Mining Temporal Association Rules in Network Traffic Data

Guojun Mao

***Abstract*—Mining association rules is one of the most important and popular task in data mining. Current researches focus on discovering frequent itemsets that is an important step to it. Many algorithms for discovering frequent itemsets have been proposed. However, for a large database, an efficient mining algorithm must be a better balance in I/O cost and main memory load. Most traditional algorithms, like *Aprioir* [Agrawal, 1993], often take higher I/O cost because of multi-scan over the analyzed database. There have been a few of algorithms, like *FP-Tree* [Han, 2000], use a limited pass numbers to databases, but they could suffer from the shortage of main memory as there does not consider time constraints to association rules. In the paper, we first discuss the problem of mining temporal association rules in databases. Then, we create the necessary *sub-operators* between itemsets and *interval operators* between time intervals to mine temporal association rules. Finally, a new algorithm called *MTAR_Sub* for mining temporal association rules is designed and discussed.**

***Index Terms*—Association rule, data mining, frequent itemset, network traffic, temporal constraint.**

I. Introduction

The Problem of mining association rules has widely been discussed since 1993[1]. The aim of mining association rules is to identify relationships between items in such databases as market basket databases. Many supermarkets, for example, need a better understanding of individuals' purchasing habits through customer transaction databases. To make a good decision, the uses of this information vary, including preparing purchasing plans, designing personalized marketing campaigns, organizing product placements and determining the timing of product promotions [1]. Similar facts may hides in network traffic, spatial or medical databases [2].

As far as is network traffic data is concerned, a traffic record in the network traffic data is always made up of some useful values related to specific network attributes, e.g. the source IP address, the destination IP address, the source IP port, the destination IP port, the protocol type, the begin time, and the end time. Mining association rules from the network traffic data can help find out the regular patterns hides in network interaction activities.

Indeed, many algorithms have been designed in the literatures for mining association rules. However, most previous algorithms have two key weaknesses. The first is that they require multi-pass the database, which would result in a bigger I/O cost. The second is that they simply see a transaction as an itemset such that the appearing or ending time of an itemset is ignored. Mining in such data as network traffic ones need fight the above two weaknesses. First, the traffic data in a network system is relatively large, so it is necessary to use more efficient algorithms rather than multi-scan methods. Second, whether or not an association rule is available should be considered in the specific time intervals.

In fact, considering a format of traffic records such that <*SIP*, *DIP*, *Sp*, *Dp*, *Pro*, *Interval*>, respectively denote the source IP address, the destination IP address, the source IP port, the destination IP port, the protocol type and the durative time interval, we can handle it as follows:

- According to <*SIP*, *DIP*, *Sp*, *Dp*, *Pro* >, all records can be divided into the different analyzing objects. Such an object can be seen as term *item* as in transaction databases.
- Attribute *Interval* reflects the time-related character of a record. This can help discover temporal association rules.

In this paper, we design a one-scan algorithm, and based on this algorithm, we describe *MTAR_Sub*, an algorithm for mining temporal association rules by using *sub-operators* between itemsets in the database.

The rest of this paper is organized as follows. In Section II, we give some concept descriptions such as the temporal association rule, and analyze precious relative work. Section III introduces a one-scan algorithm *MAR_Sub* that is based on *sub-belong* operator between itemsets. Section IV describe and interpret our new algorithm called *MTAR_Sub*, which may effectively mine time-related frequent itemsets in large databases. The performances of these algorithms are discussed in Section V.

II. Problem Description

Based on Literature [1], this section gives the descriptions of temporal association rule and so on.

Considering a time-related transaction database $\boldsymbol{D} = \{d_i\}$, each transaction d_i has be organized into two attributes values related to the attributes of *interval* and *itemset*, respectively represents durative time interval and inclusion items of this transaction. Let $\boldsymbol{I}$ be an investigated set of items, and $\boldsymbol{T}$ be an interested time interval. Supposed for each tuple in $\boldsymbol{D}$, its itemset contains in $\boldsymbol{I}$ and its interval falls in $\boldsymbol{T}$, an temporal association rule in $\boldsymbol{D}$ is an implication of the form $x \Rightarrow y$ in $[t_1, t_2]$, where $x, y \subset \boldsymbol{I}$, $x \cap y$, and $[t_1, t_2] \subseteq \boldsymbol{T}$ is the time interval that $x \Rightarrow y$ is available.

