

# Mining Flexible Multiple-Level Association Rules in All Concept Hierarchies

## (Extended Abstract)

Li Shen and Hong Shen

School of Computing and Information Technology  
Griffith University, Nathan, QLD4111, Australia  
{L.Shen, H.Shen}@cit.gu.edu.au

**Abstract.** We introduce the problem of mining FML (flexible multiple-level) association rules in all concept hierarchies related to a set of user-interested database attributes, as interesting association rules among data items may occur at multiple levels of multiple relevant concept hierarchies. We present a complete classification of all FML rules and show that direct application of previous research can find only a small part of strong FML rules. We propose an efficient method to generate all strong FML rules in all concept hierarchies.

## 1 Introduction

Association discovery is one of the most important problems in data mining. Previous research emphasized techniques for mining association rules at a *single* concept level [1, 2, 6–8, 10]. Some works extended this paradigm to mining rules at *generalized* abstract levels [9] or at *multiple* levels [4]. However, they can only find multiple-level rules in a *fixed* concept hierarchy. Our study is motivated by the goal of mining *more flexible* multiple-level association rules in *all* concept hierarchies related to a set of user-interested attributes.

We assume that the database contains two parts: (1) an item data set  $I$  and a database relation containing the description of each item in  $I$ ; (2) a transaction data set  $T$ , which consists of a set of transactions  $\langle T_i, a \rangle$ , where  $T_i$  is a transaction identifier and  $a \subseteq I$ . The *support* of an itemset  $a (\subseteq I)$  in  $T$ ,  $\sigma(a/T)$ , is the ratio of the number of transactions (in  $T$ ) containing  $a$  to the total number of transactions in  $T$ . An *association rule* is an expression  $a \Rightarrow b$ , where  $a, b \subseteq I$  and  $a \cap b = \emptyset$ . The *confidence* of  $a \Rightarrow b$  in  $T$  is the ratio of  $\sigma(a \cup b/T)$  to  $\sigma(a/T)$ . We use `minsup` and `minconf` to denote the user-specified minimum support and confidence respectively. An itemset  $a$  is *large* if  $\sigma(a/T) \geq \text{minsup}$ ; an association rule  $a \Rightarrow b$  is *strong* if  $\sigma(a \cup b/T) \geq \text{minsup}$  and  $\frac{\sigma(a \cup b/T)}{\sigma(a/T)} \geq \text{minconf}$ . The task of mining association rules is to generate all *strong* association rules.

Clearly, the above problem statement doesn't consider the possible concept hierarchies implied by the description relation of  $I$ . Since interesting associations may occur in these concept hierarchies, in this paper, we propose the problem of mining FML association rules in all concept hierarchies related to a set of user-interested database attributes and present an efficient solution for that.

## 2 Problem Description

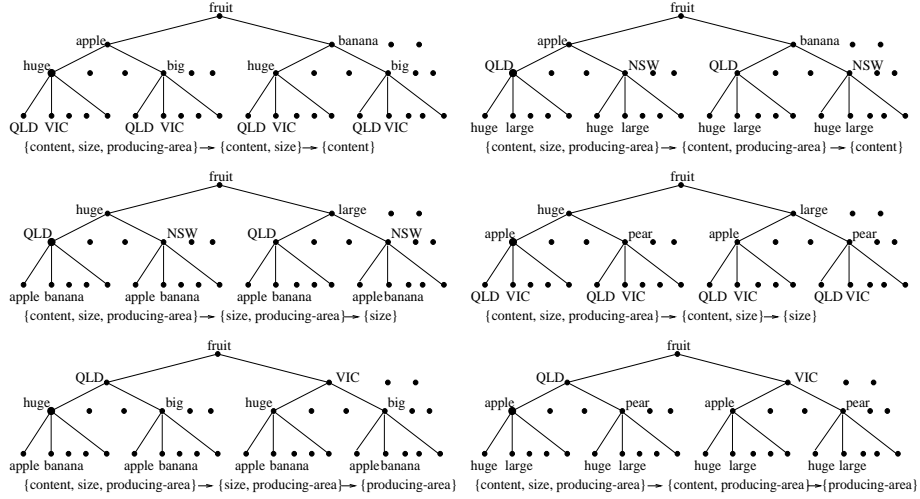
Given a set of user-interested database attributes and some query, we can generate an attribute-information-encoded and task-relevant set of transactions (called *encoded transaction table*) instead of the original database, see [4] for details. Table 1(b) is an abstract example, where any item is encoded by the attribute-code relations based on Table 1(a). For example, ‘121’ denotes an item with attribute values {apple, large, Queensland} because the first digit, ‘1’, represents ‘apple’ corresponding to the first attribute *content*, the second digit, ‘2’, represents ‘large’ corresponding to the second attribute *size*, and the third digit, ‘1’, represents ‘Queensland’ corresponding to the third attribute *producing-area*. Our study will be based on the encoded transaction table.

**Table 1.** (a) Attribute-code table (left) (b) Encoded transaction table (right)

attribute-code	content	size	producing-area	TID	Items
1	apple	huge	Queensland	$T_1$	{111, 121, 211, 221}
2	banana	large	Victoria	$T_2$	{111, 211, 323}
3	pear	big	New-South-Wales	$T_3$	{111, 122, 211, 221, 413}
...	...	...	...	...	...

