Mining Association Rules with Composite Items *

Xinfeng Ye,
Department of Computer Science,
University of Auckland,
New Zealand.

John A. Keane,
Department of Computation,
UMIST,
Manchester, UK.

Abstract

Association rules can be used to express relationships between items of data. Mining associations rules is to analysis the data in a database to discover "interesting" rules. Existing algorithms for mining association rules require that a record in the database contain all the data items in a rule. This requirement makes it difficult to discover certain useful rules in some applications. To solve the problem, this paper describes an algorithm for mining association rules with composite items. The algorithm has the potential to discover rules which cannot be discovered by existing algorithms.

1 Introduction

Data mining has attracted much research in recent years. The problem of mining association rules was introduced in [1]. Association rules can be used to express relationships between items of data. An association rule is an expression $X \Rightarrow Y$, where X and Y are sets of items. X and Y are termed as *itemsets*. For a set of transactions, where each transaction is a set of items, the meaning of an association rule is that the transactions which contain the items in X tend to also contain the items in Y.

For an itemset, say X, the *support* of X, denoted as $s(X)$, is the number of transactions that contain all items in X. Given a *minimum support* δ, an itemset X is *large* or is referred to as a *large itemset* if $s(X) \geq \delta$. The *confidence* of an association rule $X \Rightarrow Y$ is $\frac{s(X \cup Y)}{s(X)}$, i.e. the percentage of the transactions which contain X that also contain Y. Rules are useful if their confidence is above a *minimum confidence* value specified by the users.

An example of such a rule might be that 90% of patients who cough and have chest pain are suffering from pneumonia. Here, the two symptoms, coughing and chest pain, are the items in X; and, pneumonia is the item in Y. 90% is the confidence of the rule.

*This work is supported by Auckland University under grant A18/XXXXX/62090/F3414040, and by ESPRIT HPCN Project No 22693.

The problem of mining association rules is to find all rules $X \Rightarrow Y$ such that $X \cup Y$ is a large itemset and the confidence of $X \Rightarrow Y$ is above the minimum confidence.

Many algorithms have been developed for mining association rules [1, 2, 3, 4, 5, 6]. In these algorithms, if a transaction supports an itemset, the transaction must contain all the items in the itemset. This requirement makes it difficult to discover certain useful rules in some applications.

For example, assume that a patients database contains the symptoms of the patients and the diagnosis. The symptoms and the diagnosis are the items. A transaction in the database consists of a patient's symptoms and the corresponding diagnosis. The database is mined to discover the symptoms of diseases. Assume that:

(a) a disease, say A, makes the patients either show symptom B or C,

(b) the number of patients who show B (or C) and are diagnosed as having A is smaller than the minimum support, and

(c) the number of patients who either show B or C and are diagnosed as having A exceeds the minimum support.

From assumption (b), itemsets $\{A, B\}$ and $\{A, C\}$ are not large. As a result, rules $\{B\} \Rightarrow \{A\}$ and $\{C\} \Rightarrow \{A\}$ will not be generated. This is unsatisfactory. Ideally, the mining should be able to find a rule which indicates that, if a patient shows symptom B or C, then the patient is suffering from disease A. Combining B and C to form a new item $B \vee C$. A record in the database contains $B \vee C$ if the record contains either B or C. From assumption (c), it can be seen that itemset $\{A, B \vee C\}$ is large. As a result, rule $\{B \vee C\} \Rightarrow \{A\}$ will be generated.

The algorithm in this paper allows itemsets to include composite items like $B \vee C$ in the above example. A *composite item* consists of several items.

A transaction contains a composite item if the transaction contains at least one of the items in the composite item. As a result, not only the rules involving simple items, e.g. $\{B, C\} \Rightarrow \{A\}$, can be discovered, it is also possible to discover the rules involving composite items, e.g. $\{B \vee C\} \Rightarrow \{A\}, \{B \vee C, D\} \Rightarrow \{A\}$ etc. Thus, the algorithm in this paper has the potential to discover rules which cannot be discovered by existing algorithms.