A. Measures

In general, mining association rules can be decomposed into two sub-problems, i.e. discovering frequent itemsets by

Manuscript received March 9, 2013; revised May 6, 2013. This work was supported in part by the CHINA National Science Foundation under Grant 61273293 and the Discipline Construction Foundation of CUFE.

Guojun Mao is with the School of Information, Central University of Finance & Economics, Beijing, China, 100081 (e-mail: maximmao@hotmail.com).

the support measure and creating association rules by the confidence measure. So is to mine the temporal association rules.

Given an itemset $x \subseteq \boldsymbol{I}$ and time interval $[t_1, t_2]$**,** the *support count* of x in $[t_1, t_2]$ is defined as the number of transactions that contain x and occur during $[t_1, t_2]$ in ***D*,** and the *support* of x in $[t_1, t_2]$, *support* $(x, [t_1, t_2])$, is defined as the ratio of its *support count* to the number of transactions that occur during $[t_1, t_2]$ in ***D***. A temporal association rule can hold *confidence* $(x => y, [t_1, t_2])$, the confidence measure can be calculated by *support*(x union y, $[t_1, t_2]$) / *support*(x, $[t_1, t_2]$).

The problem of mining temporal association rules is to find all association rules in the database that satisfies at least three user-specified constraints: *minimum support*, *minimum confidence* and *time interval*.

B. Previous Work

While generating rules after obtaining frequent itemsets is relatively straightforward, discovering frequent itemsets from a database is a more important and various task, and so more efforts were taken in mining frequent itemsets in databases. *Apriori* has been considered as the most classical algorithm for discovering frequent itemsets [1]. In order to make use of the limited main memory, *Apriori* algorithm employs an iterative approach to generate frequent itemsets. A frequent itemset with the size k can be generated just after all frequent itemsets with the size k-1 are found. Obviously, Apriori is multi-scan mining algorithm into the databases, so its I/O burden is a big problem.

After *Apriori*, some new techniques were used to improve mining efficiency. For example, [3] uses partition technique to divide the database into a series of smaller dataset; [4] applies hash method to reduce the time of calculating the support measure; and [5] uses sampling method to reduce the size of the mined database. Obviously, this kind of algorithms cannot get expected mining efficiency because they still have to do multi-scan over the database.

FP-Tree [6] is the first algorithm that mines frequent itemsets without generating candidate itemsets. It converts the information from the database into a tree in main memory and makes it possible to mine association rules by one or two passes over databases.

Instant [7] presents a real one-scan algorithm to the database. It makes use of two linear data structures in main memory to make discovering frequent itemsets more efficient. Indeed, with increasing the sizes of databases, *Instant* will be faced with new challenge. This paper will design the new algorithm for mining rules from larger databases by using temporal constraints.

Constraints can make the mining more effective and more efficient [8, 9]. If a database holds temporal attributes, they often play a critical role in processing information from the database [10]. In general, a specific user is always interested in the data within the specified time intervals rather than the full database. Also, an association rules can be available during specific a time interval. Therefore, making use of temporal constraints can become a better choice to some applications for mining associate rules in large databases.

III. One-Scan Alogorithm

Neglecting the other factors such as appearing time, a tuple in a transaction database can simply be seen as an itemset, and so all transactions in the database can be seen as a set of itemsets. Behind our one-scan algorithm, we find out a new operator between itemsets, called *sub-belong*, rather than a traditional set operator.

Definition 1(*sub-belong* operator). Let $\boldsymbol{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of items which determines the scope of items in the discussed project. Let x be an itemset with $x \subset \boldsymbol{I}$, and Y be a set of itemsets that each itemset is contained in $\boldsymbol{I}$. We define x ***sub-belongs*** to Y if x is contained by any itemset in Y, *i.e.* $\exists\, y \in Y$: $x \subseteq y$, denoted by $x \in_{sub} Y$.

Example 1. If $Y_1 = \{AB, CD\}$ and $Y_2 = \{ABCD, AD\}$, then according to the traditional set operator, $AB \in Y_1$, but we cannot say $AB \in Y_2$. However, as AB is a sub-itemset of itemset ABCD, then we can say $AB \in_{sub} Y_2$.

This operator between itemsets can make discovering frequent itemsets in databases more simple and efficient, so we will make use of it to construct our one-scan mining algorithm called *MAR_Sub*.