A concept hierarchy can be defined on a set of database attribute domains such as  $D(a_1), \dots, D(a_n)$ , where, for  $i \in [1, n]$ ,  $a_i$  denotes an attribute and  $D(a_i)$  denotes the domain of  $a_i$ . The concept hierarchy is usually partially ordered according to a general-to-specific ordering. The most general concept is the null description *ANY*; whereas the most specific concepts correspond to the specific attribute values in the database. Given a set of  $D(a_1), \dots, D(a_n)$ , we define a concept hierarchy  $H$  as follows:  $H^n \rightarrow H^{n-1} \rightarrow \dots \rightarrow H^0$ , where  $H^i = D(a_1^i) \times \dots \times D(a_n^i)$  for  $i \in [0, n]$ , and  $\{a_1, \dots, a_n\} = \{a_1^n, \dots, a_n^n\} \supset \{a_1^{n-1}, \dots, a_{n-1}^{n-1}\} \supset \dots \supset \emptyset$ . Here,  $H^n$  represents the set of concepts at the primitive level,  $H^{n-1}$  represents the concepts at one level higher than those at  $H^n$ , etc., and  $H^0$ , the highest level hierarchy, may contain solely the most general concept *ANY*. We also use  $\{a_1^n, \dots, a_n^n\} \rightarrow \{a_1^{n-1}, \dots, a_{n-1}^{n-1}\} \rightarrow \dots \rightarrow \{a_1^1\}$  to denote  $H$  directly, and  $H^0$  may be omitted here. For example, there are 6 concept hierarchies related to the attribute set {content, size, producing-area}. Based on Table 1(a), every hierarchy of them implies a concept tree like Fig. 1. For more details about concept hierarchies, see [5].

We introduce FML items to represent concepts at any level of any hierarchy. We regard \*, called *trivial-digit*, as a special digit and insert it into the set of all digits. An FML item is defined as a sequence of digits (may include \*), in which \* means that we don’t care about its corresponding attribute value. For example, 12\*, representing {apple, large, *trivial*}, means large apples. Thus, any concept at any level of any concept hierarchy can be represented by an FML item. For simplicity, any item (or itemset) mentioned later always refers to an FML item (or itemset).



**Fig. 1.** Concept hierarchies related to attribute set  $\{\text{content, size, producing-area}\}$

Given an item  $x = x_1 x_2 \dots x_n$ , we define its *flat-set*  $S_f(x) = \{(i, x_i) \mid i \in [1, n] \text{ and } x_i \neq *\}$ . Given two items  $x$  and  $y$ ,  $x$  is called a *generalized-item* of  $y$  if  $S_f(x) \subseteq S_f(y)$ , which means that  $x$  represents a higher-level concept which contains the lower-level concept represented by  $y$ . Thus,  $*5*$  is a generalized-item of  $35*$  due to  $S_f(*5*) = \{(2, 5)\} \subseteq S_f(35*) = \{(1, 3), (2, 5)\}$ . If  $S_f(x) = \emptyset$ ,  $x$  is called *trivial-item*, which represents the most general concept *ANY*.

## 2.1 FML Itemsets and FML Rules

Let  $T$  be an encoded transaction table,  $t$  a transaction in  $T$ ,  $x$  an item, and  $c$  an itemset. We say that: (1)  $t$  *supports*  $x$  if there exists an item  $y$  in  $t$  such that  $x$  is a generalized-item of  $y$ ; (2)  $t$  *supports*  $c$  if  $t$  supports every item in  $c$ . The *support* of an itemset  $c$  in  $T$ ,  $\sigma(c/T)$ , is the ratio of the number of transactions (in  $T$ ) which support  $c$  to the total number of transactions in  $T$ . Given a *minsup*, an itemset  $c$  is *large* if  $\sigma(c/T) \geq \text{minsup}$ , otherwise it is *small*.

Let  $c_1 = \{12*, 23*\}$  and  $c_2 = \{12*, 23*, 1**\}$ . Note that  $1**$  is actually a redundant item in  $c_2$  because it can not provide any specific information due to the existence of  $12*$ . Every transaction supporting  $12*$  must also support  $1**$  and so the supports of  $c_1$  and  $c_2$  are always the same. As a result,  $c_1$  and  $c_2$  have the same meaning in our study. Given an itemset  $c$ , we define its *simplest-form*  $F_s(c) = \{x \in c \mid \forall y \in c, S_f(x) \not\subseteq S_f(y)\}$ , and its *complete-form*  $F_c(c) = \{x \mid S_f(x) \subseteq S_f(y), y \in c\}$ . Since an item in an itemset is redundant iff it is a generalized-item of another item in the same itemset, the following three statements about two itemsets are equivalent to each other: (1) they have the same meaning in our study, (2) their simplest-forms are the same, (3) their complete-forms are the same. Thus, itemsets with the same simplest (or complete) form

will not be distinguished in our study. We usually use the simplest-form to denote an itemset. For  $c_1$  and  $c_2$  mentioned above,  $F_s(c_1) = F_s(c_2) = \{12*, 23*\}$  and  $F_c(c_1) = F_c(c_2) = \{12*, 23*, 1***, 2***, *2*, *3*, ***\}$ . So we treat  $c_1$  and  $c_2$  as one itemset, which can be denoted by  $F_s(c_1)$  or  $F_s(c_2)$ .