Mining association rules can be decomposed into two steps:

1. the database is scanned to find all large itemsets;
2. association rules are generated from these large itemsets.

The first step is expensive, as the database needs to be scanned. The second step is relatively easier [5]. This paper concentrates on the first step: finding large itemsets.

The organisation of the paper is as follows: in §2, the definition of composite items are given; the algorithms for finding large composite items and large itemsets are discussed in §3; conclusions are given in §4.

2 Association Rules with Composite Items

Let $I = \{a_1, ..., a_m\}$ be the set of literals called *atomic items*. A *composite item* is formed by combining several atomic items. The general form of a composite item is $a_1 \vee ... \vee a_n$ where $a_j \in I$ for $1 \leq j \leq n$ and $a_i \neq a_j$ for $i \neq j$. A database consists of transactions where each transaction is a set of atomic items. A transaction *contains* a composite item if the transaction contains at least one of the atomic items which form the composite item. Atomic items and composite items will be referred to as *items* in general. An item is *large* if the number of transactions containing the item exceeds the minimum support.

For a composite item consisting of i atomic items, e.g. $a_1 \vee a_2 \vee ... \vee a_i$, the *(i-1)*-subitem of $a_1 \vee a_2 \vee ... \vee a_i$ is a composite item formed by selecting *i-1* atomic items from $\{a_1, a_2, ..., a_i\}$. For example, $A \vee B \vee C$ has three distinct 2-subitems. They are $A \vee B, A \vee C$ and $B \vee C$.

An *itemset* is a set of items such that none of the items in the set have common items. For example, $\{A, B \vee C\}$ is a valid itemset. However, $\{A, A \vee B, C \vee D, C \vee D \vee E\}$ is not a valid itemset as:

(a) A and $A \vee B$ have common item A, and

(b) $C \vee D$ and $C \vee D \vee E$ have common item $C \vee D$.

An *association rule* is an implication of the form $X \Rightarrow Y$ where X, Y and $X \cup Y$ are itemsets, $X \neq \emptyset, Y \neq \emptyset$ and $X \cap Y = \emptyset$.

An itemset consisting of i items is called an i-itemset. The *(i-1)*-subset of an i-itemset is an *(i-1)*-itemset which is formed by selecting *i-1* items from the i-itemset. An i-itemset has i distinct *(i-1)*-subsets. For example, a 3-itemset $\{A, B, C\}$ has three 2-subsets, i.e. $\{A, B\}, \{A, C\}$ and $\{B, C\}$.

3 Algorithm for Finding Large Itemsets

Finding large itemsets can be carried out in two steps:

- Step 1. Find all the large items and the large composite items[1].

- Step 2. Find all the large itemsets, i.e. the itemsets whose supports are greater than the minimum support.

  At this step, each large composite item is treated as an independent item like the atomic items.

It is assumed that the users are only interested with the composite items generated from a set of atomic items. This set of atomic items will be provided by the users.

For example, assume that:

(a) there are five atomic items *A, B, C, D* and *E*, and

(b) the users are only interested in the composite items generated from items *A, B* and *C*.

From *A, B* and *C*, four composite items, $A \vee B, A \vee C, B \vee C$ and $A \vee B \vee C$, can be formed. At step 1, the database is scanned to find the large items. Assume that:

(a) the large composite items are $A \vee B, A \vee C$ and $A \vee B \vee C$, and

(b) the large atomic items are *A, D* and *E*.

At step 2, itemsets are formed using the large items, i.e. $A, D, E, A \vee B, A \vee C$ and $A \vee B \vee C$; and, the database is scanned to find out which of

[1]An item (or a composite item) is large if the number of transactions containing the item (or composite item) exceeds minimum support.

these itemsets are large. The reason that only the large atomic items and the large composite items are used in constructing itemsets is because all the items in a large itemset must be large [3].

