SeqARM: An Association Rule Mining Algorithm Based on Sequence Constraint

Shenshen Bai
Department of Information Engineering
Lanzhou Vocational Technical College
Lanzhou China 730070
shenshen128@sohu.com

***Abstract*—Sequence constraint mainly considers the occurrence time of item, which can determine whether an association rule is valid. This paper proposes a novel algorithm SeqARM that aims to mine strong association rule with sequence constraint and runs on the second phase of association rule mining. SeqARM employs a fine data structure, named FI-Tree, which is used to save and find frequent itemsets according to a few characteristics of association rule. This work can dramatically reduce the number of invalid association rules, and speed up the procedure of association rules. At last, the experiments prove that SeqARM can improve the performance and effect of the association rule mining.**

Keywords-association rule; sequence constraint; SeqARM; FI-Tree

I. INTRODUCTION

Association rule mining is to find the correlation between items and play an important role in data mining. The procedure of mining association rule includes two phases [5]. The first one is to find all frequent itemsets with a minimum support threshold. And the second one is to generate strong association rules from the frequent itemsets with a minimum confidence threshold. Since [1] proposed this problem, many researchers addressed a lot of works on this issue [2, 8, 6, 9, 3]. However, many presented works focused on the first phase, few authors focused their attentions on the second phase.

In another way, these works did not adequately take the demands of user and the constraints of application into account. The results may not have true values and real meanings. To solve the drawbacks, some researchers proposed constraint-based association rule algorithms [7, 4]. These constraints are directly dependent on the value of items and are easily applied to the process of frequent itemsets mining. However, in real-world applications, there is another constraint which isn't related with the value of item, e.g. the occurrence time of item. Considering the grades of a undergraduate student, obviously the grades of forth semester may not affect the grades of third semester. Let *X* = {*C Programming = 'A', Data Structure = 'B', Database = 'B'*} be a frequent itemset. We know that rule *R1* = {*Database = 'B'* => (*C Programming = 'A'* ∧ *Data Structure = 'B'*)} is an illegal association rule, because *C Programming* and *Data Structure* are prerequisite courses of Database. However, rule *R2* = {(*C Programming = 'A'* ∧ *Data Structure = 'B'*) => *Database = 'B'*} is a valid association rule. This constraint cannot be applied to the first phase of association rule mining. We call this kind of constraint as *sequence constraint*. This paper focuses on sequence constraint and the second phase of association rule mining, proposes a novel strong association rule mining algorithm with sequence constraint, named *SeqARM*.

The rest of the paper is organized as follows. In section II, the definitions and properties of association rule are presented. Section III proposes the sequence-constraint-based association rules mining algorithm and section IV experimentally shows this algorithm is efficient and effective. Section V summarizes this proposal.

II. BASIC DEFINITIONS AND PROPERTIES

The first thing of this section is giving the definitions of frequent itemset and association rule. The second one is presenting two properties of association rule.

Definition 1: Let $I = \{I_1, I_2, \cdots, I_n\}$ be a set of items, and $D = \{T_1, T_2, \cdots, T_m\}$ be a transaction database, $T_i = \{tid_i, X_i\}$ be a transaction, where tid_i is an identifier and $X_i \subseteq I$ is an itemset. The *support* of an itemset X_i is the number of transactions containing X_i in D, denoted by *sup(X)*. Given ε is the minimum support threshold. X_i is a frequent itemset if $sup(X_i) \geq \varepsilon$). If $|X_i| = k$, X_i is called frequent k-itemset.

Definition 2: Given itemset $X_i \subset I, X_j \subset I$ and $X_i \cap X_j = \phi$, $X_i \Rightarrow X_i$ is an association rule. The *support* of $X_i \Rightarrow X_i$ is equal to the support of $X_i \cup X_j$, i.e. $sup(X_i \Rightarrow X_i) = sup(X_i \cup X_j)$. The *confidence* of $X_i \Rightarrow X_i$ is $P(X_j|X_i)$, denoted by $conf(X_i \Rightarrow X_i)$, $conf(X_i \Rightarrow X_i) = sup(X_i \cup X_j)/sup(X_i)$.