Given two itemsets  $c_1$  and  $c_2$ , we define: (1)  $c_1 \sqsubseteq c_2$  if  $F_c(c_1) \subseteq F_c(c_2)$ ; (2)  $c_1 \sqsubset c_2$  if  $F_c(c_1) \subset F_c(c_2)$ . We say that  $c_1$  is a *generalized-subset* of  $c_2$  if  $c_1 \sqsubseteq c_2$ , and a *proper generalized-subset* of  $c_2$  if  $c_1 \sqsubset c_2$ . For example,  $\{12*, *23, 2*1, *3*\}$  is a generalized-subset of  $\{123, 2*1, 23*\}$ . Clearly, every transaction supporting an itemset must also support all of its generalized-subsets. Hence, all generalized-subsets of a large itemset must also be large.

Direct application of previous research [1] is not suitable for our problem description because some more specific FML rules can be generated in our study. For example, assume that  $\{**34, 2***\}$  is a large itemset, besides generating the rule  $\{**34\} \Rightarrow \{2***\}$ , it could be informative to also show that  $\{**3*\} \Rightarrow \{**34, 2***\}$ . To include the latter rule, we present a new description of all the desired FML rules. Let  $l$  be an itemset and  $a \sqsubseteq l$ . An *FML association rule* is an expression  $F_s(a) \Rightarrow F_s(F_c(l) - F_c(a))$ , denoted by  $r(l, a)$ . FML association rules are also called rules directly. Given a minsup and minconf, a rule  $r(l, a)$  is *strong* if  $\sigma(l/T) \geq \text{minsup}$  and  $\frac{\sigma(l/T)}{\sigma(a/T)} \geq \text{minconf}$ , otherwise it is *weak*. The problem of mining FML association rules is to generate all *strong* FML rules, which can be decomposed into two parts: (1) finding all large FML itemsets, (2) finding all strong FML rules from all large FML itemsets.

We classify all itemsets  $c$ 's into three types as follows according to whether they are associated with a single hierarchy or a single level. (1) *SHSL (Single-Hierarchy-Single-Level)*: all items in  $c$  can be confined to a fixed level of a fixed hierarchy, e.g.  $\{1**, 2***\}$  and  $\{111, 232\}$ . (2) *SHML (Single-Hierarchy-Multiple-Level)*: all items in  $c$  can be confined to a fixed hierarchy, but not to any fixed level of the hierarchy, e.g.  $\{11*, 2***\}$ . (3) *MHML (Multiple-Hierarchy-Multiple-Level)*: all items in  $c$  can not be confined to any fixed hierarchy<sup>1</sup>, e.g.  $\{11*, ***2\}$ .

Let  $r(l, a)$  be a rule. If  $a \sqsubseteq l$ ,  $r(l, a)$  is called a *basic* rule; otherwise, it is an *extensive* rule. We classify all rules  $r(l, a)$ 's into five types as follows. (1) *BSHSL (Basic-SHSL)*:  $r(l, a)$  is basic and  $l$  is SHSL, e.g.  $\{1***\} \Rightarrow \{2***\}$  and  $\{111\} \Rightarrow \{232\}$ . (2) *BSHML (Basic-SHML)*:  $r(l, a)$  is basic and  $l$  is SHML, e.g.  $\{11*\} \Rightarrow \{2***\}$ . (3) *BMHML (Basic-MHML)*:  $r(l, a)$  is basic and  $l$  is MHML, e.g.  $\{***2\} \Rightarrow \{11*\}$ . (4) *ESHML (Extensive-SHML)*:  $r(l, a)$  is extensive and  $l$  is SHSL/SHML<sup>2</sup>, e.g.  $\{1***\} \Rightarrow \{12*\}$ . (5) *EMHML (Extensive-MHML)*:  $r(l, a)$  is extensive and  $l$  is MHML, e.g.  $\{1**, ***2\} \Rightarrow \{12*\}$ .

Previous studies can only generate either some strong BSHSL rules at the primitive level [1, 2, 6–8, 10], or some strong BSHSL rules in a fixed concept hierarchy [4, 9]. Han and Fu presented a discussion in [4] about some more flexible rules, however, this only covers a small part of strong BSHML rules in a fixed

<sup>1</sup> This statement implies that  $c$  is related to at least two different concept levels so that no Multiple-Hierarchy-Single-Level itemsets exist in this classification.

<sup>2</sup> Since any extensive rule is always associated with at least two concept levels, no Extensive-SHSL rules exist in this classification.

concept hierarchy. We will propose an efficient approach to generate all strong FML rules. We summarize the relevant notation in Table 2, where some concepts will be introduced later.

