ is frequent in DB if and only if the support count of β is greater than MIS(α) in B.*

Rationale. *According to the QFP-tree construction process, the MIS value of each itembag in the conditional pattern base of α is no less than MIS(α). Then the MIS value of $\alpha \cup \beta$ is equal to MIS(α). In Lemma 4.1, the support of $\alpha \cup \beta$ in DB is equal to the support of β in B. Thus, if $\alpha \cup \beta$ is frequent with respect to MIS(α), β is frequent with respect to MIS(α).*

Based on the lemmas and properties, we have the following algorithm for mining the complete set of PQAR-MMS using the QFP-tree (see Figure 4).

5. **Experimental Evaluation and Discussion.** In this section, we perform a simulation study to compare the performances of the following algorithms: the MQA-1 [19],

FP-growth [20], and QFP-growth algorithms. The first algorithm finds quantitative association rules, i.e., simple rules, the second finds the traditional association rules without quantitative data, and the last one finds precise quantitative association rules. All the experiments are performed on a 3.2G Hz Pentium PC with 990 megabytes of main memory, running on Microsoft Windows 2000 Professional. All of the programs are implemented using Sun Java language.

The synthetic data generator is designed by modifying the well-known generator proposed in Agrawal and Srikant [21]. In our generation program, the parameter's definition and generation processes remain almost the same as those in Agrawal and Srikant [21]. In addition, we have a new parameter Q because our purpose is to mine precise quantitative association rules. Let Q be the average quantity of items to be purchased. The quantity of each item is chosen from a Poisson distribution with μ equal to Q. The parameters of the new synthetic data generator are shown in Table 7. For the experiments, we generate a number of datasets to test the algorithms. $N = 1000$ and $|L| = 2000$ are set and two parameters, $|T| = 7$ and $|I| = 4$, are fixed. We test three types of datasets for the number of transactions, $|D| = 50$K, $|D| = 100$K, and $|D| = 200$K.

In the experiments, Figures 5(a)-5(c) and Figures 6(a)-6(c) use the dataset N1000-T7-I4-D50K for three different MIN values, 0.3%, 0.5%, and 0.7%. Figures 5(a)-5(c) show the different scalabilities of the run times using the QFP-growth, FP-growth, and MQA-1 algorithms and Figures 6(a)-6(c) show the different frequent patterns generated from the QFP-growth, FP-growth, and MQA-1 algorithms. Figure 5(a) shows the run times for

Algorithm 2 (The QFP-growth Algorithm)
Input: QFP-tree, MIS(a_i) of each itembag a_i in *MIN_F*.
Output: The complete set of all a_i's conditional frequent patterns and the complete set of all support values of a_i's conditional patterns.
Method: call QFP-growth (QFP-tree, *null*, MIS(α)).

Procedure QFP-growth (*Tree*, α, MIS(α)) {
1. For each a_i in the header of *Tree* do {
2. Generate patter β=$a_i \cup \alpha$ with support=a_i.support;
3. Construct β's conditional pattern base and β's conditional QFP-tree *Tree* β;
4. If *Tree* $\beta \neq$null then call QFP-growth (*Tree* β, β, MIS(α)); }
}

FIGURE 4. The QFP-growth algorithm

TABLE 7. Parameters

$\|D\|$	*Number of transactions*
$\|T\|$	*Average size of the transactions*
$\|I\|$	*Average size of maximal potentially large patterns*
$\|L\|$	*Number of maximal potentially large patterns*
N	*Number of items*
Q	*Average quantity of items*

