Mining Association Rules to Discover Calendar Based Temporal Classification

V. SRINIVASAN
Lecturer, Department of MCA
Velalar College of Eng. and Technology,
Thindal, Erode-9.
Email: sweetsrinivasan@yahoo.com
Mobile: 98651-13150

Communication Address
Second street,
10,Jeganathapuram colony,
surampatti,Erode-638 009
Tamilnadu

M. ARUNA
Lecturer, Department of MCA
Velalar College of Eng. and Technology,
Thindal, Erode-9.
Email: saiaruna2005@yahoo.co.in
Mobile: 98651-13150

Communication Address
Second street,
10,Jeganathapuram colony,
surampatti,Erode-638 009
Tamilnadu

Abstract

A well-known approach to knowledge discovery in databases involves the identification of association rules linking database attributes. Extracting all possible association rules from a database however is a computationally intractable problem, because of the combinatorial explosion in the number of sets of attributes for which incidence-counts must be computed. We study the problem of discovering association rules that display regular cyclic variation over time. For example, if we compute association rules over monthly sales data, we may observe seasonal variation where certain rules are true at approximately the same month each year. Similarly, association rules can also display regular hourly, daily, weekly, monthly etc., variation that is cyclical in nature. We demonstrate that existing methods cannot be naively extended to solve this problem. We then present two new algorithms for discovering such rules. The first one, which we call the sequential algorithm, treats association rules and cycles more or less independently. By studying the interaction between association rules and time, we devise a new technique called *cycle rule*, which reduces the amount of time needed to find association rules. The second algorithm, is transition algorithm, we demonstrate the effectiveness of the transition algorithm through a series of experiments.

1 Introduction

An important topic in data mining research is concerned with the discovery of association rule. An association rule describes an interesting relationship among different attributes and we refer to such relationship as an association in this paper. Recent advance in data collection and storage technology have made it possible for many companies to keep large amounts of data relating to their business online.. This activity is commonly referred to as *data mining*. One major application domain of data mining is in the analysis of transactional data. It is assumed that the database system records information about user transactions, where each transaction is a collection of items. In this setting, *association rules* capture interrelationships between various items. An association rule captures the notion of a set of items occurring together in transactions.

The data mining process is controlled by a user who sets minimum thresholds for the support and confidence parameters. The user might also impose other restrictions, like restricting the search space of items, in order to guide the data mining process. However,

Proceedings of the 2008 International Conference on Computing, Communication and Networking (ICCCN 2008)
978-1-4244-3595-1/08/$25.00 © 2008 IEEE

all the above work treat all the data as one large segment, with no attention paid to segmenting the data over different time intervals. It may be the case that Sales between the interval of months and the interval of time differs for example Bread and jam are sold in summer season than compared to the winter season. Now we can we can conclude that although an association rule may have the user specified minimum confidence and support within the entire time spectrum, analysis of the data in finer time granularity may reveal that the association rule exists only in certain time intervals, and does not occur in the remaining time intervals. Casual observation of many association rules over monthly data may disclose seasonal variation where peaks occur at approximately the same month in each year. Association rules could also display regular hourly, daily, weekly, monthly etc., variation that has the appearance of cycles. If these time intervals display a periodicity, discovering these rules and their periodicities may reveal interesting information that can be used for prediction and decision making. Discovering such regularities in the behavior of association rules over time are the subject of this paper. We believe that our techniques will enable marketers to better identify trends in sales and allow for better forecasting of future demand.

In this paper we analyze that the transactional data is based on monthly time-stamped and that time intervals are specified by the user to divide the data into disjoint segments. We believe that users will typically opt for "natural" segmentations of the data based on months, weeks, days, etc., and that users are best qualified to make this decision based on their understanding of the underlying data. We refer to an association rule as *cyclic* if the rule has the minimum confidence and support at regular time intervals. Such a rule need not hold for the entire transactional database, but rather only for transactional data in a particular periodic time interval. That is, each cyclic rule must have the user specified minimum support and confidence over a specific periodic time interval.

We apply a relatively straightforward extension of existing association rule mining techniques for solving this problem. This extension treats association rules and cycles independently. It applies the existing methods for discovering association rules to each segment of data and then applies pattern matching algorithms to detect cycles in association rules. By studying the interaction between cycles and association rule mining more closely, we identify techniques called *cycle rule* which allow us to significantly reduce the amount of wasted work performed during the data mining process. We then show the effectiveness of these techniques by presenting the results of a series of experiments and conclusion.