MAR_Sub employs two main data structures in the memory called *SIS* and *SIS** to record related sets. The notation of some important structures is given in Table I. Algorithm 1 is pseudo-code of *MAR_Sub*.

TABLE I: The Data Objects for the Algorithm

Name	Meanings
SIS	The set of itemsets obtained by scanning the database ***D*** that are useful in the future to *MAR_Sub*
*SIS**	The set of frequent and available itemsets produced by *MAR_Sub*
Support(IS)	The support measure of itemset *IS*

Algorithm 1. MAR_Sub: Mining Association Rules with Sub-operators

1) input (min-support);
2) SIS ← Ø; SIS* ← Ø;
3) **for** all d∈ D **do begin**
4) IS ← itemset of d;
5) Join (IS, SIS);
6) make_fre (IS, SIS, SIS*, min-support) ;
7) **end**
8) Answer←SIS*.

In Algorithm 1, each of iterations is related with a tuple of the database and consists of three phases:

- An itemset (called *IS*) is abstracted from the tuple of the database.
- *IS* is tested and can be inserted into *SIS*, and its *support* can be recalculated when it is necessary.
- *SIS** can be updated through *IS*, new *SIS* and given *min*-support.

We first discuss Procedure *join*(*IS, SIS*). If *IS* has not been in *SIS*, This process puts *IS* into *SIS*, and recalculates its support measure *support*(*IS*). That is:

- If IS ∉ SIS_{old}, insert *IS* into *SIS*, *support*(*IS*) =1/ |*D*|;

- If $IS \in SIS_{old}$, $support(IS) = support(IS)_{old} + 1/|D|$.

Procedure *make_fre*(*IS*, *SIS*, *SIS**, *min-support*) tries to find sub-itemsets of *IS* which can become new frequent ones. Its core idea is based on the following: $\forall IS^* \in_{sub} \{IS\}$, *IS** may become a new frequent itemset if *support* (*IS**) $\geq$ *min-support*. Additionally, to reduce the size of *SIS* in memory, we insert such *IS** only when *IS* does not exist in *SIS**, and such a processing is effective and complete [1].

Procedure 1 gives the pseudo-code of *make_fre*(*IS*, *SIS*, *SIS**, *min-support*).

Procedure 1. *make_fre*(*IS*, *SIS*, *SIS**, *min-support*)
```
for all IS*∈sub {IS} do begin
  support (IS*) ← 0;
  for all s∈ SIS do
    if  IS*∈sub{s}  support(IS*)←support(IS*)+support
(s);
  if support (IS*) >= min-support
    if IS*∉sub SIS*  do begin
      prune(IS*, SIS*); //see the following Procedure 2
      insert IS* into SIS*;
    end
  prune(IS*, SIS); //see  Procedure 2
end;
```

In Procedure 1, Procedure *prune*() are called to reduce memory usages. According to the Lemma in [1], all sub-itemsets of a frequent itemset are definitely frequent. Therefore, we only keep such frequent itemsets that are not contained by any other itemset in *SIS**. It is reasonable to prune the sub-itemsets of *IS** in *SIS** before *IS** is inserted into *SIS**. Obviously, when *IS** is frequent, its sub-itemsets should also be cut from *SIS*. This is because that since *IS** is frequent, it and its sub-itemsets are surely frequent and so they no longer need to be checked in the future. These jobs can be done by using the above *sub-belong* operator. That is, for each itemset *x* in *X*, if $x \in_{sub} \{y\}$, then we can call Procedure *prune*(*y*, *X*) to delete *x* from *X*, and its description is omitted here

Example 2. Given the transaction database shown in Table II. Table III provides the processing details by using *MAR_Sub*, where *min-support* = 40%, i.e. *minimum support count* is 2, and (ABCD, 1) represents that the itemset is ABCD and its support count is 1.

IV. MTAR_SUB ALGORITHM

In this section, we will discuss mined temporal association rules from the transaction databases with the temporal attributes. Such databases can be used as the particular kind of data organizations such as network traffic records.

Table IV gives such a database sample that contains the attributes of *interval* and *itemset*. Such databases can be derived from original network traffic datasets.

From such databases shown as Table IV, we can mine the temporal association rules as stated in Section 2.1. Note that the value of each *interval* attribute in Table IV has been formatted into an *integer-interval* to process in a normal way, and its implementation is easy to computers, so here its processing detail is omitted.