**Table 2.** Notation used in this paper

$T$	Encoded transaction table	$O(l)$	Set of all outcomes of $l$
$\sigma(c/T)$	Support of itemset $c$ in $T$	$O_k(l)$	Set of all $ k $ -outcomes of $l$
$S_f(x)$	Flat-set of item $x$	$O^e(l)$	Set of all elementary outcomes of $l$
$F_s(c)$	Simplest-form of itemset $c$	$O_k^e(l)$	Set of all elementary $ k $ -outcomes of $l$
$F_c(c)$	Complete-form of itemset $c$	$C_k$	Set of candidate $[k]$ -itemsets
$r(l, a)$	$F_s(a) \Rightarrow F_s(F_c(l) - F_c(a))$	$L_k$	Set of all large $[k]$ -itemsets
$\hat{r}(l, o)$	$F_s(F_c(l) - o) \Rightarrow F_s(o)$	$H_m$	Set of candidate $ m $ -outcomes of $l_k$
$F_o(l, a)$	$F_c(l) - F_c(a)$	$U_m$	Set of all strong $ m $ -outcomes of $l_k$
$G_k(c)$	Set of all $[k]$ -generalized-subsets of itemset $c$		
$V_m(a, l)$	Set of all $ m $ -suboutcomes of $a$ versus $l$		
$E_r(a)$	Extension-result of self-extensive $[2^m - 1]$ -itemset $a$ , where $m > 1$		

### 3 Finding Large FML Itemsets

Given an itemset  $c$ , we call the number of elements in  $F_s(c)$  its *size*, and the number of elements in  $F_c(c)$  its *weight*. An itemset of size  $j$  and weight  $k$  is called a  $(j)$ -*itemset*,  $[k]$ -*itemset* or  $(j)[k]$ -*itemset*. Let  $c$  be a  $(j)[k]$ -itemset. We use  $G_i(c)$  to indicate *the set of all  $[i]$ -generalized-subsets of  $c$* , where  $i \leq k$ . Thus the set of all  $[k - 1]$ -generalized-subsets of  $c$  can be generated as follows:  $G_{k-1}(c) = \{F_s(F_c(c) - \{x\}) \mid x \in F_s(c)\}$ , and the size of  $G_{k-1}(c)$  is  $j$ . Hence, for  $k \geq 1$ ,  $c$  is a  $(1)$ -itemset iff  $|G_{k-1}(c)| = 1$ . With this observation, we call a  $[k - 1]$ -itemset  $a$  *self-extensive* if there exists a  $(1)[k]$ -itemset  $b$  such that  $G_{k-1}(b) = \{a\}$ ; at the same time, we call  $b$  *the extension-result of  $a$* . Thus, all self-extensive itemsets  $a$ 's can be generated from all  $(1)$ -itemsets  $b$ 's as follows:  $F_c(a) = F_c(b) - F_s(b)$ .

Let  $b$  be a  $(1)$ -itemset and  $F_s(b) = \{x\}$ . By  $|F_c(b)| = |\{y \mid S_f(y) \subseteq S_f(x)\}| = 2^{|S_f(x)|}$ , we have that  $b$  is a  $(1)[2^{|S_f(x)|}]$ -itemset. Let  $a$  be the self-extensive itemset generated by  $b$ , i.e.,  $a$  is a  $[2^{|S_f(x)|} - 1]$ -itemset such that  $F_c(a) = F_c(b) - F_s(b)$ . Thus, if  $|S_f(x)| > 1$ , there exist  $y, z \in F_s(a)$  and  $y \neq z$  such that  $S_f(x) = S_f(y) \cup S_f(z)$ . Clearly, this property can be used to generate the corresponding extension-result from any self-extensive  $[2^m - 1]$ -itemset, where  $m > 1$ . For simplicity, given a self-extensive  $[2^m - 1]$ -itemset  $a$ , where  $m > 1$ , we directly use  $E_r(a)$  to denote its extension-result. For example,  $E_r(\{12*, 1*3, *23\}) = \{123\}$ .

Our algorithm makes multiple passes over the data. In the first pass, we count the supports of all  $[2]$ -itemsets and then select all large ones. In pass  $k - 1$ , we start with  $L_{k-1}$ , the set of all large  $[k - 1]$ -itemsets, and use  $L_{k-1}$  to generate  $C_k$ , a superset of all large  $[k]$ -itemsets. We call elements in  $C_k$  *candidate itemsets*. We count the support for these itemsets during the pass over the data. At the end of the pass, we determine which of these itemsets are actually large, and

obtain  $L_k$  for the next pass. This process continues until no new large itemsets are found. Note that  $L_1 = \{\{\text{trivial-item}\}\}$  because  $\{\text{trivial-item}\}$  is the unique [1]-itemset and supported by all transactions.