2 Problem Definitions

Let us take the set of items and a set of transactions, where each transaction consists of a subset of the set of items, the problem of discovering association rules is defined as finding relationships between the occurrences of items within transactions. An *association rule* of the form A to B is a relationship between the two disjoint item sets A and B. An association rule is described in terms of *support* and *confidence*. The support of an itemset A is the fraction of transactions that contain the itemset. An itemset is called *large*, if its supports exceeds a given threshold supmin. The *confidence* of a rule A to B is the fraction of transactions containing A that also contain B: The association rule A to B

holds: In order to deal with cyclic association rules, we enhance the transaction model by a time attribute that describes the time when the transaction was executed. In this paper, we assume that a unit of time is given user. We denote the ith time unit, i _ 0; by ti: That is, ti corresponds to the time interval [i _ t; (i+1) _ t); where t is the unit of time. We denote the set of transactions executed in ti by D[i]:1 holds in time unit ti, if the support of A to B in D[i] exceeds supmin and the confidence of A to B in D[i] exceeds. A *cycle* c is a tuple (l; o) consisting of a length l (in multiples of the time unit) and an offset o (the first time unit in which the cycle occurs), 0 _ o < l: We say that an association rule has a cycle c = (l; o) if the association rule holds in every i^{th} time unit starting with time unit. For example, if the unit of time is a month or season based than for every season of month the arrangement for the store is to be changed accordingly. We denote the minimum and maximum cycle lengths of interest by l_{min} and l_{max}, respectively. An association rule may have multiple cycles. For example, if the unit of time is based on seasons By definition, once a cycle exists, all of its multiples with length less than or equal to l_{max} will exist. Therefore, it is only interesting to discover "large" cycles, where a large cycle is the one that is not multiple of any other cycle. Let c = (l; o) be a cycle. A time unit ti is said to be "part of cycle c" or "participate in cycle c" if o = i mod l holds. An association rule can be represented as a binary sequence where the ones correspond to the time units in which the rule holds and the zeros correspond to the time for future research units in which the rule does not have the minimum con- fidence or support. A cycle can also be represented by a binary sequence. For example, cycle (4; 3) can also be represented as 0001. Similar to association rules, itemsets can also be represented as binary sequences where ones correspond to time units in which the corresponding itemset is large and zeros correspond to time units in which the corresponding itemset does not have the minimum support.

3. Association Rules based on cycle

An example of an association rule is 90% of transactions that contains bread also contain butter; 3% of all transactions contain both of these items. The 90% is referred to as the confidence and the 3% the support of the rule. The algorithms for discovering association rules cannot be applied directly for discovering cyclic association rules. In order to use the existing algorithms for detecting cyclic association rules, one may consider extending the set of items with seasons attributes, and then generating the rules. For example, one such rule could be A to B. This approach segments the database in a manner where all transactions that have the same season attribute value are within the same segment. For example, if the season attribute is month, then all the transactions that occurred on Months will be within one segment. In this case, the support of the rule A to B is the ratio of the number of transactions that occurred on Months and contain A and B to the total number of transactions. Similarly, the confidence of this rule is the ratio of the number of transactions that occurred on Months and contain A and B to the number of transactions that occurred on Months and contain A. It is possible that this approach will detect non-existing cycles. For example, this approach may detect that every Month A to B holds, although A to B holds only every season base, or only on some Month but not on all. The ratio of the number of transactions that occurred on the remaining Months that contain both A and B to the total number of transactions is high enough to compensate for the other Months. This approach assumes that the support for A to B exceeds the

minimum support threshold. Therefore, a non-existing cycle for every Month can be detected.

3.1 The Sequential Algorithm

The existing algorithms discover the association rules in two steps. In the first step, large item sets are generated. In the second step, association rules are generated from the large item sets. The running time for generating large item sets can be substantial, since calculating the supports of items etc and detecting all the large item sets for each time unit grows exponentially in the size of the large item sets. To reduce the search space for the large item sets, the existing algorithms exploit the following property: "Any superset of a small item set must also be small." The existing algorithms calculate support for item sets iteratively and they prune all the supersets of a small itemset during the consecutive iterations. Let us refer to this pruning technique as *support-pruning.* The execution of algorithm is as follows 1. The set of candidate k_itemsets is generated by extending the large (k_1)_itemsets discovered in the previous iteration (support-pruning).
2. Supports for the candidate k_itemsets are determined by scanning the database. 3. The candidate k_itemsets that do not have minimum support are discarded and the remaining ones constitute the large k_itemsets. The idea is to discard most of the small k_itemsets during the support-pruning step so that the database is searched only for a small set of candidates for large k_itemsets. In the second step, the rules that exceed the confidence threshold conmin are constructed from the large itemsets generated in the first step with one of the existing algorithms. For our experimental evaluation of the sequential algorithm, All the rules in each time unit fit into main memory, then the running time of the cycle detection phase is feasible. However, if the rules in all the time units do not fit into main memory, then the overhead of I/O operations substantially increases the running time of the cycle detection phase and therefore the sequential algorithm may become infeasible for detecting cyclic association rules.