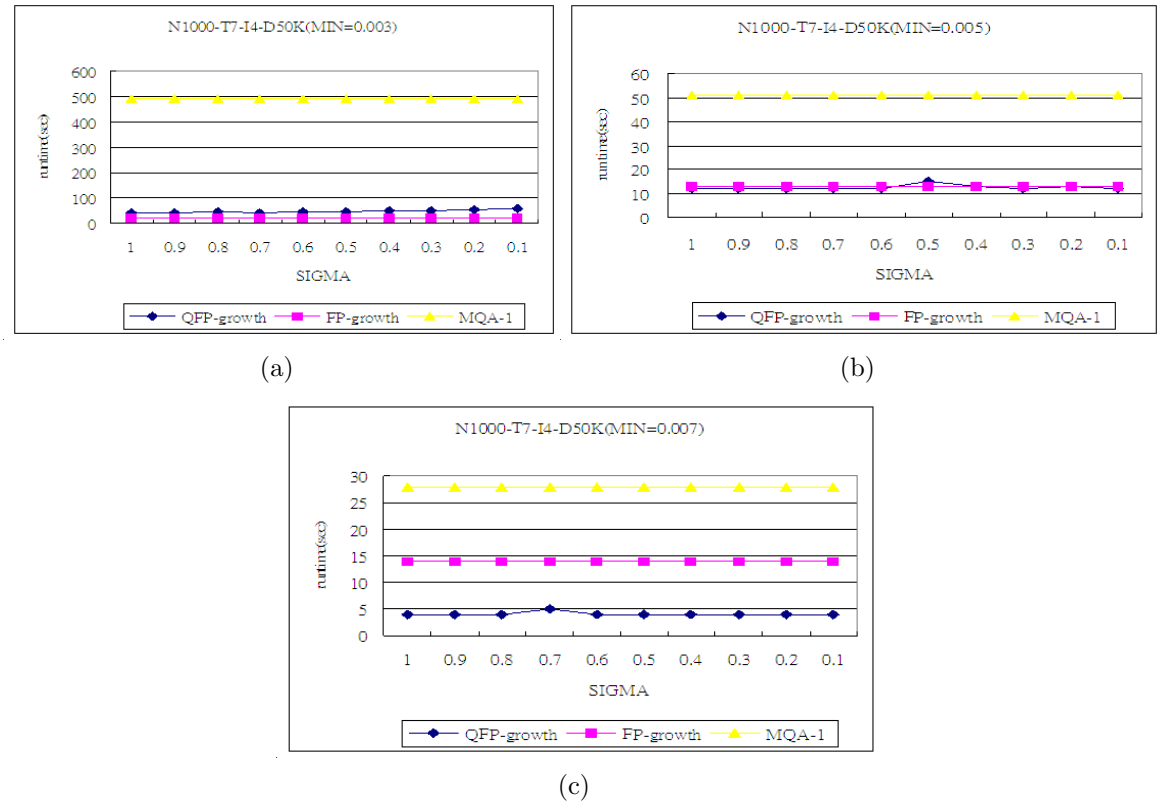

FIGURE 5. (a) Run times of N1000-T7-I4-D50K ($MIN = 0.003$), (b) run times of N1000-T7-I4-D50K ($MIN = 0.005$), (c) run times of N1000-T7-I4-D50K ($MIN = 0.007$)

$MIN = 0.3\%$, Figure 5(b) shows them for $MIN = 0.5\%$, and Figure 5(c) shows them for $MIN = 0.7\%$. All results indicate that the QFP-growth and FP-growth algorithms are faster than the MQA-1 algorithm. The results of Figures 6(a)-6(c) show that the QFP-growth algorithm can certainly reduce the number of possible redundant patterns which would be found by the MQA-1 algorithm. For example, in Figure 8(b), the QFP-growth algorithm generates about 100 frequent patterns, but the MQA-1 algorithm needs to generate more than 5000.

The datasets N1000-T7-I4-D50K, N1000-T7-I4-D100K, and N1000-T7-I4-D200K are used for these experiments to test the scalability with the numbers of transactions. Figure 7(a) shows the result for $MIN = 0.3\%$, Figure 7(b) shows the result for $MIN = 0.5\%$, and Figure 7(c) gives the result for $MIN = 0.7\%$. The presented runtime is derived from the average of ten tests for σ from 0.1 to 1.0. Figures 7(a)-7(c) depict the results. These three algorithms present almost linear scalability with the number of transactions from 50K to 200K. However, as the number of transactions is increased, the gap between the run times of the QFP-growth and the MQA-1 becomes larger. It is about 30 times faster for the QFP-growth algorithm than for the MQA-1 algorithm.