TABLE II: THE DATA SAMPLE

No.	Items
1	A, B, C, D
2	B, C, E
3	A, B, C, E
4	B, D, E
5	A, B, C, D

TABLE III: PROCESS OF THE SAMPLE

#	*IS*	*SIS*	*SIS**
0		Ø	Ø
1	ABCD	{(ABCD, 1) }	Ø
2	BCE	{(ABCD, 1) , (BCE, 1) }	{BC}
3	ABCE	{(ABCD, 1) , (ABCE, 1) }	{BCE}
4	BDE	{(ABCD, 1) , (ABCE, 1) , (BDE, 1) }	{*BCE*, *BD*}
5	ABCD	{(ABCE, 1) , (BDE, 1) }	{*BCE*, *ABCD*}

TABLE IV: THE SAMPLE WITH TIME ATTRIBUTE

No.1	Interval	Itemset
1	[10,50]	A, B, C, D
2	[30,60]	B, C, E
3	[70,80]	A, B, C, E
4	[70,120]	B, D, E
5	[70,90]	A, B, C, D
6	[300,500]	A, B, C, D,E

Table IV considers such a database with the attributes *interval* and *itemset*, respectively provide interesting time interval and collected itemset within this interval. As far as interesting time interval is concerned, we need consider both user time focus and real data in the database. If the user specifies a time interval called *Tuser*, then for any itemset in the database, the interesting time intervals should be fell in *Tuser*. For example, for Table IV, if *Tuser* = [20, 100], the interesting time interval of itemset ABCD should be [20, 90]. Following with this idea, we first give two operators between time intervals which are going to be used to preprocess time intervals before mining temporal association rules.

Definition 2 (Interval operators). Let $t_1 = [t_1^-, t_1^+]$ and $t_2 = [t_2^-, t_2^+]$ be two variables of time intervals, then we define:

- ***interval*** intersection($\prod$): If $t_1^+ >= t_2^-$ or $t_2^+ >= t_1^-$, then $t_1 \prod t_2 = [max(t_1^-, t_2^-), min(t_1^+, t_2^+)]$; others, $t_1 \prod t_2 = \emptyset$.
- interval ***union*** ($\sum$): If $t_1^+ >= t_2^-$ or $t_2^+ >= t_1^-$, then $t_1 \sum t_2 = [min(t_1^-, t_2^-), max(t_1^+, t_2^+)]$; others, $t_1 \sum t_2 = \emptyset$.

To make closer and more precise the time interval to a temporal association rule, we can use Operator $\prod$ to filter the database by *Tuser* that the user specifies. Filtering the database means removing the data that the user is not interested in, which can obviously improve the mining quality and efficiency. Procedure 2 is the pseudo-code for filtering process by using $\prod$.

Procedure 2. *filter*(*D*, *Tuser*, D_1)
```
1) for all d∈D
2)  if (d.interval ∏ Tuser ≠ Ø) do begin
3)  i ← d.interval ∏ Tuser;
4)  j ← d.itemset;
```

5) Insert < i, j > into D_1;
7) **end;**

After filtering, the data not within *Tuser* are removed, but such fact may occur that intervals of tuples are scattered and crossed one another. Therefore, it is necessary to consider some suitable time intervals, called *mining intervals*, to discover time-related frequent itemsets or mine temporal association rules. We will explore Operator $\sum$ to merge correlative time intervals in the database into non-overlapping intervals, such that any time-related frequent itemset or rule will be associated with one of these non-overlapping intervals. Procedure 3 gives the pseudo-code for merging intervals by using $\sum$.

Procedure 3. Merge ($D1$, TS)
1) TS ← Ø;
2) for all d∈ $D1$
3) i← d.interval;
4) if TS is not Ø
5) for all j ∈ TS
6) if (j $\sum$ $i \neq$ Ø) do begin
7) $i \leftarrow j \sum i$;
8) delete j from TS ;
9) end
10) insert i into TS;

Example 3. For the database in Table IV, supposed *Tuser* = [30, 100], the result to filter by Procedure 2 is shown in Table V. Also, by Procedure 3, we can obtain two non-overlapping mining intervals: [30, 60] and [70, 100].

TABLE V: FILTERING RESULT TO TABLE IV

No.1	Interval	Itemset
1	[30,50]	A, B, C, D
2	[30,60]	B, C, E
3	[70,80]	A, B, C, E
4	[70,100]	B, D, E
5	[70,90]	A, B, C, D

Now, we can design our algorithm *MTAR_Sub*. Algorithm 2 gives its pseudo-code description.

