Expert Systems with Applications 30 (2006) 650–657

Expert Systems with Applications

www.elsevier.com/locate/eswa

An efficient data mining approach for discovering interesting knowledge from customer transactions

Show-Jane Yen*, Yue-Shi Lee

Department of Computer Science and Information Engineering, Ming Chuan University, Taipei, Taiwan, ROC

Abstract

Mining association rules and *mining sequential patterns* both are to discover customer purchasing behaviors from a transaction database, such that the quality of business decision can be improved. However, the size of the transaction database can be very large. It is very time consuming to find all the association rules and sequential patterns from a large database, and users may be only interested in some information.

Moreover, the criteria of the discovered association rules and sequential patterns for the user requirements may not be the same. Many uninteresting information for the user requirements can be generated when traditional mining methods are applied. Hence, a data mining language needs to be provided such that users can query only interesting knowledge to them from a large database of customer transactions. In this paper, a data mining language is presented. From the data mining language, users can specify the interested items and the criteria of the association rules or sequential patterns to be discovered. Also, the efficient data mining techniques are proposed to extract the association rules and the sequential patterns according to the user requirements.
© 2005 Elsevier Ltd. All rights reserved.

Keywords: Data mining; Association rule; Sequential pattern; Interesting knowledge; Transaction database

1. Introduction

An association rule (Han and Pei, 2000) describes the association among items in which when some items are purchased in a transaction, others are purchased too. An association rule has the form X⇒Y, in which X and Y are two sets of items. In this paper, we refer to X as an antecedent and Y as a consequent of this rule. The *length* of an itemset *i* is the number of items in the itemset *i*, and an itemset of length *k* is called a *k-itemset*. A transaction *t* *supports* an itemset *i* if *i* is contained in *t*. The *support* for an itemset *i* is defined as the ratio of the number of transactions that supports the itemset *i* to the total number of transactions. If the support for an itemset *i* satisfies the user-specified *minimum support* threshold, then *i* is called *frequent itemset*, and a frequent itemset of length *k* a *frequent k-itemset*. The *confidence* of a rule X⇒Y is defined as the ratio of the support for the itemsets X∪Y to the support for the itemset X. If itemset Z=X∪Y is a frequent itemset and the confidence of X⇒Y is no less than the user-specified *minimum confidence*, then the rule X⇒Y is an association rule.

Mining sequential patterns (Pie et al., 2001) is to find the sequential purchasing behavior for most customers from a large transaction database. A sequence is an ordered list of the itemsets$\langle s_1, s_2, \ldots, s_n\rangle$, where s_i is a set of items. A *customer sequence* is the list of all the transactions of a customer, which is ordered by increasing transaction-time. A customer sequence *c* *supports* a sequence *s* if *s* is contained in *c*. The *support* for a sequence *s* is defined as the ratio of the number of customer sequences that supports *s* to the total number of customer sequences. If the support for a sequence *s* satisfies the user-specified *minimum support* threshold, then *s* is called *frequent sequence*. The *length* of a sequence *s* is the number of itemsets in the sequence. A sequence of length *k* is called a *k-sequence*, and a frequent sequence of length *k* a *frequent k-sequence*. A *sequential pattern* is a frequent sequence that is not contained in any other frequent sequence.

In this paper, we present a data mining language, from which users only need to specify the criteria and the interested items for discovering the association rules and

* Corresponding author. Tel.: +886 33507001; fax: +886 33593874.
E-mail address: sjyen@mcu.edu.tw (S.-J. Yen).

0957-4174/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2005.07.035

sequential patterns. We also propose efficient data mining algorithms for the data mining language processing. For the data mining algorithms, we focus on discovering the associations among interested items and all the other items. For our data mining system, a user can make a query through our query language, and the system answers to the query according to user specified items and criteria immediately. If the answers do not satisfy user's needs, then user can resubmit his/her query by adjusting the criteria and item constraints.

Many constraint-based mining methods have been proposed. Hipp and Guntzer (2002) presented that data mining process should be an initial unconstrained and costly mining run. The mining queries are answered from the initial mining result such that response time can be minimized. However, the discovered association rules may become invalid or inappropriate since the transactions are increasing any time. It is very costly to re-run the unconstrained mining algorithm to obtain the up-to-date initial mining result. Ng, Lakshmanan, Han, and Mah (1999) considered aggregate constraints and item constraints for mining association rules. For item constraints, the items in the discovered frequent itemset must exactly be contained in the specified items. Pei and Han (2000, 2002) developed pattern-growth methods for constrained frequent pattern mining and sequential pattern mining. An item constraint specifies what is the particular individual or group of items that should or should not be presented in the pattern, that is, the items in the discovered patterns have to be contained in the specified itemset. In (Pei et al., 2002), they discussed about mining sequential patterns with regular expression, the items in the discovered patterns must appear in the sequence defined in the regular expression. All the above approaches cannot discover the associations among certain items and all the other items. Hence, the item constraints in the above approaches are different from our work.