**Algorithm 1.** *Finding all large FML itemsets.*

**Input:** (1)  $T$ , an encoded transaction table, (2)  $\text{minsup}$ , the minimum support.

**Output:**  $L$ , a set of all large FML itemsets.

```

 $L_1 := \{\{\text{trivial-item}\}\};$                                 /*  $\sigma(\{\text{trivial-item}\}/T) = 1$  */
 $L_2 := \{a \mid a \text{ is a [2]-itemset and } \sigma(a/T) \geq \text{minsup}\};$  /* the first pass */
for ( $k := 3; L_{k-1} \neq \emptyset; k++$ ) do {                /* pass  $k - 1$  */
     $C_k := \text{getcanditemset}(L_{k-1});$ 
    for each transaction  $t$  in  $T$  do {
         $C_t := \{c \in C_k \mid t \text{ supports } c\};$ 
        for each candidate  $c$  in  $C_t$  do  $c.\text{support}++$ ;
    }
     $L_k := \{c \in C_k \mid (\sigma(c/T) := c.\text{support}/|T|) \geq \text{minsup}\};$ 
}
 $L := \bigcup_k L_k;$ 

function  $\text{getcanditemset}(L_{k-1} : \text{set of all large } [k-1]\text{-itemsets})$  {
     $C_t := \{c \mid c = F_s(l_1 \cup l_2), c \text{ is a } [k]\text{-itemset, } l_1, l_2 \in L_{k-1}\};$  /* join */
    for every  $c \in C_t$  do if  $G_{k-1}(c) \not\subseteq L_{k-1}$  then delete  $c$  from  $C_t$ ; /* prune */
    if  $(\log_2 k)$  is not an integer then  $C_k^1 := \emptyset$ ;
    else  $C_k^1 := \{E_r(l) \mid l \text{ is self-extensible, and } l \in L_{k-1}\};$  /* self-extension */
    return  $C_t \cup C_k^1$ ;
}

```

It is obvious that the main procedure of Alg. 1 employs the idea described before. Now we focus on the function *getcanditemset*, which generates a superset of all large  $[k]$ -itemsets from  $L_{k-1}$ , the set of all large  $[k-1]$ -itemsets.

The key observation of our method is that all  $[k-1]$ -generalized-subsets of a large  $[k]$ -itemset must also be large. So a necessary condition for a candidate  $[k]$ -itemset  $c$  is  $G_{k-1}(c) \subseteq L_{k-1}$ . Furthermore, it is easy to see that the following two properties for  $G_{k-1}(c)$  and  $k \geq 3$  hold: (1) if  $|G_{k-1}(c)| > 1$ , then there exist  $a, b \in G_{k-1}(c)$  such that  $F_s(c) = F_s(a \cup b)$ ; (2) if  $|G_{k-1}(c)| = 1$ , then  $c$  is a  $(1)[2^m]$ -itemset, where  $k = 2^m$ , and so there exists a self-extensive  $[2^m - 1]$ -itemset<sup>3</sup>  $a$  such that  $G_{k-1}(c) = \{a\}$  and  $F_s(c) = F_s(E_r(a))$ . For example, given [9]-itemset  $\{123, *3*\}$  and [8]-itemset  $\{123\}$ , we have  $G_8(\{123, *3*\}) = \{\{123\}, \{12*, 1*3, *23, *3*\}\}$  and  $G_7(\{123\}) = \{\{12*, 1*3, *23\}\}$ .

With the above observation, the function *getcanditemset* uses three steps to generate the desired result from  $L_{k-1}$ . In the *join* step, the function joins  $L_1^{k-1}$  with  $L_1^{k-1}$  to obtain a superset of  $C_k - C_k^1$ , where  $C_k^1$  denotes the set of candidate  $(1)[k]$ -itemsets. The itemset  $F_s(l_1 \cup l_2)$  is inserted in  $C_t$  if it is a  $[k]$ -itemset and  $l_1, l_2 \in L_{k-1}$ . In the *prune* step, the function deletes all itemsets  $c \in C_t$  such that some  $[k-1]$ -generalized-subset of  $c$  is not in  $L_{k-1}$ . In the *self-extension* step,

<sup>3</sup>  $k \geq 3$  implies  $m > 1$  for  $k = 2^m$ .

$C_k^1$  is generated independently. Since only  $[2^m - 1]$ -itemsets can be self-extensive itemsets, where  $m$  is an integer,  $L_{k-1}$  contains self-extensive itemsets only when  $\log_2 k$  is an integer. Thus, in this step, if  $\log_2 k$  is an integer, the extension-result of each self-extensive itemset in  $L_{k-1}$  is inserted into  $C_k^1$ ; otherwise,  $C_k^1$  is empty. Finally, the function returns  $C_t \cup C_k^1$ , a superset of all large  $[k]$ -itemsets.

We evaluate the computation cost of Alg. 1. An algorithm for finding all large FML itemsets always needs to count supports for a set of *candidates*. Hence, its computation cost can often be denoted by  $O(\sum_{c \in C} (g(c) + s(c)))$ , where  $C$  is the set of all candidates,  $g(c)$  is the cost for generating  $c$  as a candidate,  $s(c)$  is the cost for counting the support of  $c$ . Since obtaining the support of  $c$  requires a scan of a very large data set,  $s(c)$  is often much greater than  $g(c)$ . So the above computation cost can be simplified as  $O(\sum_{c \in C} s(c))$ .