The basic idea of generating strong association rules from frequent itemset is also "Generation-and-Test" [5]. Given a frequent itemset, the method is (1) firstly enumerating all candidate association rules, (2) then calculating the confidence of each candidate and comparing the confidence with the threshold. However, this process is inefficient, especially for long frequent itemset. Let *L*={*A, B, C*} be a frequent 3-itemset, the number of candidate association rules of *L* is 6, they are *A=>BC, B=>AC, C=>AB, AB=>C, AC=>B, BC=>A*. For a frequent 4-itemset, the number of candidates is 14, and the number of candidates of a frequent n-itemset is $C_n^1 + C_n^2 + \cdots + C_n^{n-1} = 2^n - 2$. It can be seen that the second phase also play an important role. To improve the performance, the efficient ways are pruning and constraint.

Following are 2 basic properties of association rule, which can be used to effectively prune the candidates that are not strong association rules.

Property 1: Given a frequent k-itemset $L_k = \{I_1, I_2, \cdots, I_k\}$, $S \subset L_k$, $S' \subset S$, if $S \Rightarrow L_k - S$ is not a strong association rule, then $S' \Rightarrow L_k - S'$ is not a strong association rule.

Proof: Let λ be the minimum confidence threshold.

$\because S \Rightarrow L_k - S$ is not a strong association rule,

$\therefore conf(S \Rightarrow L_k - S) < \lambda$, i.e. $\frac{sup(L_k)}{sup(S)} < \lambda$.

$\because S' \subset S$,

$\therefore sup(S) \leq sup\ (S')$,

$\therefore \frac{sup(L_k)}{sup(S')} \leq \frac{sup(L_k)}{sup(S)}$,

$\therefore conf(S' \Rightarrow L_k - S') < \lambda$,

$\therefore S' \Rightarrow L_k - S'$ is not a strong association rule. □

Property 2: Given a frequent k-itemset $L_k = \{I_1, I_2, \cdots, I_k\}$, $S \subset L_k$, $L \supset L_k$, if $S \Rightarrow L_k - S$ is not a strong association rule, then $S \Rightarrow L - S$ is not a strong association rule.

Proof: Let λ be the minimum confidence threshold.

$\because S \Rightarrow L_k - S$ is not a strong association rule,

$\therefore conf(S \Rightarrow L_k - S) < \lambda$, i.e. $\frac{sup(L_k)}{sup(S)} < \lambda$.

$\because L \supset L_k$, ,

$\therefore sup(L) \leq sup\ (L_k)$,

$\therefore \frac{sup(L)}{sup(S)} \leq \frac{sup(L_k)}{sup(S)}$,

$\therefore conf(S \Rightarrow L - S) < \lambda$,

$\therefore S \Rightarrow L - S$ is not a strong association rule. □

Applying these two properties into the process of generating the strong association rule can dramatically reduce the number of candidates and improve the speed.

III. Mining Association Rule with Sequence Constraint

This section presents the novel sequence-constraint-based association rule mining method. Section III-A proposes the related definitions and section III-B gives a new data structure used to store frequent itemsets. The algorithm of SeqARM is shown in section III-C.

A. *Sequence Constraint*

This section presents the formally definitions of sequence constraint.

Definition 3: Let I_i, I_j be two items, notation $I_i \succ I_j$ represents that I_i's occurrence is later than I_j's.

Definition 4: Let S_i, S_j be two itemsets, if $\exists I_i \in S_i$, $\exists I_j \in S_j$, and $I_i \succ I_j$, we denote the relationship of S_i and S_j as $S_i \succ S_j$.