Meo, Psaila and Ceri (1996) proposed a SQL-like operator for extracting association rules. However, SQL-like operator cannot completely express the associations among certain items and all the other items. Furthermore, the SQL-like operator performs set-oriented operations (i.e. join operations), which are very inefficient operations. Yen and Chen (1997) proposed a data mining language for mining interesting association rules. They presented a user-friendly mining language and users can specify the interested items and the criteria of the rules to be discovered. This approach constructs an association graph and generates all the frequent itemsets by traveling the association graph. However, it needs to take a lot of memory space to record the related information. In this paper, we successfully integrate two kinds of patterns and use the similar style of the data mining language proposed in (Yen and Chen, 1997). Besides, we also propose efficient data mining algorithms to find all the associations among certain items and all the other items.

2. Data mining language and database transformation

The data mining language is defined as follows. Users can query association rules or sequential patterns by specifying the related parameters in the data mining language.

Mining ⟨Data Mining Technology⟩
From ⟨CSD⟩
With ⟨(D_1),(D_2), …,(D_m)⟩
Support ⟨s%⟩
Confidence ⟨c%⟩

In the **Mining** clause, ⟨Data Mining Technology⟩ can be ⟨association rules⟩ or ⟨sequential patterns⟩. The former is to discover association rules and the later is to discover sequential patterns.

In the **From** clause, ⟨CSD⟩ is used to specify the database name to which users query the association rules or sequential patterns.

In the **With** clause, if the ⟨Data Mining Technology⟩ is ⟨sequential patterns⟩, ⟨(D_1),(D_2), …,(D_m)⟩ are user-specified itemsets which ordered by increasing purchasing time, and (D_i) can be the notation '*' which represents any sequences. If the ⟨Data Mining Technology⟩ is ⟨association rules⟩, then m is equal to 2, and D_1 and D_2 are the itemsets in the antecedent and consequent, respectively, of the discovered rules. Besides, (D_i) and the items in D_i can be the notation '*' which represents any items.

Support clause is followed by the user-specified minimum support *s*%.

Confidence clause is followed by the user-specified minimum confidence c% if the ⟨Data Mining Technology⟩ is ⟨association rules⟩. If the ⟨Data Mining Technology⟩ is ⟨sequential patterns⟩, this clause is ignored.

In order to find the interesting association rules and sequential patterns efficiently, we need to transform the original transaction data into another type. Each item in each customer sequence is transformed into a bit string. The length of a bit string is the number of the transactions in the customer sequence. If the *i*th transaction of the customer sequence contains an item, then the *i*th bit in the bit string for this item is set to 1. Otherwise, the *i*th bit is set to 0. For example, in Table 1, the bit string for item A in CID 1 is 011. Hence, we can transform the customer sequence database (Table 1) into the *bit-string database* (Table 2).

From the bit-string database, we can easily compute the number of the transactions in a customer sequence, which contain an itemset. For example, in Table 1, if we want to know how many transactions in CID 1 support the itemset (A,C,E). We can perform logical AND operations on the bit strings for items A, C and E in CID 1. The number of 1's in the resultant bit string is the number of the transactions which contain the itemset (A,C,E) in CID 1.

Table 1
Customer sequence database (CSD)

CID	Customer sequence
1	⟨(C)(A,C)(A,C,E)⟩
2	⟨(A,E)(A)(A,C,E)(C,E)⟩
3	⟨(C)(E)(E)(C,E)⟩
4	⟨(B,D)(A,E)(B,C)(A,E)(A,B,E)(F)⟩
5	⟨(D)(D,E,F)(C,E,F)(A,D)(B,D)(D,F)⟩

Suppose a customer sequence contains the two sequences S_1 and S_2. We present an operation called *sequential bit-string operation* to check if the sequence S_1S_2 is also contained in this customer sequence. The process of the sequential bit-string operation is described as follows: Let the bit string for sequence S_1 in customer sequence c is B_1, and for sequence S_2 is B_2. Bit string B_1 is scanned from left to right until a bit value 1 is visited. We set this bit and all bits on the left hand side of this bit to 0 and set all bits on the right hand side of this bit to 1, and assign the resultant bit string to a template T_b. Then, the bit string for sequence S_1S_2 in c can be obtained by performing logical AND operation on bit strings T_b and B_2. If the number of 1's in the bit string for sequence S_1S_2 is not zero, then S_1S_2 is contained in customer sequence c. Otherwise, the customer sequence c does not contain S_1S_2.

For example, consider Table 1. We want to check if sequence ⟨(A)(C)⟩ is contained in customer sequence in CID 1. From Table 2, we can see that the bit string for items A and C in CID 1 are $B_A=011$ and $B_C=111$, respectively, and the template bit string $T_b=001$. By performing logical AND operation on T_b and B_C, we can obtain that the bit string for sequence ⟨(A)(C)⟩ in customer sequence CID 1 is 001.