A small itemset is *adjacent* if all its proper generalized-subsets are large. Thus itemsets can be divided into 3 types: (1) large, (2) small and adjacent, (3) small and not adjacent. For any itemset of Type 3, we can determine that it is small by finding out some of its small generalized-subsets. But for any itemset of Types 1 and 2, we can not determine if it is small or large unless we get its support. Hence, for completeness, the set of all candidate itemsets  $C$  must contain all Types 1 and 2 itemsets. Furthermore it is obvious that for any small  $[k]$ -itemset, it is adjacent iff all its  $[k - 1]$ -generalized-subsets are large. Thus Alg. 1 generates the smallest  $C$ , which precisely consists of all Types 1 and 2 itemsets. Hence, if the method of support-counting is fixed, Alg. 1 with cost  $O(\sum_{c \in C} s(c))$  is optimal for solving this problem because  $C$  contains only those necessary candidates.

## 4 Finding Strong FML Rules

As mentioned before, we use  $r(l, a)$  to denote rule  $F_s(a) \Rightarrow F_s(F_c(l) - F_c(a))$ , where  $l$  is an itemset and  $a \sqsubseteq l$ . Now we use  $F_o(l, a)$  to denote  $F_c(l) - F_c(a)$ , and say that  $F_o(l, a)$  is an *outcome-form* of  $l$  or the *outcome-form* of  $r(l, a)$ . Note that  $F_o(l, a)$  represents only a specific form rather than a meaningful itemset so that it is not equivalent to any other itemset whose simplest-form is  $F_s(F_o(l, a))$ . Outcome-forms are also called *outcomes* directly. Clearly, the corresponding relationship between rules and outcomes is one-to-one. An outcome is *strong* if it corresponds to a strong rule. Thus, all strong rules related to a large itemset can be obtained by finding all strong outcomes of this itemset.

Let  $l$  be an itemset. We use  $O(l)$  to denote the set of all outcomes of  $l$ , i.e.,  $O(l) = \{F_o(l, a) \mid a \sqsubseteq l\}$ . Thus, from  $O(l)$ , we can output all rules related to  $l$ :  $F_s(F_c(l) - o) \Rightarrow F_s(o)$  (denoted by  $\hat{r}(l, o)$ ), where  $o \in O(l)$ . Clearly,  $r(l, a)$  and  $\hat{r}(l, o)$  denote the same rule iff  $o = F_o(l, a)$ .

Let  $o, \hat{o} \in O(l)$ . We say that (1)  $o$  is a  $|k|$ -*outcome* of  $l$  if  $o$  exactly contains  $k$  elements; (2)  $\hat{o}$  is a *suboutcome* of  $o$  versus  $l$  if  $\hat{o} \sqsubseteq o$ . We use  $O_k(l)$  to denote the set of all  $|k|$ -outcomes of  $l$ , use  $V_m(o, l)$  to denote the set of all  $|m|$ -suboutcomes of  $o$  versus  $l$ . Let  $o$  be an  $|m + 1|$ -outcome of  $l$  and  $m \geq 1$ . If  $|V_m(o, l)| = 1$ ,  $o$  is called an *elementary* outcome; otherwise,  $o$  is a *nonelementary* outcome.

Let  $o$  be an elementary  $|m+1|$ -outcome of  $l$ . In this case, there exist  $\hat{o} \in O_m(l)$  and  $x \in F_c(l)$  such that  $V_m(o, l) = \{\hat{o}\}$  and  $\hat{o} \cup \{x\} = o$ , which implies  $o = \{y \in F_c(l) \mid S_f(y) \supseteq S_f(x)\}$ . We use  $P_u(l, x)$  to denote  $\{y \in F_c(l) \mid S_f(y) \supseteq S_f(x)\}$ . Thus, the set of all elementary outcomes of  $l$ , denoted by  $O^e(l)$ , equals  $\{P_u(l, x) \mid x \in F_c(l)\}$ . We also use  $O_k^e(l)$  to denote the set of all elementary  $|k|$ -outcomes of  $l$ .

Let  $o$  be a nonelementary  $|m+1|$ -outcome of  $l$ . In this case, we have  $|V_m(o, l)| > 1$ . Furthermore, for any  $o_1, o_2 \in V_m(o, l)$  such that  $o_1 \neq o_2$ , we have  $o = o_1 \cup o_2$ .

Let  $r(l, a)$  and  $r(l, b)$  be two rules. We say that  $r(l, a)$  is an *instantiated-rule* of  $r(l, b)$  if  $b \sqsubseteq a$ . Clearly,  $b \sqsubseteq a$  implies  $\sigma(b/T) \geq \sigma(a/T)$  and  $\frac{\sigma(l/T)}{\sigma(a/T)} \geq \frac{\sigma(l/T)}{\sigma(b/T)}$ . Hence, all instantiated-rules of a strong rule must also be strong. Let  $l$  be a large itemset,  $o_1, o_2 \in O(l)$ ,  $o_1 = F_o(l, a)$  and  $o_2 = F_o(l, b)$ . It is straightforward to get that (1)  $F_o(l, a) \subseteq F_o(l, b)$  iff  $b \sqsubseteq a$ ; (2)  $\hat{r}(l, o_1) = r(l, a)$ ; and (3)  $\hat{r}(l, o_2) = r(l, b)$ . Therefore,  $\hat{r}(l, o_1)$  is an instantiated-rule of  $\hat{r}(l, o_2)$  iff  $o_1 \subseteq o_2$ , which implies that all suboutcomes of a strong outcome must also be strong.