Definition 5: Given itemsets S_i and S_j satisfy $S_i \succ S_j$, if S_i is in the left of an association rule, S_j cannot be in the right. We name this constraint as *sequence constraint*.

B. *Data Structure*

The major operations of mining strong association rule are generating and searching subsets of frequent itemset. In order to improve the speed of these operations, this work designs an new fine data structure, named *FI-Tree* (Frequent Itemset Tree), which combines the features of *FP-Tree* [6] and *AFP-Tree* [3]. Next, we will describe FI-Tree through an example.

tid	items
100	I1, I2, I5
200	I2, I4
300	I2, I3
400	I1, I2, I4
500	I1, I3
600	I2, I3
700	I1, I3
800	I1, I2, I3, I5
900	I1, I2, I3

Table I
Transaction Database

Table I is a transaction database, which includes 9 transactions and 5 different items [5]. Set the minimum support threshold is 2, mining this database can get 13 frequent itemsets which are shown in table II. The numeric following the ":" is the support.

The algorithm of building FI-Tree is presented in algorithm 1. The input parameter *FIs* is a set of frequent itemsets, which is in ascending order according to the length of each frequent itemset, if two frequent itemset have the same length; the order is descending according to their supports. And each item in a frequent itemset is descending order according its support. These orders can be seen from table II. Figure 1 depicts the FI-Tree of table II.

id	frequent itemset
1	{ I2: 7 }
2	{ I1: 6 }
3	{ I3: 6 }
4	{ I4: 2 }
5	{ I5: 2 }
6	{ I2, I1: 4 }
7	{ I2, I3: 4 }
8	{ I2, I4: 2 }
9	{ I2, I5: 2 }
10	{ I1, I3: 4 }
11	{ I1, I5: 2 }
12	{ I2, I1, I3: 2 }
13	{ I2, I1, I5: 2 }

Table II
Frequent Itemsets

In FI-Tree, a path $P = \{N_1, N_2, \cdots, N_k\}$, where N_{i-1} is the parent of N_i, $\forall 1 < i \leq k$, the parent of N_1 is root, represents a frequent k-itemset whose support is the count of N_k. It is very convenient to get the support of any frequent itemset from FI-Tree by preorder traversal. For example, if we want to search the support of {I_1, I_3}, so long as we traverse the shadow node in figure 1, we can get its support, it is 4.

```
Algorithm 1 Build_FI-Tree($FIs$)
 1: $root$ = creat_node(null);
 2: for each $L \in FIs$ do
 3:   $T = root$;
 4:   for $i = 0$ to $L.size$ do
 5:     for each $N$ is a child of $T$ do
 6:       if $N$.item = $L[i]$.item then
 7:         $T = N$;
 8:         break;
 9:       end if
10:       $N$ = create_node($L[i]$.item); // new node
11:       $N$.count = $L$.count;
12:       make a pointer from $T$ to $N$;
13:     end for
14:   end for
15: end for
```

Furthermore, we need an array to store the additive sequence of each item. This constraint doesn't come from the frequent itemset, but come from other external file.

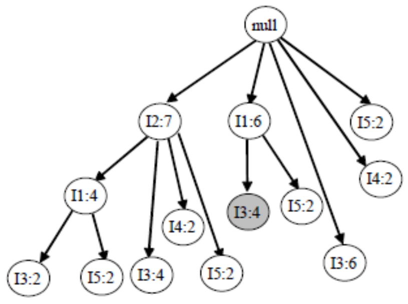


Figure 1. FI-Tree

C. Algorithm

This section presents the detail of the SeqARM algorithm. Algorithm 2 is the pseudocode.