3. Mining interesting association rules

In this section, we focus on mining interesting association rules according to proposed data mining language. We divide the query into two cases. Case 1: there are items in the antecedent of the discovered rules specified. Case 2: there are items in the consequent of the discovered rules specified, but the item in the antecedent is not specified, which can be any items. Suppose that the itemset specified in the antecedent of the discovered rules in Case 1 is X, and the itemset specified in the consequent of the discovered rules is Y. We propose an efficient algorithm called *MIAR* (Mining Interesting Association Rules) to find all the interesting association rules according to the user requirements, which is described as follows:

Table 2
Bit-string database

CID	Transaction items	Bit string for each item
1	A, C, E	011,111,001
2	A, C, E	1110,0011,1011
3	C, E	1001,0111
4	A, B, C, D, E, F	010110,101010,001000,100000, 010110,000001
5	A, B, C, D, E, F	000100,000010,001000,110111, 011000,011001

Step 1. Scan the bit-string database once to compute the support for the specified itemset, and then find all the frequent 1-itemsets.

For the record in CID i in the bit-string database, if each item in itemset X or X∪Y (or Y) is contained in this record, then perform the logical AND operations on the bit strings for the items in itemset X or X∪Y (or Y). The number m_i of 1's in the resultant bit string is the number of the transactions which contain the itemset X or X∪Y (or Y) for CID i. If there is an item in itemset X or X∪Y (or Y) not contained in CID i, then m_i is equal to 0. For each item j, if item j is contained in the record in CID i, then find the bit string for item j to count the number C_{ij} of the transactions which contain item j for CID i. Otherwise, the value C_{ij} is 0.

Suppose there are p customers and q transactions in the customer sequence database. The number of the transactions that contain the itemset X or X∪Y (or Y) is $m=\sum_{i=1}^{p} m_i$, and the support for the itemset X (or Y) is m/q. If the support is no less than the user-specified minimum support, then compute the support for each item j by the expression $\left(\sum_{i=1}^{p} C_{ij}\right)/q$.

Step 2. Generate candidate (k+1)-itemsets (k is the length of itemset X (or X∪Y) for Case 1, and k is the length of itemset Y for Cases 2), scan the bit-string database to find the frequent (k+1)-itemsets, which contain the itemset X (or X∪Y) for Case 1, the itemset Y for Case 2, and generate the (k+1)-itemset database.

The method to generate (k+1)-itemsets is as follows: For each frequent 1-itemset f, the candidate (k+1)-itemset X∪f (or X∪Y∪f) for Case 1 and Y∪f for Case 2 can be generated. For the record in CID i in the bit-string database, if the record contains the itemset X (or X∪Y) for Case 1 (the itemset Y for Case 2) and a frequent 1-itemset g, then generate the itemset X∪g (or X∪Y∪g) or Y∪g, and perform logical AND operation on the bit strings for itemset X (or X∪Y) or Y and the item g. If the resultant bit string is not zero, then output the itemset X∪g (or X∪Y∪g) or Y∪g and the resultant bit string to the (k+1)-itemset database, and accumulate the number of the transactions which contain the candidate itemset X∪g (or X∪Y∪g) or Y∪g by counting the number of 1's in the resultant bit string. After scanning each record in the bit-string database, the (k+1)-itemset database can be generated and the support for each candidate (k+1)-itemset can be obtained.

Step 3. The frequent itemsets are generated for each iteration. In the (h–k)th iteration (h≥k+1), generate candidate (h+1)-itemsets, scan the h-itemset database to generate (h+1)-itemset database and find all the frequent (h+1)-itemsets).

For every two frequent h-itemsets (a_1, a_2, …, a_{h-1}, b) and (a_1, a_2, …, a_{h-1}, c) (b>c), the candidate (h+1)-itemset (a_1, a_2, …, a_{h-1}, b, c) can be generated. For the record in

CID *i* in the h-itemset database, if the record contains two frequent h-itemsets ($a_1, a_2, \ldots, a_{h-1}$, b) and ($a_1, a_2, \ldots, a_{h-1}$, c) ($b>c$), then generate the (h+1)-itemset ($a_1, a_2, \ldots, a_{h-1}$, b, c). We can use the same method in Step 2 to generate (h+1)-itemset database and the frequent (h+1)-itemset can also be obtained.

Step 4. Generate all the association rules which satisfy the user requirements

For Case 1, if there is a specified itemset Y in the consequent in the **With** clause, then for every two sub-itemsets X_1 and Y_1 of the frequent itemset Z, where $X \subseteq X_1$, $Y \subseteq Y_1$, $X_1 \cap Y_1 = \phi$ and $X_1 \cup Y_1 = Z$, if the confidence of the rule $X_1 => Y_1$ is no less than the user-specified minimum confidence, then the rule $X_1 => Y_1$ is an association rule for user requirements. If there is no items specified in the consequent in the **With** clause, then for every two sub-itemsets X_2 and Y_2 of each frequent itemset V, where $X \subseteq X_2$, $X_2 \cap Y_2 = \phi$ and $X_2 \cup Y_2 = V$, if the confidence of the rule $X_2 => Y_2$ is no less than the user-specified minimum confidence, then the rule $X_2 => Y_2$ is an association rule for user requirements.