3.2 Increasing the efficiency of cyclic association algorithm.

The most time used for sequential algorithm is spent to calculate the support for itemsets. We present three algorithm *cycle rule, cycle skip,* and *cycle-elimination* to extract the number of itemsets for which the support must be calculated. These techniques rely on the following fact: "A cycle of the rule A to B is a multiple of a cycle of itemset. Therefore, eliminating cycles as early as possible can substantially reduce the running time of cyclic association rule. Cycle skip is a technique for avoiding counting the support of an itemset in time units. " If time unit ti is not part of a cycle of an itemset A; then there is no need to calculate the support for A in time segment D[i]:" But the cycles of an itemset A can be computed exactly only after we compute the support of A in all the time segments. It is based on the following property: "If an itemset A has a cycle (l; o), then any of the subsets of A has the cycle (l; o):" The above property implies that any cycle of itemset A must be a multiple of a cycle of an itemset that is a subset of A. This also implies that the number of cycles of an itemset A is less than or equal to the number of cycles of any of A's subset.

One can arrive at an upper bound on the cycles that an itemset A can have by looking at all the cycles of the subsets of A. By doing so, we can reduce the number of potential cycles of itemset A, which, in turn reduces the number of time units in which we need to calculate support for A. Thus, cycle rule is a technique for computing the candidate cycles of an itemset by merging the cycles of the itemset's subsets. However, it is possible in some cases that we cannot compute the candidate cycles of an itemset.

e.g1: If we know that 010 is the only cycle of item A; and 010 is also the only cycle of item B; then cycle-pruning implies that the itemset consisting of items A and B can have only the cycle 010 or its multiples. Cycle-skip need not be calculate the support for A
e.g2: If we know that 010 is the only cycle of item A and 001 is the only cycle of item B; then cycle-rule implies that the itemset A U B cannot have any cycles. Cycle-skip need not be calculated
e.g3: If the maximum cycle length we are interested is lmax and the support for itemset A is below the threshold supmin in the first lmax time units, then cycle-elimination implies that A cannot have any cycles.

The above algorithms are used to calculate the itemset accurately with different type of examples.

3.3 The Transition Algorithm

The transition algorithm used for discovering cyclic association rules in two step. In the first step, the cyclic large itemsets are discovered. In the second step, cyclic association rules are generated. In the first phase, the search space for the large itemsets is reduced using cycle-rule, cycle-skip and cycle elimination as follows. For each k; k _ 1

Step1: If k equal to 1; then all possible cycles are initially assumed to exist for each single itemset. Otherwise (if k greater than 1), cycle-rule is applied to generate the potential cycles
step2.: Time segments are processed sequentially. For each seasons unit ti :

(i) Cycle-skipping determines, from the set of candidate cycles for k_itemsets,

(ii) If a k_itemset A chosen in Step (i) does not have the minimum support in time segment; then cycle-elimination is used.

When This process terminates than the list of potential cycles for each k_itemset is empty. Cycle-rule, cycle skip and cycle-elimination can reduce the candidate k_itemsets.

3.4 Sequence Detection of cycle

The sequence of length n and the maximum cycle length of interest lmax; the running time of detecting all cycles with lengths less than or equal to lmax of the binary sequence has an upper bound of O(lmax _ n) operations. We now present a straight-forward approach to detecting cycles. This approach is composed of two steps. Initially, the set of candidate cycles contains all possible cycles. In the first step, the sequence is scanned, and each time a zero is encountered at a sequence position i, candidate cycles (j; i mod j), 1min<= j<= lmax are eliminated from the set of candidate cycles. The first step completes whenever the last bit of the sequence is scanned or the set of candidate

cycles becomes empty, which ever is first. In the second step, large cycles A straight-forward approach to eliminating cycles that are not large is to Starting from the shortest cycle, for each cycle ci = (li; oi); eliminate each other cycle cj = (lj; oj) from the set of cycles, if lj is a multiple of li and (oi = oj mod li) holds. For example, if lmax is three, and we know initially that 01, 010 and 001 cannot be cycles of a given sequence then we do not need scan bit positions 1, 5, 7, 11, etc. Also, while we are scanning the sequence, if we also eliminate candidate cycle 100 we can skip scanning every second bit of the sequence starting at that point. The transmission algorithm employees these optimization techniques. The cycle detection process can be further optimized by considering a candidate cycle ci = (li; oi) only when there is no other candidate cycle cj = (lj; oj) remaining such that ci is a multiple of cj.