We also present a comparison of the number of frequent patterns based on the above scale-up experiments. The logarithmic scale is used for the Y-axis to show the number of frequent patterns. The numbers of frequent patterns shown in Figure 8(a) are generated in the experiments in Figure 7(a), the numbers of frequent patterns shown in Figure 8(b) are generated in experiments in Figure 7(b), and the numbers of frequent patterns shown in Figure 8(a) are generated in the experiments in Figure 7(c). Figures 8(a)-8(c)

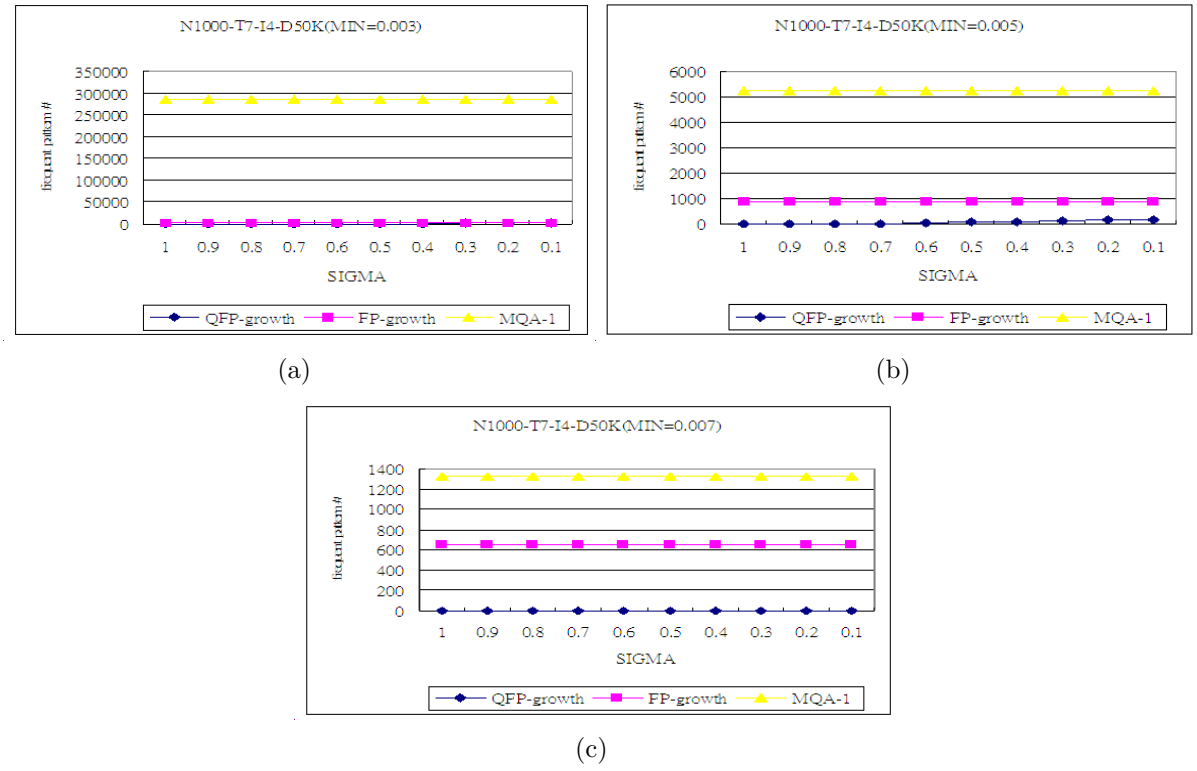

FIGURE 6. (a) Frequent patterns of N1000-T7-I4-D50K ($MIN = 0.003$), (b) frequent patterns of N1000-T7-I4-D50K ($MIN = 0.005$), (c) frequent patterns of N1000-T7-I4-D50K ($MIN = 0.007$)

indicate the difference in the number of the frequent patterns among the QFP-growth, FP-growth and MQA-1 algorithms. The QFP-growth algorithm generates less frequent patterns than do the FP-growth and MQA-1 algorithms because the QFP-growth adopts a different minimum support for each item. The MQA-1 algorithm finds much more frequent patterns because it counts the itembags, satisfying $\{ \geq, = \}$ for its quantity in the transaction.

According to the results of the above experiments, we compare the performances among the three algorithms in Table 8. The results indicate when it comes to run time and scalability, the QFP-growth and FP-growth algorithms performed better than the MQA-1 algorithm. However, when we look at the number of frequent patterns, only the QFP-growth algorithm can generate fewer patterns. In summary, only the QFP-growth algorithm exhibits superior efficiency and effectiveness because (1) it conveys the information about the quantities of items using different minimum supports and (2) provides good performance in terms of implementation. This can be practical and beneficial to merchants. The algorithm is executed to rapidly obtain the mined results so that merchants can immediately realize what quantities of different items should be bundled into different packages for selling.