This characteristic is similar to the property that all generalized-subsets of a large itemset must also be large. Hence, from a large itemset  $l$ , we first generate all strong rules with  $|1|$ -outcomes<sup>4</sup>. We then use all these strong  $|1|$ -outcomes and the function *getcandoutcome* described below to generate all possible strong  $|2|$ -outcomes of  $l$ , from which all strong rules with  $|2|$ -outcomes can be produced, etc. An algorithm using this idea is presented below.

**Algorithm 2.** Finding all strong FML rules from all large FML itemsets.

**Input:** (1)  $L$ , a set of large FML itemsets, (2) *minconf*, the minimum confidence.

**Output:**  $R$ , a set of all strong FML rules.

```

R := ∅;
for every large  $[k]$ -itemset  $l_k \in L$ ,  $k > 2$  do {
   $H_1 := \{\{a\} \mid a \in F_s(l_k)\}$ ;
   $R_i := \text{getrules}(l_k, H_1)$ ;
   $R := R \cup R_i$ ;
}

function getrules( $l_k$ : large  $[k]$ -itemset,  $H_m$ : set of candidate  $|m|$ -outcomes of  $l_k$ ) {
   $R_1 := \emptyset$ ;
  for every  $h_m \in H_m$  do {
     $\text{conf} := \frac{\sigma(F_s(l_k)/T)}{\sigma(F_s(F_c(l_k) - h_m)/T)}$ ;
    if ( $\text{conf} \geq \text{minconf}$ ) then insert rule  $F_s(F_c(l_k) - h_m) \Rightarrow F_s(h_m)$ 
      with confidence= $\text{conf}$  and support= $\sigma(F_s(l_k)/T)$  into  $R_1$ ;
    else delete  $h_m$  from  $H_m$ ;
  }
   $R_2 := \emptyset$ ;
  if ( $k > m + 1$ ) and  $H_m \neq \emptyset$  then {
     $H_{m+1} := \text{getcandoutcome}(l_k, H_m)$ ;
     $R_2 := \text{getrules}(l_k, H_{m+1})$ ;
  }
}

```

<sup>4</sup> We ignore the  $|0|$ -outcome  $\emptyset$ , which only corresponds to all trivial strong rules:  $F_s(l) \Rightarrow \emptyset$  for every large itemset  $l$ .

```

}
return  $R_1 \cup R_2$ ;
}

function getcandoutcome( $l_k$ :  $[k]$ -itemset,  $U_m$ : set of all strong  $|m|$ -outcomes of  $l_k$ ) {
   $U_t := \{o \mid o = o_1 \cup o_2, o \in O_{m+1}(l_k), o_1, o_2 \in U_m\}$ ; /* join */
   $U_t := U_t \cup O_{m+1}^e(l_k)$ ; /* elementary */
  for every  $o \in U_t$  do if  $V_m(o, l_k) \not\subseteq U_m$  then delete  $o$  from  $U_t$ ; /* prune */
  return  $U_t$ ;
}

```

This algorithm uses every large itemset to generate the desired rules. Let  $l_k$  be the current large itemset considered by the algorithm. We use  $H_m$  to denote the set of candidate  $|m|$ -outcomes of  $l_k$ , which is a superset of all strong  $|m|$ -outcomes of  $l_k$ . Note that  $H_1$  should exactly contain all  $|1|$ -outcomes of  $l_k$  so that it equals  $O_1(l_k) = \{\{a\} \mid a \in F_s(l_k)\}$ . Thus, it is easy to see that the main procedure and function *getrules* in Alg. 2 employ the idea described before.

Now we focus on the function *getcandoutcome*. It takes two arguments:  $l_k$ , a large  $[k]$ -itemset, and  $U_m$ , the set of all strong  $|m|$ -outcomes of  $l_k$ . It returns a superset of all strong  $|m+1|$ -outcomes of  $l_k$ . The key observation is that all suboutcomes of a strong outcome must also be strong. Thus a necessary condition for a candidate  $|m+1|$ -outcomes  $o$  is  $V_m(o, l_k) \subseteq U_m$ . Furthermore, as we have shown, for  $m \geq 1$ : (1) if  $|V_m(o, l_k)| > 1$ , then there exist  $o_1, o_2 \in V_m(o, l_k)$  such that  $o = o_1 \cup o_2$ ; (2) if  $|V_m(o, l_k)| = 1$ , then  $o \in O_{m+1}^e(l_k)$ . With this observation, the function *getcandoutcome* generates all elementary candidate  $|m+1|$ -outcomes of  $l_k$  and all nonelementary candidate  $|m+1|$ -outcomes of  $l_k$  respectively. In the *join* step, the function joins  $U_m$  with  $U_m$  to obtain a superset of all strong and nonelementary  $|m+1|$ -outcomes of  $l_k$ . The form  $o_1 \cup o_2$  is inserted in  $U_t$  if it is an  $|m+1|$ -outcome of  $l_k$  and  $o_1, o_2 \in U_m$ . Next, in the *elementary* step, the function merges  $O_{m+1}^e(l_k)$ , the set of all elementary  $|m+1|$ -outcomes of  $l_k$ , into  $U_t$  to obtain a superset of all strong  $|m+1|$ -outcomes of  $l_k$ . Then, in the *prune* step, the function deletes all outcomes  $o \in U_t$  such that some  $|m|$ -suboutcome of  $o$  versus  $l_k$  is not in  $U_m$ . Finally, the function returns  $U_t$ , which is a superset of all strong  $|m+1|$ -outcomes of  $l_k$ .