4 Implementation

we present the implementation details of the prototype that we built for discovering cycle rules. We first describe the synthetic data generator used to generate our data.

Our data generator is based on the synthetic data generator used in reference paper. We augmented it to generate data for cyclic rule. The generation of the large itemsets and their weights closely follows the procedure We generate L itemsets of average size I. Each itemset is associated with a weight, which is an exponentially distributed random variable. We define a *pattern* to be the union of a set of cycles of the same length. In order to model the fact that certain rules might occur cyclically, we associate pnum patterns with each large itemset that we generate. The length of the patterns is uniformly distributed between pmin and pmax time units. The density parameter pden, which is a real number between 0 and 1, controls the number of cycles in a pattern. In order to model the fact that real world data will consist of a mixture of cyclic rules and non-cyclic rules, we use the "noise" parameter V, which is a real number between 0 and 1. In a particular time unit, a large itemset is "active" and independent of cycles with a probability v. The data generation algorithm first determines which large itemsets should be used for data generation in that time unit. This is done by checking to see if the current time t participates in any of the cycles that the itemset is associated with. Following this, a determination is made as to whether the noise parameter dictates that the itemset be used. Once this is done, the weights associated with the large itemsets determine their occurrences in the transactions for the time unit. The default values we used for the parameters in our experiments are shown in Table 1.

Number of transactions/time segment, D
Number of items, N
Avg. size of large itemsets, I
Number of large itemsets, L
Avg. transaction size, T
Number of time units, u
Avg. number of patterns, pnum
Min. length of pattern, pmin
Maximum length of pattern, pmax
Avg. "density" of patterns, pden
Avg. level of "noise" in the data generated, v

4.2 Implementation Details

The sequential algorithm is based directly on apriori, with optimizations to speed up the counting of support of itemsets of size 2. We use an array for this instead of a hash-tree when memory permits. We found the array to be a much faster technique for discovering 2-itemsets. The interleaved algorithm uses a hash-tree, to store the large itemsets, their patterns and support counts. In addition, during the processing of an individual time segment, the interleaved algorithm uses a temporary hash-tree as well. Candidate generation is based on cycle rule. Procedure 1 outlines the first phase of the *interleaved* algorithm.

Procedure 1:

/* This procedure uses two hash-trees. *Itemset-hashtree which* contains candidates of size k, their potential cycles, and to store support counts for the relevant time units. An "active" itemset at time unit t is an itemset that has a cycle that t participates in. *tmphash-tree*, during the processing of time segment t, contains all the itemsets that are active in t. */

Initially, itemset-hash-tree contains singleton itemsets and all possible cycles
Begin
k = 1
While (for more candidates)
t=0
Do while t<=n-1
insert active itemsets from itemset-Hash-tree into tmp-hash-tree
Foreachnext l belongs to tmp-hash-tree
if ($sup_i < sup_{min}$)
then delete corresponding itemset

else insert (l, supl, t) into itemset-hash-tree
end foreachnext
empty tmp-hash-tree
t=t+1
Enddo
generate itemset-hash-tree for new candidates of size k + 1
k = k + 1
empty itemset-hash-tree
insert new candidates into itemset-
Hash-tree
End while
End

The interleaved algorithm has two distinct phases. In the first phase, all large itemsets with cycles have their supports counted in the appropriate time segments. In the second phase, rules are generated using the cycle and support information of the large itemsets. For the first phase, the interleaved algorithm proceeds "level-by-level" to determine itemset support. It first determines the itemset support for singleton candidate itemsets, generates cycles for them, then generates itemsets of size 2 and their potential cycles, etc. In this phase, the interleaved algorithm requires enough memory to hold all large itemsets of a particular size and their support counts in memory. In addition, it needs to store the new candidates and their "potential" cycles. The size of the latter is usually much smaller. If there is not enough memory, the support counts are broken up into chunks and written to disk.