In Case 2, for every two sub-itemsets X_3 and Y_3 of each frequent itemset Z, where $Y \subseteq Y_3$, $X_3 \cap Y_3 = \phi$ and $X_3 \cup Y_3 = Z$, if the confidence of the rule $X_3 => Y_3$ is no less than the user-specified minimum confidence, then the rule $X_3 => Y_3$ is an association rule for user requirements.

For example, Query 1 means that the user would like to find all the association rules whose antecedent and consequent contain items A and C, respectively, from the customer sequence database CSD (Table 1). The minimum support and the minimum confidence are set to 5 and 20%, respectively.

Query 1:
Mining ⟨Association Rules⟩
From ⟨CSD⟩
With ⟨(A,*),(C,*)⟩
support ⟨5%⟩
confidence ⟨20%⟩

After performing step 1, we can find all the frequent 1-itemsets and their supports. The set of the frequent 1-itemsets are {A, B, C, D, E, F}. Because (A, C) is a frequent itemset, we go on Step 2. According to step 2, we can obtain the candidate 3-itemsets (A, B, C), (A, C, D), (A, C, E) and (A, C, F). After scanning bit-string database, the generated 3-itemset database are shown in Table 3, and the frequent 3-itemset is (A, C, E). Finally, the frequent itemsets that contain the itemset (A, C) are (A, C) and (A, C, E).

According to step 4, because there is the specified itemset (A) in the antecedent and itemset (C) in the consequent of the discovered rules in Query 1, we can find three association rules: (A)⇒(C, E), (A, E)⇒(C) and (A)⇒(C).

Table 3
3-itemset database for Query 1

CID	3-itemsets	Bit string for each 3-itemset
1	(A, C, E)	001
2	(A, C, E)	0010

4. Mining interesting sequential patterns

In this section, we describe the proposed algorithm *MISP* (Mining Interesting Sequential Patterns) for finding all the interesting sequential patterns according to the user requirements. For example, in Query 2, the user would like to find all the sequential patterns which contain the sequence ⟨(E)(A)(B)⟩ from the customer sequence database (Table 1) and the minimum support threshold is set to 40%.

Query 2:
Mining ⟨Sequential Patterns⟩
From ⟨CSD⟩
With ⟨*,(E),*,(A),*,(B),*⟩
support ⟨40%⟩

Suppose the user specifies a sequence which contains m itemsets D_1, D_2, …and D_m in the **With** clause and S=⟨(D_1)(D_2)…(D_m)⟩. We divide the algorithm MISP for this type of queries into two steps: the first step is to find (m+1)-frequent sequences which contains sequence S, and the second step is to find all the *q*-frequent sequences ($q \geq m+2$) which contains sequence S. In the following, we describe the two steps:

Step 1. Find all the frequent (m+1)-sequences

Step 1.1. Scan the bit-string database, if all items in S are contained in a record, then output the items in this record and the bit string for each item into *1-itemset database*. If S is a frequent sequence, then find all frequent 1-itemsets. The frequent itemsets are found in each iteration. For the *k*th iteration ($k \geq 1$), the candidate ($k+1$)-itemsets are generated, and scan the *(k+1)-itemset database* to find ($k+1$)-frequent itemsets.Finally, we output the frequent *k*-itemsets and its bit string in each record into the *frequent itemset database*.

Step 1.2. Each frequent itemset (i.e. frequent 1-sequence) is given a unique number, and replace the frequent itemsets in the frequent itemset database with their numbers to form a 1-sequence database.

For example, in Table 2, the records which contain the sequence ⟨(E)(A)(B)⟩ in the **With** clause in Query 2 are CID 4 and CID 5, Hence, we can generate the frequent itemsets (A), (B), (C), (D), (E), (F) and (B, D), and the numbers for the frequent itemsets are 1, 2, 3, 4, 5, 6, and 7, respectively. The 1-sequence database is shown in Table 4.

Table 4
1-sequence database for Query 2

CID	1-sequence	Bit string for each 1-sequence
4	1, 2, 3, 4, 5, 6, 7	010110, 101010, 001000, 100000, 010110, 000001, 100000
5	1, 2, 3, 4, 5, 6, 7	000100, 000010, 001000, 110111, 010000,010001,000010

Step 1.3. Generate candidate 2-sequences, and scan 1-sequence database to generate *2-sequence database* and find all the frequent 2-sequences.