3.1 Patient Database Example

A patient database is shown in Figure 1(a). The symptoms and diseases are the items. It is assumed that:

(a) the minimum support is 3, and

(b) the users are interested in the composite items generated from set $\{A, B, C\}$.

From $\{A, B, C\}$, four composite items, i.e. $A \vee B \vee C, A \vee B, A \vee C$ and $B \vee C$ are formed.

The database is mined to find out the symptoms of the disease. First, the large composite items and the large atomic items are found. The large items found are shown in Figure 1(b) and (c). Then, the large atomic items in Figure 1(c) and the large composite items in Figure 1(b) are used to construct itemsets. According to the definition in §2, $X \neq \emptyset$ and $Y \neq \emptyset$ in rule $X \Rightarrow Y$ holds. Thus, only the large itemsets which contain more than one item can be used to generate rules. Figure 1(d) shows the large itemsets which will be used to generate rules.

From Figure 1(a), it can be seen that symptom A or B or C indicates that a patient suffers from disease S. However, without using the composite items, i.e. $A \vee B \vee C$ etc., S will not appear in any large itemsets. As a result, the symptoms of S cannot be discovered.

(a) **Database**

Symptoms	Diseases
D	T
A, C, D	S, T
B	S
A, B, D	S, T

(b) **Large Composite Items**

Composite Items	Support
$A \vee B \vee C$	3
$A \vee B$	3
$B \vee C$	3

(c) **Large Atomic Items**

Atomic Item	Support
D	3
S	3
T	3

(d) **Large 2- and 3-itemsets**

Itemsets	Support
$\{D, T\}$	3
$\{S, A \vee B \vee C\}$	3
$\{S, A \vee B\}$	3
$\{S, B \vee C\}$	3

Figure 1

3.2 Finding Large Items

Finding large atomic items is easy. First a counter is set up for each atomic item. Then, the database is scanned to check whether the atomic items are contained in the transactions, and the counters of the atomic items will be increased accordingly. When scanning is completed, the atomic items whose counters are greater than the minimum support are recorded as large items.

Finding large composite items is more complicated. This is because the number of composite items generated from a set of atomic items might be very large. Therefore, it is not practical to generate all composite items and find the large composite items in a single database scan. Instead, large composite items are found in several database scans. During each scan only a small set of composite items are checked.

The large composite items are found in several steps. At each step, first a candidate set is formed. The set contains the composite items which might be large. Then, the database is scanned to find out which items in the candidate set are large. Assume the large items found at the current step all consist of i atomic items. The *(i-1)*-subitems of the large items are used to form the candidate set of the next step.

When generating the candidate set, the following principle is used. If a transaction, say t, does not contain a composite item, say $a_1 \vee a_2 \vee ... \vee a_n$, then, according to the definition in §2, t does not contain any of $a_1, a_2, ..., a_n$. As a result, t does not contain any composite items which are generated from the items in $\{a_1, a_2, ..., a_n\}$. This means that, if $a_1 \vee a_2 \vee ... \vee a_n$ is not a large item, then none of the composite items generated from $\{a_1, a_2, ..., a_n\}$ are large. Thus, the composite items generated from $\{a_1, a_2, ..., a_n\}$ will not be included in the candidate set.