At the beginning, SeqARM initialize R with ϕ (line 1), which stores the strong association rules. Next, it treats each child of root as a frequent 1-itemset, then calls the iteration function *generate_rule*() which is the core of SeqARM (lines 2~4). Firstly, *generate_rule*() visits FI-Tree by preorder traversal. Line 7 adds child node N of current subtree into array L. Now L is frequent (k+1)-itemset. Line 9 gets all proper subsets of L except DS that saves the deleted itemsets before. The purpose of this step is pruning. Let L^i represents L of ith iteration of *generate_rule*(). According to line 19, $\forall s \in DS, s \Rightarrow L^{i-1} - s$ is not a strong association rule. If $|L^{i-1}| < |L^i|, L^{i-1} \subset L^i$, according property 2, $s \Rightarrow L^i - s$ is not a strong association rule. Lines 11~21 determine whether $ss \Rightarrow L - ss$ is a strong association rule with sequence constraint where $ss \in S$. If $ss \succ L - ss$ is true, ss is skipped (lines 12, 13). Otherwise, the confidence of $ss \Rightarrow L - ss$ is calculated (line 15). If $ss \Rightarrow L - ss$ is a strong association rule, it is appended into R (lines 16, 17). Otherwise, according to property 1, line 19 deletes ss and its subsets from S and simultaneously appends them into DS. If N is not leaf node, *generate_rule*() is called again (lines 22~24).

```
Algorithm 2 SeqARM($T, \lambda$)
Parameter: $T$: root of FI-Tree, $\lambda$: minimum confidence
 1: $R = \emptyset$;
 2: for each child $N$ of $T$ do
 3:   $L[0].item = N.item$, $L[0].count = N.count$;
      //$L$ is an array, stores a frequent itemset
 4:   generate_rule($\lambda, L, N, 1, R, \emptyset$);
 5: end for
Procedure generate_rule($\lambda, L, T, k, R, \bar{S}$);
Parameter: $\lambda$: minimum confidence, $L$: frequent itemset,
    $T$: root of subtree, $k$: length of $L$, $R$: set of strong as-
    sociation rules, $\bar{S}$ saves the itemsets deleted in previous
    iteration.
 6: for each child $N$ of $T$ do
 7:   $L[k].item = N.item$, $L[k].count = N.count$;
 8:   $DS = \bar{S}$;
 9:   $S$ = subset($L$) $\backslash DS$; // $S$ stores all proper subsets of
      $L$ except $DS$, property 2.
10:   sort($S$); // descending order according to length
11:   for each $ss \in S$ do
12:     if $ss \succ L - ss$ then
13:       continue; // sequence constraint
14:     end if
15:     $c$ = conf($ss \Rightarrow L - ss$); // $c = L.count \div sup(ss)$
16:     if $c \geq \lambda$ then
17:       $R = R \cup (ss \Rightarrow L - ss)$; // strong association
          rule
18:     else
19:       $S$ = remove_subset($S, ss, DS$);
          // remove $ss$ and its subset from $S$ and add them
          into $DS$, property 1
20:     end if
21:   end for
22:   if $N$ is not leaf node then
23:     generate_rule($\lambda, L, N, k + 1, R, DS$)
24:   end if
25: end for
```

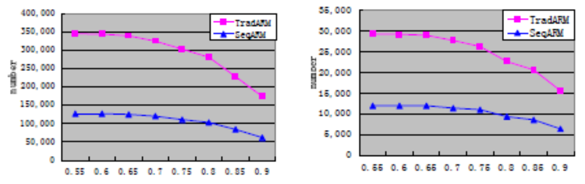
IV. Experimental Evaluation

This section experimentally evaluates the effect and performance of SeqARM. The evaluated criteria are running time and the number of association rule. The comparison is traditional association rule method, named as "TradARM". The experiments are conducted on an Intel Core T2080 1.73GHz CPU with 1G Bytes of main memory, running on Windows XP professional SP2. All codes are implemented in C# using Microsoft Visual C# 2005 Express Edition. We use two real datasets, pumsb* and mushroom, which are standard test datasets and can be downloaded from http://fimi.cs.helsinki.fi/.