The candidate 2-itemsets are generated as follows: For each frequent 1-sequence f except D_1, the itemset D_1 is combined with the frequent 1-sequence to generate a candidate 2-sequence. If there is a notation '*' appears before the itemset D_1 in the **With** clause, then the candidate 2-sequence $\langle(f)(D_1)\rangle$ is generated. If the notation '*' appears after the itemset D_1, then the candidate 2-sequence $\langle(D_1)(f)\rangle$ is generated. If the reverse order of a candidate 2-sequence is contained in the specified sequence S, then this candidate 2-sequence can be pruned.

For each record in the 1-sequence database, we use the frequent 1-sequences in the record and apply the above method to generate candidate 2-sequences. Suppose that the two frequent 1-sequences X and Y in a record generate candidate 2-sequence Z. We perform the sequential bit-string operation on the two bit strings for the two frequent 1-sequences X and Y, and the resultant bit string is the bit string for the candidate 2-sequence Z. If this bit string is not zero, then output the candidate 2-sequence Z and its bit string into 2-sequence database. After scanning 1-sequence database, the 2-sequence database can be generated and the candidate 2-sequences can be counted.

For Query 2, after scanning 1-sequence database (Table 4), the generated 2-sequence database is shown in Table 5, and the frequent 2-sequences are $\langle(5)(1)\rangle$, $\langle(5)(2)\rangle$, $\langle(4)(5)\rangle$, $\langle(5)(3)\rangle$ and $\langle(5)(6)\rangle$.

Step 1.4. Generate candidate 3-sequences, and scan 2-sequence database to generate 3-sequence database and find all the frequent 3-sequences.

The method to generate candidate 3-sequences is described as follows: For every two frequent 2-sequences $S_1=\langle(D_1)(r)\rangle$ which is a sub-sequence of S and $S_2=\langle(D_1)(t)\rangle$ (or $S_1=\langle(D_1)(r)\rangle$ and $S_2=\langle(t)(D_1)\rangle$), we can generate the candidate 3-sequences $\langle(D_1)(r)(t)\rangle$ and $\langle(D_1)(t)(r)\rangle$ (or $\langle(t)(D_1)(r)\rangle$).

Step 1.5. Frequent (h+1)-sequences ($3\le h\le m$) are generated in each iteration. For the (h−2)th iteration, we use frequent h-sequences to generate candidate (h+1)-sequence, and scan h-sequence database to generate (h+1)-sequence database, and find all the frequent (h+1)-sequences.

The following method is used to generate candidate (h+1)-sequences: For any two frequent h-sequence $S_1=\langle(s_1)(s_2)\ldots(s_{h-1})(r)\rangle$ and $S_2=\langle(s_1)(s_2)\ldots(s_{h-1})(t)\rangle$, in which either $\langle(s_1)(s_2)\ldots(s_{h-1})\rangle$ is a sub-sequence of S or (r) and (t) are contained in S, the candidate (h+1)-sequences $\langle(s_1)(s_2)\ldots(s_{h-1})(r)(t)\rangle$ and $\langle(s_1)(s_2)\ldots(s_{h-1})(t)(r)\rangle$ can be generated. If a generated candidate (h+1)-sequence contains more than one itemsets which are not contained in S, then the candidate (h+1)-sequence can be pruned.

For each record in the h-sequence database, we use the frequent h-sequences in this record and the above method to generate candidate (h+1)-sequences. For each generated candidate (h+1)-sequence, we perform the sequential bit-string operation on the two bit strings for the two frequent h-sequences, which generate the candidate (h+1)-sequence. The resultant bit string is the bit string for the candidate (h+1)-sequence in this record. If the resultant bit string is not zero, then output the candidate (h+1)-sequence and its bit string into (h+1)-sequence database, and count the support for the candidate (h+1)-sequence. After scanning the h-sequence database, the (h+1)-sequence database can be generated and the supports for the candidate (h+1)-sequences can be computed.

For example, the frequent 3-sequences are $\langle(5)(1)(2)\rangle$, $\langle(4)(5)(1)\rangle$, $\langle(4)(5)(2)\rangle$, $\langle(5)(3)(1)\rangle$, $\langle(5)(3)(2)\rangle$, $\langle(5)(1)(6)\rangle$ and $\langle(5)(2)(6)\rangle$, and the generated candidate 4-sequences are $\langle(4)(5)(1)(2)\rangle$, $\langle(5)(3)(1)(2)\rangle$, $\langle(5)(1)(6)(2)\rangle$ and $\langle(5)(1)(2)(6)\rangle$. After scanning 3-sequence database, the frequent 4-sequences are $\langle(4)(5)(1)(2)\rangle$, $\langle(5)(3)(1)(2)\rangle$ and $\langle(5)(1)(2)(6)\rangle$.