Initially the candidate set contains a single composite item which includes all the atomic items used to construct the composite items. For example, assume $\{a_1, a_2, a_3, a_4\}$ is the set of atomic items used to generate composite items. The item in the candidate set of the first step is $a_1 \vee a_2 \vee a_3 \vee a_4$. As discussed earlier, if $a_1 \vee a_2 \vee a_3 \vee a_4$ is not large, then none of the composite items generated from $\{a_1, a_2, a_3, a_4\}$ are large. Thus, we can stop looking for large composite items after the first step. If $a_1 \vee a_2 \vee a_3 \vee a_4$ is large, it is used to generate the items in step 2's candidate set. Each item in step 2's candidate set is a 3-subitem of $a_1 \vee a_2 \vee a_3 \vee a_4$. Thus, step 2's candidate set is $\{a_1 \vee a_2 \vee a_3, a_1 \vee a_2 \vee a_4, a_1 \vee a_3 \vee a_4, a_2 \vee a_3 \vee a_4\}$. The database is scanned to find the large items from the set. Assume that the large items are $a_1 \vee a_2 \vee a_3$ and $a_1 \vee a_2 \vee a_4$. The 2-subitems of these two items form the candidate set of step 3. Thus, step 3's candidate set is $\{a_1 \vee a_2, a_1 \vee a_3, a_1 \vee a_4, a_2 \vee a_3, a_2 \vee a_4\}$. As $a_1 \vee a_3 \vee a_4$ and $a_2 \vee a_3 \vee a_4$ are not large, from the earlier discussion, $a_1 \vee a_3, a_1 \vee a_4, a_2 \vee a_3$ and $a_2 \vee a_4$ are not large. Hence, step 3's candidate set is reduced to $\{a_1 \vee a_2\}$. It can be seen that, by observing the large items obtained at a step (e.g. step 2), it is possible to reduce the number of items in the candidate set of the next step (e.g. step 3).

The algorithm below is used to find all large composite items. Assume that $A = \{a_1, a_2, ..., a_k\}$ is the set of atomic items used to generate the composite items. CC_i represents the candidate set of a step. LC_i denotes the set of large composite items found at a step.

```
1.  $CC_k = \{a_1 \vee a_2 \vee ... \vee a_k\}$
2.  for ($i = k; CC_i \neq \emptyset$ && $i > 1; i--$) do
3.    for each transaction $t$ in the database do
4.      for each candidate $c \in CC_i$ do
5.        if contain(t, c) then c.count++ fi
6.      end_for_each
7.    end_for_each
8.    $LC_i = \{c \in CC_i \mid c.count \geq minimum\ support\}$
9.    if $i > 2$ then
10.       $CC_{i-1} = \emptyset$
11.       for each $ci \in LC_i$ do
12.           $CC_{i-1} = CC_{i-1} \cup \{sa \mid sa\ is\ an\ (i\text{-}1)\text{-}subitem\ of\ ci\}$
13.       end_for_each
14.       for each $c \in CC_{i-1}$ do
15.         if $\sim complete(c, LC_i)$ then
              $CC_{i-1} = CC_{i-1} - \{c\}$
            fi
16.       end_for_each
17.   fi
18. end_for
19. $LCI = \bigcup_i LC_i$

complete($a_1 \vee ... \vee a_i, LC$)
  let $S = \{a_1 \vee ... \vee a_i \vee a \mid a \in A - \{a_1, ..., a_i\}\}$
  if $S \subseteq LC$ then return 1 else return 0 fi
```

The candidate set of the first step is given at line 1. The database is scanned in lines 3-7. *contain(t, c)* is a predicate which determines whether transaction t contains the composite item c. LC_i contains all the large items found at the current step. A composite item is included in LC_i if the support of the item is greater than the minimum support (line 8).

The candidate set of the next step, CC_{i-1}, includes all the *(i-1)*-subitems of the large items in LC_i (lines 11-13). The items in candidate set CC_{i-1} will be checked against the items in LC_i to eliminate the items which are not large (lines 14-16). The elimination is carried out according to *complete* (line 15).

The composite items in LC_i have one more atomic item than the composite items in CC_{i-1}. In *complete*, a composite item, say c, in CC_{i-1} is extended to include one atomic item which is not in c. The extended items have the same number of atomic items as the items in LC_i. If all extended items are large (i.e. they are in LC_i), then c remains in CC_{i-1}. Otherwise, c is not large. As a result, c is removed from CC_{i-1} (line 15). This is because, as described earlier, if the extended item, say $c \vee a$, is not large, then none of $c \vee a$'s *(i-1)*-subsets are large. That is, c is not large.