Algorithm 2. MTAR_Sub: Mining Temporal association rules with Sub-operators
1) input (min-support, Tuser) ;
2) filter (D, Tuser, $D1$);
3) Merge (D1, TS);
4) for all t ∈ TS do
//for each mining interval, mining time-related rules
5) SIS ← Ø; SIS* ← Ø;
6) for d ∈ $D1$ do begin
7) if d.interval $\prod t \neq$ Ø do begin
8) IS ← d.itemset;
9) Join (IS, SIS) ; //see Sec. 3
10) make_fre(IS, SIS, SIS*, min-support); // see Sec. 3
11) end
12) add time flag t all itemsets in SIS*;
13)end.

In fact, in Algorithm *MTAR_Sub*, line 5) to 10) do mining within a mining interval, so found frequent itemsets are related to this time interval. Such that, all discovered frequent itemsets are time-related, so further mined rules are all temporal association rules.

About how to mine temporal association rules after getting frequent itemsets, it is relatively straightforward and often do the same work as most literatures [1]. Therefore, we will not provide a formal description in this paper.

V. EXPERIMENTS

We implemented the *MTAR_Sub*, *MAR_Sub* and *Apriori* [1] on a computer with Pentium 4 of 512M RAM.

In our experiments, the tested datasets are extracted from some network traffic datasets with attributes <*SIP*, *DIP*, *Sp*, *Dp*, *Pro*, *Interval*>. We arrange these datasets into the data tables containing the two attributes <*interval*, *itemset*>. According to the different values of <*SIP*, *DIP*, *Sp*, *Dp*, *Pro*>, the investigated items have 20 values, and the time interval scope is [20, 900]. To test the effectiveness and efficiencies of the methods, we used a series of time-related transaction databases, with 1000~10000 tuples, every itemset contains 2~20 items.

Using T_{user} = [30, 500], *min-support* = 20% and *min*-confidence = 80% in the database with 10000 tuples that each has less than 21 items represented by capital letters, the following are some were found rules:

- <A,C> => D (24.13%, 98.24%, [30, 68])
- <D,F,J> => A (26.81%,91.60%, [30, 68])
- <I> => F (34.03%, 99.20, [82, 314])
- <F,H> => I (22.18%, 97.12, [82, 314])
- <B,C> => G (21.50%, 96.83, [421, 500])
- <B,G,S> => J (20.56%, 89.61, [421, 500])

The first experiment on *MTAR_Sub* was conducted to test the main memory spaces used by *SIS* and *SIS** with increasing sizes of the datasets. If *min-support* is 20%, the memory usages of *SIS* and *SIS** on some different data sizes are shown on Fig. 1.

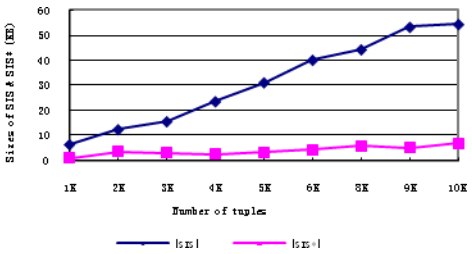

Fig. 1. Changes of *SIS* and *SIS** on different sizes of datasets in *MTAR_Sub*. Tuser = [30, 500], min-support = 20%. On the 10K database, three mining intervals are generated: [30, 68], [82, 314] and [421, 500].

The second experiment about *MTAR_Sub* was done on a dataset with 10000 tuples, using different *min-support* ranges. It aims at testing main usages to generate *SIS* and *SIS** with different min-supports. Fig. 2 shows experimental results.

The third experiment is the performance comparison *MTAR_Sub* with *Apriori* [1]. This experiment uses min-support of 20% on a series of datasets with 1000~9000 tuples. Their execution times with increasing sizes of the datasets are shown on Fig. 3.

These results tell us that *MTAR_Sub* has better availabilities in many situations. In summary, these experiment studies

illustrate: (1) how *MTAR_Sub* can efficiently work as it uses limited spaces of the main memory and take a little of time to run, and (2) how *MTAR_Sub* can effectively generate time-related association rules from the temporal transaction databases as it make mining goal into specific the temporal interval . These achievements are mainly from its effective operators for scanning and filtering to the investigated databases.