In the Second Procedure, if there is space in memory to hold all the large itemsets and their support counts for all the time units, rule generation can run entirely in memory. If there is a shortage of memory and there is space to hold only a single level of large itemsets and their support counts, we can generate rules in a level-by-level fashion as well starting at the level of the largest itemset. For doing this, we can use a modification of the procedure 2 that we call *RuleCycles* as shown in procedure 2. It is a set oriented rule generation procedure that creates rules such that all rules needing to look at the support counts of k-itemsets are generated during one iteration of the outer while loop.

Procedure RuleCycles()
Level = size of the largest itemset
/* *ruleList* is a list of records that have four fields. The first three fields contain an itemset name, to support array for the itemset, and the list of cycles for the itemset. The fourth field is a list of candidate rules generated from the itemset that are known to have cycles. */

/* rules list is the current list
RuleList = fg
while (level _ 1)
Read in support counts and cycles
of large itemsets of size level
newRuleList = fg

foreachnext lItem 2 ruleList **do**
    if (lItem's itemset is being used to
        generate rules for the first time)
        then generate singleton rules
    else generate candidate rules using
      apriori-gen on lItem's current rules
verify cycles for each rule generated and
 discard rules without cycles.
lItemNew= lItem with old rules replaced
 by the new rules
newRuleList = newRuleList + lItemNew
endforeachnext
ruleList = newRuleList + quintuplets
 created from large itemsets at current
  level
endwhile

For example, suppose we have the large itemset ABCD with a cycle 001. Initially, *rule List* is the empty set and in the first iteration gets set to ABCD, sup-array$_{ABCD}$, In the next round, ABC to D, ABD to C, ACD to B and BCD to A are generated and tested. This round keeps the support counts of all 3-itemsets in memory. Suppose only ABC to D and ABD to C have cycles. *ruleList* becomes ((ABCD, sup-array$_{ABCD}$, {001}, (ABC ->D, and ABD -> C))). In the next round, only the rule AB-> CD is generated. This round keeps the support counts of all the 2-itemsets in memory. If AB to CD has a cycle, *ruleList* is transformed to (ABCD, sup-array$_{ABCD}$, 001, AB -> CD) and vanishes in the next round. This algorithm requires only one member of *ruleList* to be in memory at any time. If the support counts of a particular level do not fit into memory as well, one has to sort *ruleList* according to the candidate rules that it stores and merge the itemset support.

5 Experimental Results

The results of an extensive set of experiments conducted to analyze the behavior of the sequential and Transition algorithms. In these experiments, it should be kept in mind that the sequential algorithm had an inherent advantage in considering the time segments of the data one by one. All but the largest time segments that we used fit entirely in the main memory of the machine that we used.

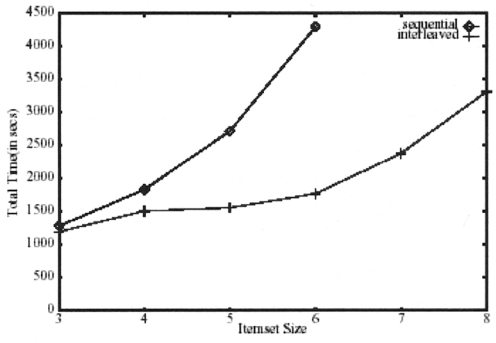

Figure 2: Time vs two algorithms

5.1 Minimum Support and noise

Figure 2 plots the execution time for the interleaved and sequential algorithms as support is varied from 1%to 0:25%. With support set to 1%, the Transition algorithm is only about 5% faster than the sequential algorithm. This is because the number of large itemsets at this level of support is fairly small and the transition algorithm does not incur a significant benefit from its cycle techniques. As support decreases, the amount of wasted work done by the sequential algorithm increases
significantly as the number of itemsets found to be large by the sequential algorithm increases. The interleaved algorithm, from early on, concentrates only on shown. large itemsets that can contribute to cycles and benefits from this strategy.