6. **Conclusions and Future Work.** We propose a quantitative tree structure, the QFP-tree, which extends the traditional frequent pattern tree (FP-tree). The proposed algorithm stores all information related to frequent patterns and contains more information. We, therefore, develop an efficient mining algorithm to discover precise quantitative association rules with multiple minimum supports. We also study its performance in comparison with two other algorithms. In the scalability tests, the executive time of the QFP-growth

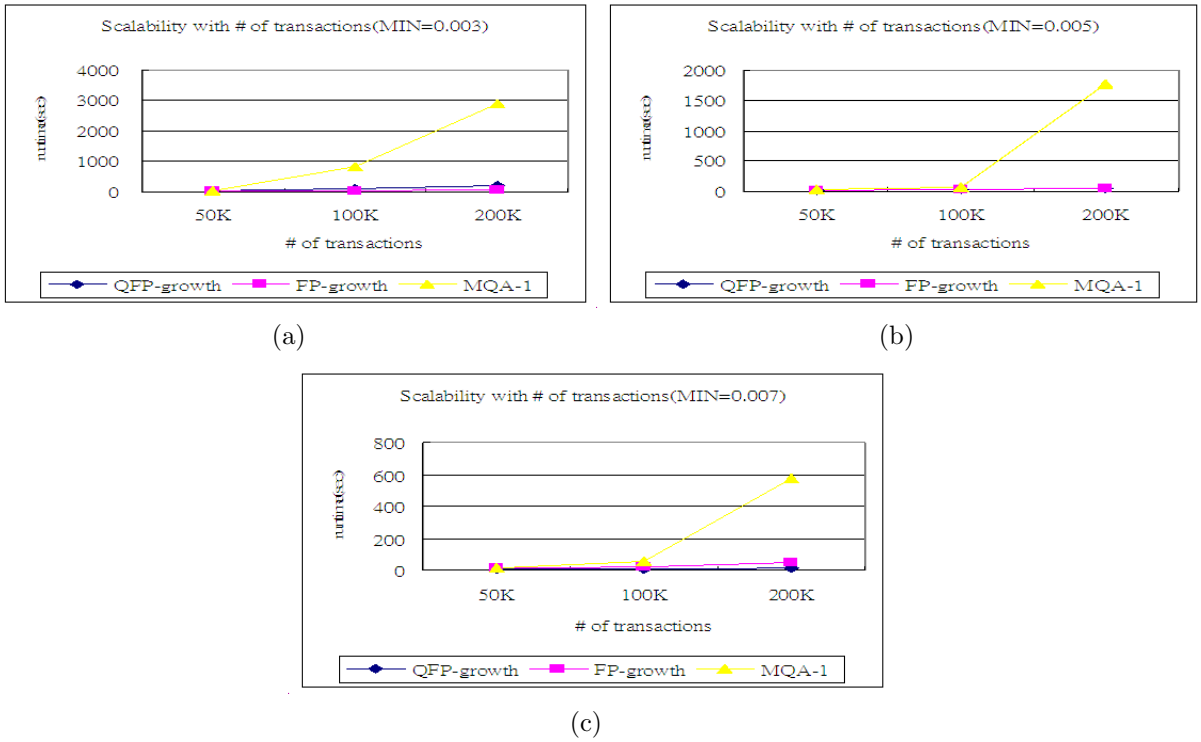

FIGURE 7. (a) Scalability with # of transactions ($MIN = 0.003$), (b) scalability with # of transactions ($MIN = 0.005$), (c) scalability with # of transactions ($MIN = 0.007$)

TABLE 8. The comparison results

Feature \ Algorithm	QFP-growth	FP-growth	MQA-1	The well-performed algorithm(s)
Run time	Good	Good	Bad	QFP-growth and FP-growth
Scalability	Linear	Linear	Exponential	QFP-growth and FP-growth
Frequent pattern #	Fewer	More	Most	QFP-growth

algorithm is better than that of the MQA-1 algorithm as the dataset size increases from 50K to 200K and the support threshold decreases from 0.7% to 0.3%. Another study of the numbers of frequent patterns generated by the QFP-growth and MAQ-1 algorithms presents a varying number of frequent patterns, i.e., the former generates fewer patterns than the latter does. This all shows that mining precise quantitative association rules with multiple minimum supports is more realistic in business applications and helps us to reduce more redundancy patterns, making bundling strategies quick and precise.