We evaluate the computation cost of Alg. 2. The computation cost of an algorithm for generating all strong FML rules from all large FML itemsets can be denoted by  $O(|C|t + |C|s)$ , where  $C$  is the set of all candidate rules,  $s$  is the average cost for generating one element in  $C$ , and  $t$  is the average cost for computing the confidence of one element in  $C$ . In Alg. 2,  $|C| = \sum_{l \in L} |H(l)|$ , where  $H(l)$  is the set of all candidate outcomes of  $l$ . If we use the *hashing* technique described in [1] to search itemsets efficiently, both costs  $t$  and  $s$  can be limited. Hence, it is reasonable to take  $O(|C|)$  as the main cost of Alg. 2.

A weak rule is *adjacent* if all its instantiated-rules but itself are strong rules. Thus rules can be divided into 3 types: (1) strong, (2) weak and adjacent, (3) weak and not adjacent. For any rule of Type 3, we can determine its weakness by finding out some of its weak instantiated-rules. But for any rule of Types 1 and 2, we can not determine if it is strong or weak unless we get its confidence.

Hence, for completeness, the set of all candidate rules  $C$  must contain all Types 1 and 2 itemsets. A rule is called a  $|k|$ -rule if its outcome is a  $|k|$ -outcome. Clearly, for any weak  $|k|$ -rule, it is adjacent iff all its  $|k - 1|$ -instantiated-rules are large. Thus Alg. 2 generates the smallest  $C$ , which precisely consists of all Types 1 and 2 rules. So Alg. 2 with cost  $O(|C|t + |C|s)$  is optimal for bounded costs  $t$  and  $s$ .

## 5 Conclusions

We have extended the study of mining multiple-level association rules in a fixed hierarchy to mining FML (flexible multiple-level) rules in all concept hierarchies related to a set of user-interested database attributes. We have classified all FML rules into five types and observed that the direct application of the existing algorithms can generate only a small part of strong Basic-Single-Hierarchy-Single-Level rules, which are confined in a fixed hierarchy. To generate all strong FML rules, we propose efficient algorithms for finding all large FML itemsets and for finding all strong FML rules from large itemsets respectively. Our performance study shows that the computation costs of both algorithms are near optimal.

## Acknowledgments

The second author was supported by Australian Research Council (ARC) Large Grant A849602031. We thank Prof. Paul Pritchard for his useful comments.

## References

1. Agrawal R., Mannila H., Srikant R., Toivonen H., Verkamo A.: Fast Discovery of Association Rules. In: Advances in Knowledge Discovery and Data Mining. AAAI/MIT Press (1996)
2. Agrawal R., Imielinski I., Swami A.: Mining Associations between Sets of Items in Massive Databases. In: Proc. of ACM SIGMOD Conf., Washington D.C. (1993)
3. Chen M., Han J., Yu P.S.: Data Mining: An Overview from Database Perspective. In: IEEE Transactions on Knowledge and Data Engineering, Vol.8, No.6 (1996)
4. Han J., Fu Y.: Discovery of Multiple-level Association Rules from Large Databases. In: Proc. of the 21st VLDB Conference, Zurich, Switzerland (1995)
5. Han J., Fu Y.: Dynamic Generation and Refinement of Concept Hierarchies for Knowledge Discovery in Databases. In: AAAI'94 Workshop on Knowledge Discovery in Databases (KDD'94), Seattle, WA (1994) 157-168
6. Park J. S., Chen M., Yu P. S.: An Effective Hash-Based Algorithm for Mining Association Rules. In: Proc. of ACM-SIGMOD Conf., San Jose, CA (1995)
7. Savasere A., Omiecinski E., Navathe S.: An Efficient Algorithm for Mining Association Rules in Large Databases. In: 21st VLDB Conf., Zurich, Switzerland (1995)
8. Srikant R., Agrawal R.: Mining Quantitative Association Rules in Large Relational Tables. In: Proc. of ACM SIGMOD Conf., Montreal, Canada (1996)
9. Srikant R., Agrawal R.: Mining Generalized Association Rules. In: Proc. of the 21st VLDB Conf., Zurich, Switzerland (1995)
10. Zaki M. J., Parthasarathy S., Li W.: A Localized Algorithm for Parallel Association Mining. In: 9th Annual ACM Symposium on Parallel Algorithms and Architectures, Newport, Rhode Island (1997)