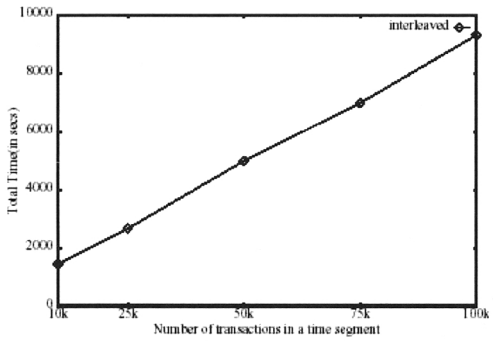

Figure 3: Execution time increases for
the transaction

Figure 3 shows the running times for the transition and sequential algorithms for various levels. Noise influences the data generated in two ways. One, the number of itemsets that do not lead to cycles goes up as the noise level increases. Two, as the amount of noise in the data increases; it leads to spurious cycles, where an itemset is used for data generation in a cyclic fashion. accidentally as dictated by the noise parameter. As can be seen from the graph, the interleaved algorithm is relatively unaffected by noise. (The curves are quite horizontal for supports of 0:33% and 0:25% also.) However, the sequential algorithm shows more interesting behavior. The graphs of the running time increase more or less linearly with noise, but the slope of the graphs is different at different support levels. The explanation for this lies in the number of new itemsets added by noise. At high support levels, noise does not add many itemsets. (If the transaction size is T , large itemset size is I, there can only be about T=(I_sup) large itemsets with support sup. For example, if T = 8, I = 4, sup = 0:5%, there can be no more than 400 large itemsets with support 0:5% in the data.) Therefore, for high support levels, the slope of the sequential algorithm rises slowly. At low support levels, noise tends to have a much more dramatic impact on the amount of wasted work performed by the sequential algorithm and its running time rises with a much steeper slope.

5.5 Conclusions and Future work.

In this paper, we propose a new algorithm cycle rule and transition algorithm for mining association rule to discover calendar based temporal classification which proves

better than the sequential algorithm. The transition algorithm performs at least as well and often times, significantly better than the sequential algorithm. Performance benefits range from 5%, when support is very high, to several hundred percent, when large itemset sizes are over 5. Further, the transition algorithm scales increasing data. Thus the algorithm produces all interesting association rules according to the users needs so one can say that the transition algorithm and the cycle rule gains more performance.

We have studied the problem of discovering association rules that display regular cyclic variation over season Information about such variations will allow marketers to better identify trends in association rules and help better forecasting.

Recently, we have explored the integration of calendars into an association rule mining framework. This lets us consider rules like, "X -> Y , the first working day of every month." we also address the problem of approximate matching, which relaxes the rather rigid definition of when a cycle belongs to an association rule. Interesting open problems include the handling of multiple time units. Integrating cycles and calendars into the *numeric*, rather than the Boolean association rules.

References

1. Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In Proceedings of the 20th International Conference on Very Large Data Bases 1994

2. T. Imielinski R. Agrawal and A. Swami. Mining associations between sets of items in massive databases. In Proceedings of the ACM SIGMOD Int'l
Conference on Management of Data May 1993

3. V. Aranda, J. Calero, G. Delgado, S´anchez D., Serrano J.M., and Vila M.A. Flexible land classification for olive cultivation using user knowledge. In Proceedings of 1st. Int. ICSC Conf. Fuzzy Technologies 2002.

4. R. Agrawal and G. Psaila, "Active Data Mining," in Proc. of the 1st Int'l Conf. on Knowledge Discovery and Data Mining, Montreal, 1995.

5. B. Liu, W. Hsu, and Y. Ma, "Integrating Classification and Association Rule Mining," in Proc. of the 4th Int'l Conf. on Knowledge Discovery and Data Mining, 1998.

6. B. Liu, Y. Ma, and R. Lee, "Analyzing the Interestingness of Association Rules from the Temporal Dimension," in Proc. of the 1st IEEE Int'l Conf. on Data Mining, 2001.

7. H. Mannila, H. Toivonen, and A.I. Verkamo, "Efficient Algorithms for Discovering Association Rules," in Proc. of the AAAI Workshop on Knowledge Discovery in Databases, 1994,

8. R. Srikant and R. Agrawal, "Mining Quantitative Association Rules in Large Relational Tables," in Proc. of the ACM SIGMOD Int'l Conf. on Management of Data,1996,

9. J. Han and Y. Fu. "Discovery of Multi-level Association Rules From Large Databases". In Proceedings of the 21st International Conference on Very Large Data Bases, September 1995.

10. J. S. Park, M. Chen, and P. Yu. "An Effective Hashbased Algorithm for Mining Association Rules". In Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, May 1995.

11. S. Ramaswamy, S. Mahajan, and A. Silberschatz. Calendric association rules. Technical report, Bell Labs, 1997. Submitted for publication.

12. R. Srikant andR.Agrawal. "Mining Generalized Association Rules". In Proceedings of the 21st International Conference on Very Large Data Bases, September 1995

13. H. Toivonen. "Sampling Large Databases for Association Rules". In Proceedings of the 22nd International
Conference on Very Large Data Bases,
September 1996.