When the candidate set becomes empty or all the composite items have been checked (i.e. all the items containing more than one atomic item have been checked) (line 2), all the large composite items have been found. The result is stored in set LCI which is the union of all the set of the large items found at each step (line 19).

For example, in the patient database shown in Figure 1(a), assume that:

(a) the minimum support is 4, and

(b) the set used to generate the composite items is $\{A, B, C, D\}$.

The candidate set of step 1 is $CC_4 = \{A \vee B \vee C \vee D\}$ (line 1). It can be seen that the support of $A \vee B \vee C \vee D$ is 4. Thus, $LC_4 = \{A \vee B \vee C \vee D\}$. The 3-subitems of $A \vee B \vee C \vee D$ are used to form CC_3 (line 12). Hence,
$CC_3 = \{A \vee B \vee C, A \vee B \vee D, A \vee C \vee D, B \vee C \vee D\}$

In *complete*, $A \vee B \vee C$ is extended with D to form a new item $A \vee B \vee C \vee D$. As $A \vee B \vee C \vee D$ is in LC_4, $A \vee B \vee C$ remains in CC_3.
For the same reason, all the other items in CC_3 also remain in CC_3.

At step 2, LC_3 is $\{A \vee B \vee D, B \vee C \vee D\}$. CC_2 is formed by the 2-subitems of $A \vee B \vee D$ and $B \vee C \vee D$ (line 12). Thus,

$$CC_2 = \{A \vee B, A \vee D, B \vee C, B \vee D\}$$

In *complete*, $A \vee B$ is extended with C and D respectively. Thus, for $A \vee B$ in *complete*, $S = \{A \vee B \vee C, A \vee B \vee D\}$. As $A \vee B \vee C$ is not in LC_3, $A \vee B \vee C$ is not large. As a result, $A \vee B$ cannot be large. Thus, $A \vee B$ is removed from CC_2.

Similarly, $A \vee D$ and $B \vee C$ are also removed from CC_2. Hence, after checking against the items in LC_3, CC_2 is $\{B \vee D\}$.

At step 3, LC_2 is $\{B \vee D\}$. As all the composite items have been checked, the algorithm terminates. Thus,

$$LCI = LC_4 \cup LC_3 \cup LC_2 = \{A \vee B \vee C \vee D, A \vee B \vee D, B \vee C \vee D, B \vee D\}.$$

3.3 Finding Large Itemsets

Once all the large items are found, a procedure based on the algorithm in [3] is used to find large itemsets. Modifications to [3] have been made to accommodate composite items. In the algorithm here each large composite item is treated as an independent item like the large atomic items.

The algorithm finds the large itemsets in several steps. At each step, a candidate set is formed first. The set contains the itemsets which might be large. The candidate set is generated according to the large itemsets found at the pervious step. After the candidate set is formed, the database is scanned to find the large itemsets. Modifications to the procedure in [3] have been made to reduce the amount of computation required.

In the following, A is the set of all the atomic items; L_k is the set of large itemsets obtained at step k of the algorithm; C_k is the set of candidate large itemsets; and C_k is generated in procedure *candidate_gen*.

```
20. $L_1 = \{\{a\} \mid a \in A$
            **and** $a$ is a large atomic item$\}$
        $\cup \{\{ca\} \mid ca \in LCI\}$
21. **for** $(k = 2; L_{k-1} \neq \emptyset; k++)$ **do**
22.     $C_k =$ *candidate_gen*$(L_{k-1})$
23.     **for each** transaction $t$ in database **do**
24.         $CC_t = \emptyset$
25.         **for each** itemset $c \in C_k$ **do**
26.             **if** $t$ contains all the items in $c$ **then**
27.                 $CC_t = CC_t \cup \{c\}$
28.             **fi**
29.         **end_for_each**
30.         **for each** itemset $c \in CC_t$ **do**
                $c.count{+}{+}$
            **end_for_each**
31.     **end_for_each**
32.     $L_k = \{c \in C_k \mid c.count \geq minimum\ support\}$
33. **end_for**
34. answer= $\bigcup_k L_k$
```