Step 2. The frequent (m+n+1)-sequences ($n\ge 1$) which contain the specified sequence S are generated in each iteration. For the nth iteration, we use the frequent (m+n)-sequences to generate candidate (m+n+1)-sequences and scan the (m+n)-sequence database and 1-sequence database to generate (m+n+1)-sequence database in which the candidate (m+n+1)-sequences are contained in each record but the bit strings are not, and find the frequent (m+n+1)-sequences.

The method to generate candidate (m+n+1)-sequences is as follows: For every two frequent (m+n)-sequences $S_1=\langle(s_1)(s_2)\ldots(s_i)(r)(s_{i+1})\ldots(s_{m+n-1})\rangle$ and $S_2=\langle(s_1)(s_2)\ldots(s_j)(t)(s_{j+1})\ldots(s_{m+n-1})\rangle$ ($i\le j$), in which (r) is not contained in S_2 and (t) is not contained in S_1, a candidate (m+n+1)-sequence $\langle(s_1)(s_2)\ldots(r)\ldots(t)\ldots(s_{m+n-1})\rangle$ can be generated. For each record in (m+n)-sequence database, we also use every two frequent (m+n)-sequences in this record and apply the above method to generate a candidate (m+n+1)-sequence, and perform the sequential bit-string operations on the bit strings for the itemsets in the candidate (m+n+1)-sequence by scanning the 1-sequence database. If the resultant bit string is not zero, then output the candidate (m+n+1)-sequence into the (m+n+1)-sequence database, and count the support for the candidate

Table 5
2-sequence database for Query 2

CID	2-sequence	Bit string for each 2-sequence
4	$\langle(3)(5)\rangle$,$\langle(4)(5)\rangle$,$\langle(7)(5)\rangle$,$\langle(5)(1)\rangle$,$\langle(5)(2)\rangle$,$\langle(5)(3)\rangle$,$\langle(5)(6)\rangle$	000110, 010110, 010110, 000110, 001010, 001000, 000001
5	$\langle(4)(5)\rangle$,$\langle(5)(1)\rangle$,$\langle(5)(2)\rangle$,$\langle(5)(3)\rangle$,$\langle(5)(4)\rangle$,$\langle(5)(6)\rangle$,$\langle(5)(7)\rangle$	010000, 000100, 000010, 001000, 000111, 000001, 000010

(m+n+1)-sequence. After scanning (m+n)-sequence database, the (m+n+1)-sequence database can be generated and the frequent (m+n+1)-sequences can be found.

For the above example, according to step 2, the generated candidate 5-sequences are ⟨(4)(5)(3)(1)(2)⟩, ⟨(4)(5)(1)(2)(6)⟩ and ⟨(5)(3)(1)(2)(6)⟩. After scanning the 4-sequence database, the frequent 5-sequences are ⟨(4)(5)(3)(1)(2)⟩, ⟨(4)(5)(1)(2)(6)⟩ and ⟨(5)(3)(1)(2)(6)⟩. These frequent 5-sequences can further generate candidate 6-sequence ⟨(4)(5)(3)(1)(2)(6)⟩.

Step 3. For each frequent sequence, the code for each itemset in the frequent sequence is replaced with the itemset itself. If a frequent sequence is not contained in another frequent sequences, then this frequent sequence is a sequential pattern.

For the above example, the frequent sequences which satisfy the user requirement in Query 2 are ⟨(E)(A)(B)⟩, ⟨(D)(E)(A)(B)⟩, ⟨(E)(C)(A)(B)⟩, ⟨(E)(A)(B)(F)⟩, ⟨(D)(E)(C)(A)(B)⟩, ⟨(D)(E)(A)(B)(F)⟩, ⟨(E)(C)(A)(B)(F)⟩ and ⟨(D)(E)(C)(A)(B)(F)⟩, and the sequential pattern is ⟨(D)(E)(C)(A)(B)(F)⟩.

5. Experimental result

In this section, we evaluate the performance of algorithms MIAR and MISP. For mining interesting association rules, we use four datasets, IBM-Artificial, BMS-POS, BMS-WebView-1 and BMS-WebView-2, to evaluate the performances of MIAR algorithm in comparison with constrained FP-growth algorithm (Pei and Han, 2002). The first dataset was generated using a transaction data generator obtained from IBM Almaden (http://www.almaden.jbm.com/cs/quest//syndata.html # assoc Syn Data) and we design the dataset as T10I4D100K, which is often used in the association rule research community. The last three datasets are the real-world data sets used in KDD CUP 2000. BMS-POS dataset contains several years worth of point-of-sale data from a large electronics retailer. BMS-WebView-1 and BMS-WebView-2 datasets contain several months worth of clickstream data from two e-commerce web sites. Table 6 describes the four datasets in terms of the number of transactions, the number of distinct items, the maximum transaction size, and the average transaction size.