Studying precise quantitative association rules with multiple minimum supports represents a new and promising type of research in data mining. Our research has contributed a novel model and has shown its efficiency through experiments. In the future, practitioners can extend the model for application in different types of real life datasets to further validate its effectiveness.

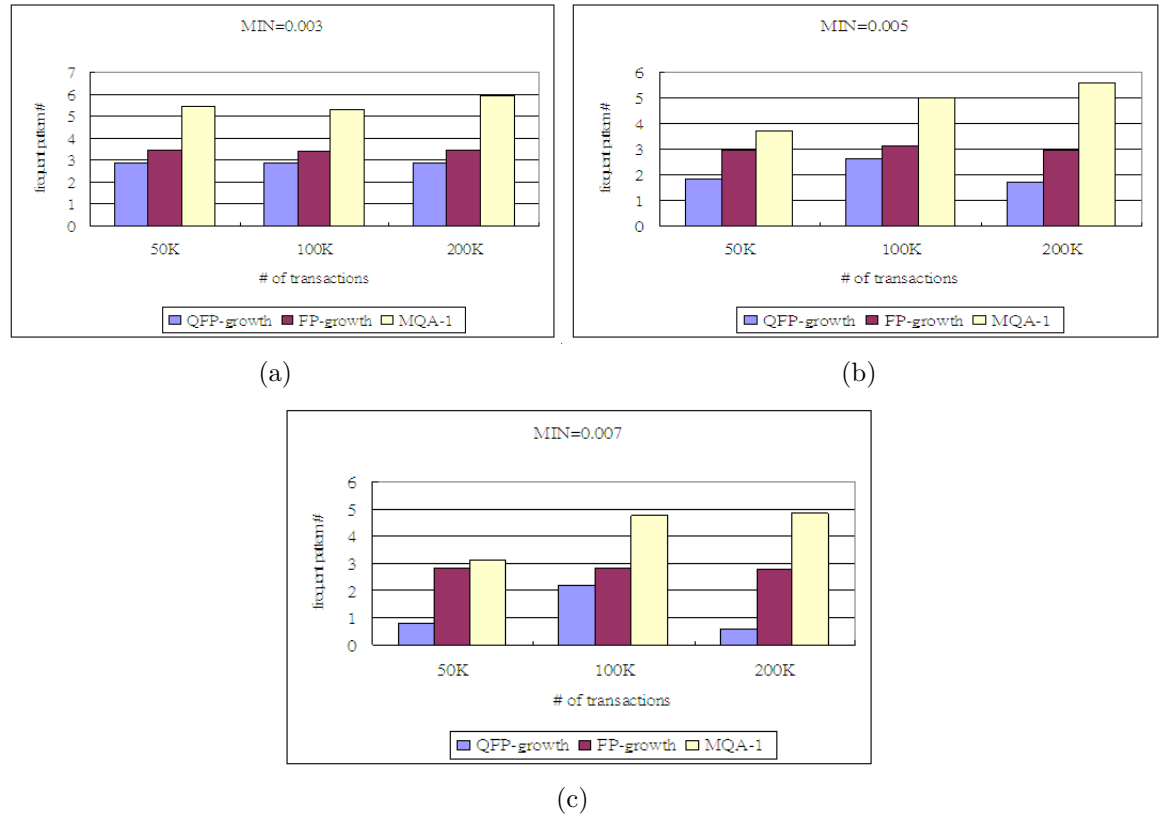

FIGURE 8. (a) Scalability with # of transactions ($MIN = 0.003$), (b) scalability with # of transactions ($MIN = 0.005$), (c) scalability with # of transactions ($MIN = 0.007$)

Acknowledgment. The authors would like to thank the Executive Editor, Dr. Yan Shi, and anonymous reviewers for their helps and valuable comments to improve this paper. They also thank Jia-Hui Zhuang for the implementations of all algorithms. This work is partially supported by the National Chin-Yi University of Technology under contract number NCUT 12-R-MM-003 and the National Science Council of Taiwan under contract numbers NSC-93-2416-H-167-011 and NSC 100-2410-H-194-019-MY2.