Initially the large itemsets are formed using the large items (line 20). The database is scanned in lines 23-31. Each transaction in the database is checked to see whether it supports the candidates in C_k. Set CC_t contains all the itemsets being supported by transaction t (lines 25-29). When scanning is completed, the itemsets whose support exceed the minimum support become the large itemsets (line 32). These large itemsets are used to generate the candidate large itemset for the next step (line 22). At the end, the large itemsets found at different stages are joined together (line 34).

```
**Procedure** candidate_gen$(L_{k-1})$
35. $S = \{\{s_1, ..., s_i\} \mid (s_j \in L_{k-1}$ *where* $1 \leq j \leq i)$
    **and**
    $(\forall s_m, s_n : 1 \leq m < n \leq i.s_m$ *and* $s_n$ *are different
     in two items and the two different items do not
     have common atomic items*$)$
    **and**
    $(\sim \exists e \in S.\{s_1, ..., s_i\} \subset e)\}$
36. $S' = \{\{s_1, ..., s_k\} \mid \{s_1, ..., s_k\} \in S$
         **such that**
         $\mid B \mid = k$ **where** $B = s_1 \cup s_2 \cup ... \cup s_k\}$
37. $C_k = \{\{a_1, ..., a_k\} \mid \forall a_i : 1 \leq i \leq k.a_i \in B$
         **where** $B = s_1 \cup s_2 \cup ... \cup s_k$
                 **such that** $\{s_1, ..., s_k\} \in S'\}$
```

Procedure *candidate_gen* calculates the large itemset candidates according to the large itemsets obtained at the previous step. Each candidate in set C_k is a k-itemset. If a k-itemset, say S, is large, then each of the *(k-1)*-subset of S should also be large [3]. If one of the *(k-1)*-subset of an k-itemset is not large, than the k-itemset is not large [3]. In *candidate_gen*, the large *(k-1)*-itemsets obtained at the previous step are checked to see whether they are the *(k-1)*-subsets of some k-itemsets. A k-itemset becomes a candidate if all its *(k-1)*-subsets are large.

In *candidate_gen*, firstly several sets of itemsets are formed (line 35). Each set will be checked to see whether it contains all the *(k-1)*-subsets of a k-itemset. The second conjunct in the condition on line 35 means all the itemsets in $\{s_1, ..., s_i\}$ are the *(k-1)*-subsets of a k-itemset. This is because each pair of the *(k-1)*-subsets of a k-itemset are different in two items. In addition, according to the definition in §2, each atomic item should appear in an itemset only once (e.g. $\{A \vee B, A \vee C\}$ is not a valid itemset, as A appears twice). Thus, two different items should not have common atomic items. Condition "$\sim \exists e \in S.\{s_1, ..., s_i\} \subset e$" in line 35 means the subset of any element in S should not be in S (e.g. if $\{\{A,B\},\{A,C\},\{B,C\}\}$ is in S, then $\{\{A,B\},\{A,C\}\}$ should not be in S).

Each k-itemset has k distinct *(k-1)*-subsets. Hence, in S only the sets which contain exactly k itemsets need to be considered. These sets are used to form S' (line 36). If a set in S', say *ss*, with k elements (itemsets) has k distinct items (i.e. $\mid B \mid = k$ in line 36), then *ss* must contain all the *(k-1)*-subsets of a k-itemset. As the itemsets in *ss* are from L_{k-1}, all the itemsets in *ss* are large. As a result, the k-itemset becomes a candidate (line 37).