We also generate three queries to compare MIAR algorithm with constrained FP-growth algorithm. For the three queries, the With clauses in our mining language for

Table 6
Dataset characteristics

	Number of trans-actions	Distinct items	Maximum transaction size	Average transaction size
IBM-artificial	100,000	870	29	10.1
BMS-POS	515,597	1,657	164	6.5
BMS-webview-1	59,602	497	267	2.5
BMS-webview-2	77,512	3,340	161	5.0

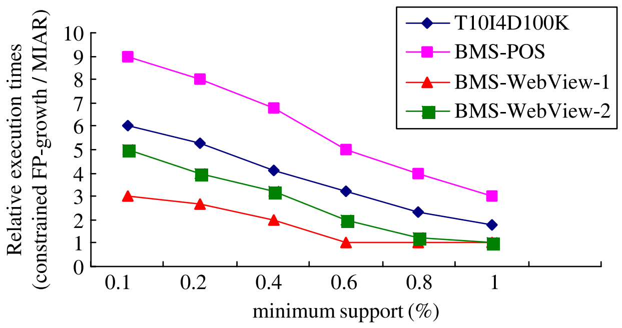

Fig. 1. Relative execution times for different datasets.

Query1, Query2 and Query3 are (itemset, *)(*), (*)(itemset, *) and (itemset, *)(itmeset, *), respectively, in which we arbitrarily choose two frequent items in the itemset.

For all the datasets and the different minimum support thresholds, Fig. 1 shows the relative execution times for constrained FP-growth and MIAR for a generated query Query 1. Fig. 2 shows the relative execution times for the generated three queries in the dataset BMS-POS for different minimum support thresholds. From Figs. 1 and 2, we can see that the larger the database size or the larger the minimum support threshold, the larger the performance gap between constrained FP-growth and MIAR, and MIAR outperforms constrained FP-growth. This is because when the size of the dataset increases or the minimum support decreases, the number of the frequent itemsets increases. Hence, constrained FP-growth needs to take a lot of time to construct more conditional pattern bases and conditional FP trees to generate the large amount of frequent itemsets. Furthermore, the initial FP-tree and some conditional FP-trees cannot fit in main memory.

For MIAR, because the bit-string database is generated, the support for each candidate can be counted by easily performing logical AND operations. Although MIAR also needs to construct itemset databases, the space for storing the conditional pattern base and FP-tree are larger than the space for storing the itemset database. In Fig. 2, because there are more items specified in Query 3 than that of Query 1 and Query 2, there are more frequent itemsets generated for Query 1 and Query 2 than that of Query 3. Hence,

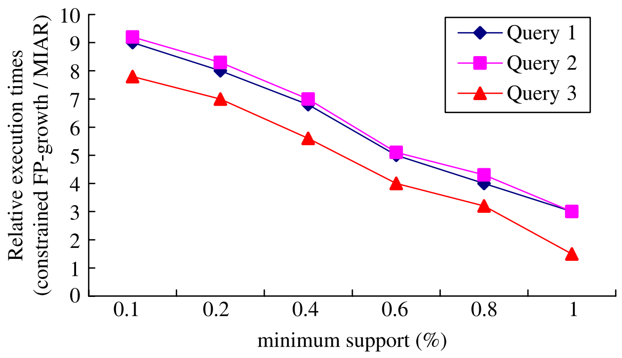

Fig. 2. Relative execution times for dataset BMS-POS.

Table 7
Dataset characteristics

	Number of customers	Number of transactions	Distinct items	Average number of transactions per customer
IBM-artificial	250,000	2,500,435	100,00	10
BMS-POS	515,597	3,367,020	1,657	6.5
BMS-webview-1	59,602	149,639	497	2.5
BMS-webview-2	77,512	358,278	3,340	5.0

the performance gap is slightly larger for Query 1 and Query 2 than that of Query 3.

For mining interesting sequential patterns, we also use three real-world data sets BMS-POS, BMS-WebView-1 and BMS-WebView-2 used in KDD CUP 2000 and generate a synthetic transaction data set C10-T8-S8-I8, in which C is the average number of transactions per customer, T is the average number of items per transaction, S is the average length of maximal potentially frequent sequences, and I is the average size of itemsets in maximal potentially frequent sequences, to evaluate the performance of MISP algorithm in comparison with PrefixSpan algorithm (Pei et al., 2001).

In the synthetic datasets, the number of customers is set to 250,000, and the number of items is set to 10,000. The synthetic datasets we used for our experiments were generated using standard procedure described in (Agrawal and Srikant, 1995). Table 7 describes the four datasets in terms of the number of customers, the number of transactions, the number of distinct items, and the average number of transactions per customer.

We also generate three queries Query 1, Query 2 and Query 3, in which the numbers of the specified items are 5, 3 and 2, respectively. Fig. 3 shows the relative execution times for PrefixSpan (Pie et al., 2001) and MISP for Query 2. Fig. 4 shows the relative execution times for the generated three queries in the dataset BMS-POS for different minimum support thresholds.