REFERENCES

[1] R. E. Valdes-Perez, Discovery tools for science apps, *Communications of the ACM*, vol.42, no.11, pp.37-41, 1999.
[2] J. McCarthy, Phenomenal data mining, *Communications of the ACM*, vol.43, no.8, pp.75-79, 2000.
[3] C. Apte, B. Liu, E. P. D. Pednault and P. Smyth, Business applications of data mining, *Communications of the ACM*, vol.45, no.8, pp.49-53, 2002.
[4] Y. Yin, An approach to mining bundled commodities, *Knowledge-Based Systems*, vol.21, no.4, pp.321-331, 2008.
[5] J. Zhu and Y. Dai, A novel procedure of customer pattern recognition with data mining, *ICIC Express Letters*, vol.5, no.6, pp.1953-1959, 2011.
[6] M. S. Chen, J. Han and P. S. Yu, Data mining: An overview from a database perspective, *IEEE Transactions on Knowledge and Data Engineering*, vol.8, no.6, pp.866-883, 1996.
[7] U. Fayyad, G. Piatetsky-Shapiro and P. Smyth, The KDD process for extracting useful knowledge from volumes of data, *Communication of the ACM*, vol.39, no.11, pp.27-34, 1996.
[8] H. Yun, D. Ha, B. Hwang and K. H. Ryu, Mining association rules on significant rare data using relative support, *Journal of Systems and Software*, vol.67, no.3, pp.181-191, 2003.

[9] A. J. T. Lee, W.-C. Lin and C. S. Wang, Mining association rules with multi-dimensional constraints, *Journal of Systems and Software*, vol.79, no.1, pp.79-92, 2006.
[10] C. Kim, J.-H. Lim, R. T. Ng and K. Shim, SQUIRE-Sequential pattern mining with quantities, *Journal of Systems and Software*, vol.80, no.10, pp.1726-1745, 2007.
[11] C.-J. Chu, V. S. Tseng and T. Liang, Mining temporal rare utility itemsets in large databases using relative utility thresholds, *International Journal of Innovative Computing, Information and Control*, vol.4, no.11, pp.2775-2792, 2008.
[12] Y. L. Chen and C. H. Weng, Mining fuzzy association rules from questionnaire data, *Knowledge-Based Systems*, vol.22, no.1, pp.46-56, 2009.
[13] Y. J. Lee, J. W. Lee, D. J. Chai, B. H. Hwang and K. H. Ryu, Mining temporal interval relational rules from temporal data, *Journal of Systems and Software*, vol.82, no.1, pp.155-167, 2009.
[14] R. Agrawal, T. Imielinski and A. Swami, Mining association rules between sets of items in large databases, *Proc. of the 1993 ACM SIGMOD International Conference on Management of Data*, pp.207-216, 1993.
[15] B. Liu, W. Hsu and Y. Ma, Mining association rules with multiple minimum supports, *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp.337-341, 1999.
[16] G. D. Eppen, W. A. Hanson and R. K. Martin, Bundling – New products, new markets, low risk, *MIT Sloan Management Review*, vol.32, no.4, pp.7-14, 1991.
[17] A. Ovans, Make a bundle bundling, *Harvard Business Review*, vol.75, no.6, pp.18-20, 1997.
[18] R. Srikant and R. Agrawal, Mining quantitative association rules in large relational tables, *Proc. of the 1996 ACM SIGMOD International Conference on Management of Data*, pp.1-12, 1996.
[19] P. Y. Hsu, Y. L. Chen and C. C. Ling, Algorithms for mining association rules in bag databases, *Information Sciences*, vol.166, pp.31-47, 2004.
[20] J. Han, J. Pei and Y. Yin, Mining frequent pattern without candidate generation, *Proc. of the 2000 ACM SIGMOD International Conference on Management of Data*, pp.1-12, 2000.
[21] R. Agrawal and R. Srikant, Fast algorithms for mining association rules, *Proc. of the 20th International Conference on Very Large Data Bases*, pp.487-499, 1994.