Here is an example showing how *candidate_gen* works. Assume that
$L_2 = \{\{A,B\},\{A,B \vee C\},\{A,D\},\{B \vee C,D\}\}$.
From line 35,
$S = \{\{\{A,B\},\{A,D\}\},\{\{A,B \vee C\},\{A,D\},\{B \vee C,D\}\}\}$.

- $\{\{A,B\},\{A,B \vee C\}\}$ is not in S, as B and $B \vee C$ have B in common.
- $\{\{A,D\},\{B \vee C,D\}\}$ is not in S, as $\{\{A,D\}, \{B \vee C,D\}\} \subset \{\{A,B \vee C\},\{A,D\},\{B \vee C,D\}\}$ holds.

Each candidate in C_3 is a 3-itemset, i.e. each candidate has three items. $\{\{A,B\},\{A,D\}\}$ only has two elements. Thus, $\{\{A,B\},\{A,D\}\}$ cannot contain all the *2*-subsets of any *3*-itemset.
As a result, $\{\{A,B\},\{A,D\}\}$ will not be considered further.
As only $\{\{A,B \vee C\},\{A,D\},\{B \vee C,D\}\}$ has three elements and contains three distinct items (i.e. $A, B \vee C$ and D), according to line 36,

$$S' = \{\{\{A,B \vee C\},\{A,D\},\{B \vee C,D\}\}\}$$

Hence, from line 37, the candidate in C_3 is $\{A, B \vee C, D\}$.

The procedure generating the candidates in [3] consists of two phases. First, in the join phase, the large itemsets generated at the *k-1* step are "joined" together to form k-itemsets. Then, in the prune phase, the *(k-1)*-subsets of all the k-itemsets generated at the join phase are calculated. If a *(k-1)*-subset of a k-itemset is not one of the large itemsets obtained at the previous step, then the k-itemset is removed from the candidate set.

In contrast, procedure *candidate_gen* in this paper only adds a k-itemset to the candidate set if all the *(k-1)*-subsets of the k-itemset are large. This avoids generating any k-itemset which will be deleted later. Also, it does not need to calculate the *(k-1)*-subsets of any k-itemset. Thus, the scheme in this paper requires less computation than the one in [3].

4 Conclusions

The algorithm in this paper allows large itemsets to contain composite items. This enables certain useful rules to be discovered in some applications.

Other work has presented algorithms for mining association rules in the presence of taxonomies over the items [5, 6]. In those algorithms, a transaction is regarded as supporting the upper level of a hierarchy if the transaction supports at least one item which is at a lower level in the hierarchy. The algorithms require the users to provide taxonomies. If it is intended to investigate the possible relationships between some composite items, e.g. $A \vee B, A \vee C$ and $B \vee C$, and some atomic items, then either several taxonomies have to be provided or each pair of A, B and C has to be placed under a hierarchy in a taxonomy.

In contrast, the algorithm in this paper does not require the users to provide a taxonomy. The users need only provide the set of atomic items in which they are interested. From this set the composite items will be generated by the algorithm automatically. Thus, the algorithm in this paper is both easier to use and more flexible.

References

[1] R. Agrawal, T. Imielinski and A. Swami, Mining association rules between sets of items in large databases, *Proc. of ACM SIGMOD Conference*, pp. 207-216, 1993

[2] R. Agrawal and J. Shafer, Parallel mining of association rules, *IBM Research Report RJ10004*, 1996

[3] R. Agrawal and R. Srikant, Fast algorithms for mining associations rules, *Proc. of 10th Int. Conference on VLDB*, 1994

[4] R. Agrawal and R. Srikant, Mining sequential patterns, *Proc. of 11th Int. Conference on Data Engineering*, 1995

[5] M. Holsheimer, M. Kersten, H. Mannila and H. Toivonen, A perspective on database and data mining, *Technical report CS-R9531*, CWI, The Netherlands, 1995

[6] R. Srikant and R. Agrawal, Mining generalised association rules, *Proc. of 21st Int. Conference on VLDB*, 1995