In the experiments we define the sequence between items as following:

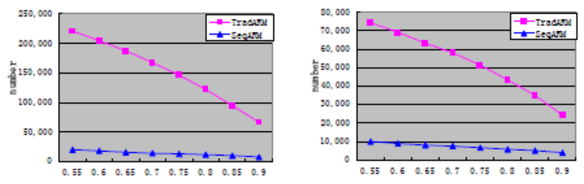
Given I_i, I_j are two items, if $I_i > I_j$, it is considered I_i's occurrence is later than I_j's, i.e. $I_i \succ I_j$.

Figure 2 and 3 show the number of association rule generated by two algorithms. The number of association rule generated by SeqARM is 1/2 to 1/3 of the number generated by TradARM on pumsb*. The number generated by SeqARM is an order of magnitude smaller than the number generated by TradARM on mushroom. From this view, SeqARM can obviously reduce the number of association rule which is invalid, the result is more meaningful.



(a) minimum support: 0.45 (b) minimum support: 0.5

Figure 2. Number of association rule in pumsb*



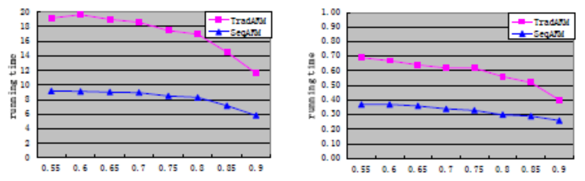
(a) minimum support: 0.25 (b) minimum support: 0.3

Figure 3. Number of association rule in mushroom

Figure 4 and 5 give the running time of two algorithms. These two figures demonstrate that SeqARM is faster than TradARM. In figure 4 the time of SeqARM is about half of TradARM's and in figure 5 the time of SeqARM is far less than TradARM. It can be seen from this point that SeqARM is not only able to avoid generating invalid association rule, but also improves the performance of mining association rule.

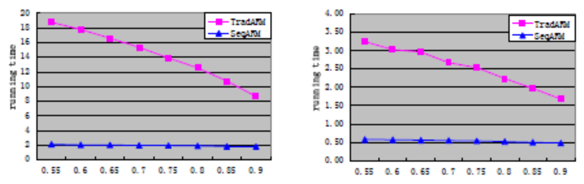
V. Conclusion

In this paper, we propose a new kind of constraint of association rule, called sequence constraint. To mine association rule with sequence constraint, a novel algorithm SeqARM is presented. SeqARM uses FI-Tree to store frequent itemsets. Experiments show that SeqARM outperform the traditional method and supply accurate association rules.



(a) minimum support: 0.45 (b) minimum support: 0.5

Figure 4. Running time on pumsb*



(a) minimum support: 0.25 (b) minimum support: 0.3

Figure 5. Running time on mushroom

References

[1] R. Agrawal, T. Imieli´nski, and A. Swami. Mining association rules between sets of items in large databases. In Proceedings of the 1993 ACM SIGMOD international conference on Management of data, SIGMOD '93, pages 207–216, New York, NY, USA, 1993. ACM.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[3] X. Chen, L. Li, Z. Ma, S. Bai, and F. Guo. F-miner: A new frequent itemsets mining algorithm. In ICEBE, pages 466–472, 2006.

[4] L. Cui, S. Yuan, and C. Zhao. Algorithms for mining constrained association rules. Chinese Journal of Computers, 23(02):216–220, 2000.

[5] J. Han and M. Kamber. Data Mining: Concepts and Techniques, Second Edition (The Morgan Kaufmann Series in Data Management Systems). Morgan Kaufmann, 2 edition, Jan. 2006.

[6] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In SIGMOD Conference, pages 1–12, 2000.

[7] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In KDD, pages 67–73, 1997.

[8] M. J. Zaki. Scalable algorithms for association mining. IEEE Transactions on Knowledge and Data Engineering, 12:372–390, 2000.

[9] M. J. Zaki and K. Gouda. Fast vertical mining using diffsets. In KDD, pages 326–335, 2003.