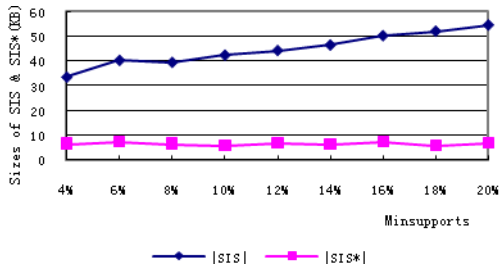

Fig. 2. Changes of *SIS* and *SIS** on different min-supports in *MTAR_Sub*. *Tuser* = [30, 500]. Three *mining intervals* are generated: [30, 68], [82, 314] and [421, 500].

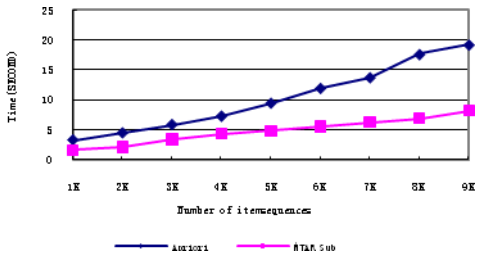

Fig. 3. Run times of Ariori and MTAR_Sub on different sizes of databases. Tuser = [30, 500] in MTAR_Sub, min-supports = 20% in both.

VI. Conclusion

We have presented an algorithm to efficiently mine temporal association rules from transaction databases. In fact, we are facing two challenges in mining association rules from a large of databases: (1) how to reduce the numbers to scan the database in order to make I/O cost lower, and (2) how to use constraints in order to make mine rules more effective. We solve the first issue using a new operator between itemsets called *sub-belong* that can help construct the one-scan mining algorithm. We address the second problem by exploiting temporal constrains that can make an association rule is related to a specific time interval.

We did experiments to test our method with the case of network traffic data, and these experiments have shown this method how to work more effective and efficient.

Acknowledgment

Guojun Mao thanks the CHINA National Science Foundation and the Discipline Construction Foundation of CUFE to supports for this work.

References

[1] R. Agrawal, T. Imielinki, and A. Swami, "Mining association rule between sets of items in large database," in *Proc. SIGMOD'93*, Washington, DC, USA, May, 1993, pp. 207-216.
[2] W. Ding, C. F. Eick, X. Yuan, J. Wang, and J. P. Nicto, "A framework for regional association rule mining and scoping in spatial datasets," *J. Geoinformatica*, vol. 15, pp. 1-28, Jan. 2011.
[3] A. Savasere, E. Omiecinski, and S. B. Navathe, "An efficient algorithm for mining association rules in large databases," Technical Report GIT-CC-95- 04, University of Maryland at College Park.
[4] J. S. Park, M. S. Chen, and P. S Yu, "An effective hash-based algorithm for mining association rules," in *Proc. SIGMOD'95*, York, NY, 1995, pp. 175-186.
[5] Y. R. Li and R. P. Gopalan, "Effective sampling for mining association rules," in *Proc. AI'04*, San Jose, CA, USA, May, 1995, pp. 391-401.
[6] J. W. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *Proc. SIGMOD'00,* Dallas, TX, USA, May, 2000, pp. 1-12.
[7] G. J. Mao, X. D. Wu, X. Q. Zhu, G. Chen, and C. N. Liu, " Mining maximal frequent itemsets from data streams," *J. Information Science*, vol. 33, pp. 251-262, Mar. 2007.
[8] K. Wang**,** H. Yu, and J. W. Han, "Pushing support constraints into association rules mining," *IEEE Trans. on Knowledge and Data Engineering*, vol. 15, pp. 642-658, Mar. 2003.
[9] M. Shaheen, M. Shahbaz, and A. Guergachi, "Contextbased positive and negative spatio-temporal association rule mining," *J. Knowledge-Based Systems*, vol. 37, pp. 261-173, Jan 2013**.**
[10] G. J. Mao and C. N. Liu, "A method of data mining based on temporal constraints," *J. Acta Electronica Sinica*, vol. 31, pp. 1690-1694, 2003.

Mao Guojun is a professor of computer science at the Central University of Finance and Economics (China). He received his Ph.D degree in Computer Application Technology from the Beijing University of Technology, China. His research interests include data mining, distributed computing and knowledge-based systems, and he has published extensively in these areas more than 100 papers in various journals and conferences, including JIS and ICDM.