The experimental results show that MISP outperforms PrefixSpan algorithm, and the performance gap increases as the minimum support threshold decreases because when the minimum support decreases, the number of the frequent sequences increases, the number of the projected databases increases and the size of each projected database also increases, such that the performance is degraded for PrefixSpan algorithm. Besides, PrefixSpan algorithm needs to take extra time to pick the frequent itemsets from the large amount of frequent itemsets to match the user queries. However, for MISP, we only focus on the items specified in user queries, that is, there is no redundant frequent sequence can be generated. Hence, MISP can significantly outperform PrefixSpan algorithm.

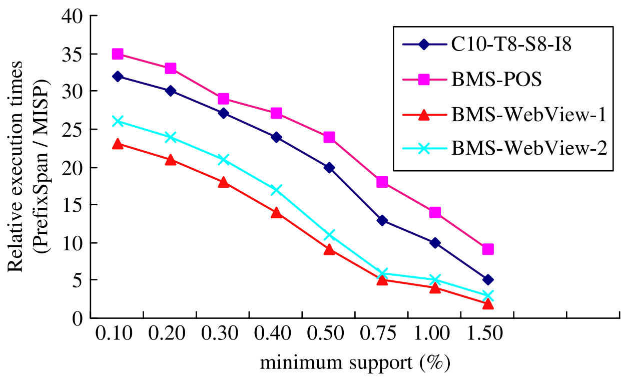

Fig. 3. Relative execution times.

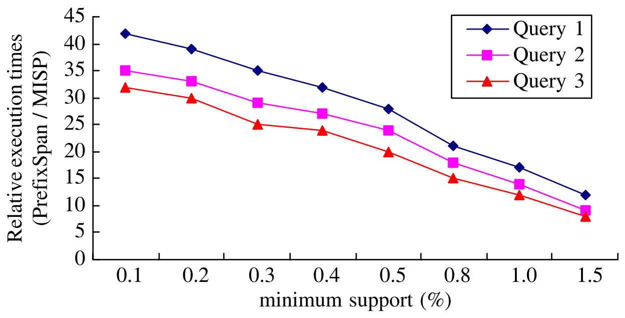

Fig. 4. Relative execution times for dataset BMS-POS.

For the algorithm MISP, because there are more items specified in Query 1 than that of Query 2 and more items specified in Query 2 than that of Query 3, there are fewer candidates, smaller bit-string database and smaller sequence databases for Query 1 than that of Query 2 and for Query 2 than that of Query 3. Hence, the performance for executing Query 1 is better than that of Query 2 and the performance for executing Query 2 is better than that of Query 3.

6. Conclusion

In this paper, we introduce a data mining language. From the data mining language, users can specify the interested items or the sequences, and the minimum support and the minimum confidence threshold to discover association rules and sequential patterns.

We propose the efficient data mining algorithms MIAR and MISP to process the user requirements. Our algorithms can reduce the number of the combinations of itemsets or sequences in each customer sequence for counting the supports of the candidates, and reduce the number of the candidates according to the user's requests. In order to improve the efficiency, we generate bit-string database and itemset (sequence) databases and propose a sequential bit-string operation for counting the supports of the candidates by easily performing logical bit operations. Although the bit-string database and itemset (sequence) database cost extra memory space, it is more important to reduce the response time for a data mining query system.

References

Agrawal, R. & Srikant, R. (1995). Mining sequential patterns. In *Proceedings of international conference on data engineering* (ICDE). pp. 3–14.

Han, J. & Pei, J. (2000). Mining frequent patterns by pattern-growth: Methodology and implications. In *Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining*. pp. 30–36.

Hipp, J., & Guntzer, U. (2002). Is pushing constraints deeply into the mining algorithms really what we want?—an alternative approach for association rule mining. *In SIGKDD Explorations*, *4*(1), 50–55.

Meo, R., Psaila, G., & Ceri, S. (1996). A new SQL-like operator for mining association rules. In *Proceedings of international conference on very large data base*. pp. 122–133.

Ng, R., Lakshmanan, L.S., Han, J., & Mah, T. (1999). Exploratory mining via constrained frequent set queries. In *Proceedings of ACM SIGMOD*. pp. 556–558.

Pei, J. & Han, J., (2000). Can we push more constraints into frequent pattern mining? In *Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining*. pp. 350–354.

Pei, J, & Han, J., (2002). Constrained frequent pattern mining: A pattern-growth view. *In SIGKDD Explorations*, *4*(1), 31–39.

Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., & Hsu, M.C. (2001). PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of international conference on data engineering*. pp. 215–224.

Pei, J., Han, J., & Wang, W. (2002). Mining sequential patterns with constraints in large databases. In *Proceedings of ACM conference on information and knowledge management* (CIKM). pp. 18–25.

Yen, S.J., & Chen, A.L.P. (1997). An efficient data mining technique for discovering interesting association rules. In *Proceedings of international conference on database and expert systems applications* (DEXA). pp. 664–669.