# An Effective Boolean Algorithm for Mining Association Rules in Large Databases

Suh-Ying Wur and Yungho Leu
Department of Information Management
National Taiwan University of Science and Technology
{yhl,tammy}@cs.ntust.edu.tw

## Abstract

*In this paper, we present an effective Boolean algorithm for mining association rules in large databases of sales transactions. Like the Apriori algorithm, the proposed Boolean algorithm mines association rules in two steps. In the first step, logic OR and AND operations are used to compute frequent itemsets. In the second step, logic AND and XOR operations are applied to derive all interesting association rules based on the computed frequent itemsets. By only scanning the database once and avoiding generating candidate itemsets in computing frequent itemsets, the Boolean algorithm gains a significant performance improvement over the Apriori algorithm. We propose two efficient implementations of the Boolean algorithm, the BitStream approach and the Sparse-Matrix approach. Through comprehensive experiments, we show that both the BitStream approach and the Sparse-Martrix approach outperform the Apriori algorithm in all database settings. Especially, the Sparse-Matrix approach shows a very significant performance improvement over the Apriori algorithm.*

## 1. Introduction

Due to the rapid growth in the size and number of databases, there is a great need for discovering knowledge hidden in large databases. Knowledge discovery in databases is also known as data mining. Through data mining, we can find useful patterns and rules from databases. These patterns and rules are very useful for decision making of an organization. Therefore, data mining has gained a lot of attentions recently. Specialists from different areas, including machine learning, statistics, artificial intelligence, and expert systems, have developed many powerful tools for data mining.

As stated in [5], many kinds of knowledge can be mined from a database. Among them, the association rule is a very useful knowledge to be mined. The definition of an association rule is described in [3] and,for convenience, is restated in the following. Let I=$\{i_1, i_2, \cdots, i_n\}$ be a set of items. Given a set of sales transactions $D$, where each transaction $T$ is a subset of $I$, an association rule is an expression of the form $X \rightarrow Y$, where $X$ and $Y$ are subsets of $I$ and $X$ ∩Y=$\phi$. An association rule $X \rightarrow Y$ holds in the transaction set $D$ with a confidence $c$ if $c\%$ of the transactions that contain $X$ also contain $Y$. An association rule $X \rightarrow Y$ has a support $s$ if $s\%$ of the transactions in $D$ contain both $X$ and $Y$. The task of mining association rules is to find all the association rules which satisfy both the user-defined minimum support and minimum confidence.

A set containing $k$ items is called a $k - itemset$. A $k - itemset$ is called a frequent $k - itemsets$ if given a minimum support $s$ and a transaction set $D$, at least $s\%$ of the transactions in $D$ contain the $k - itemset$. The process of mining association rules can be decomposed into two steps. First, all the frequent itemsets are identified. Then, the association rules satisfying both the minimum support and the minimum confidence are identified based on the frequent itemsets computed in the first step. A lot of literature, such as [1, 2, 3, 4, 6, 7, 8, 9] have pointed out that, due to the huge amount of data in a database, the process of generating frequent itemsets turns out to be the bottleneck in mining association rules. Therefore, researchers have focused on developing efficient and effective algorithms for the generation of frequent itemsets.

In this paper, we present a Boolean algorithm for efficiently mining association rules. In the proposed algorithm, we first construct an item table and a transaction table by scanning the database once. Then, we repeatedly apply OR and AND operations on the item table and the transaction table respectively to generate the frequent itemsets. Since both OR and AND operations can be efficiently implemented, the Boolean algorithm is very efficient in generating frequent itemsets. Through comprehensive experiments, we show that the Boolean algorithm outperforms the

Aprior algorithms in all the test cases.

This paper is organized as follows. In Section 2, we review the related work. Section 3 is devoted to the details of the Boolean algorithm. In Section 4, we show the experiment results and compare the performance of the Boolean algorithm with that of the Apriori algorithm. Finally, we conclude this paper and give a word about the future work in Section 5.

## 2. Related Work

Agrawal et al. Proposed an algorithm, called AIS algorithm [1], for generating frequent itemsets. In the AIS algorithm, frequent itemsets are generated through iterations on scanning the database. The iteration terminates when no new frequent itemset is derived. After reading a transaction in the $k^{th}$ iteration, the AIS algorithm computes the candidate $k - itemsets$ by first deriving a set of $(k-1) - itemsets$ which contains itemsets that are both in the frequent $(k-1)-itemsets$ and in the transaction. Then, the AIS algorithm extends the derived $(k-1) - itemsets$ with other items in the same transaction. A candidate $k - itemset$ computed from reading a transaction is added to the set of candidate $k - itemsets$ for the $k^{th}$ iteration if it is not already contained in the set of all candidate $k - itemsets$, or the count of the corresponding candidate itemset is increased by one if it is already generated by an earlier transaction. At the end of the iteration, only those candidate $k - itemsets$ that have a sufficient count value are considered as the frequent $k - itemsets$. One disadvantage of the AIS algorithm is that it generates too many invalid candidate itemsets.

Houtsma and Swami proposed the SETM algorithm [7] that uses SQL for generating the frequent itemsets. Although it uses standard SQL join operation for generating candidate itemsets, the SETM algorithm generates candidate itemsets through a process of iterations similar to that of the AIS algorithm. The disadvantage of the SETM algorithm is similar to that of the AIS algorithm. That is, it generates too many invalid candidate itemsets.

Agrawal and Srikant also proposed two fast algorithms, called Apriori and AprioriTid [3], for generating frequent itemsets. In the Apriori algorithm, the candidate $k - itemsets$ is generated by a cross product of the frequent $(k-1)-itemsets$ with itself. Then, the database is scanned for computing the count of the candidate $k - itemsets$. The frequent $k - itemsets$ consist of only the candidate $k - itemsets$ with sufficient support. This process is repeated until no new candidate itemsets is generated. It is noted that in the Apriori algorithm, each iteration requires a pass of scanning the database, which incurs a severe performance penalty. In order to reduce the number of scanning of the database in the Apriori algorithm, the authors proposed an alternative for fast mining association rules in [3], called AprioriTid. Unlike the Apriori algorithm, the AprioriTid algorithm scans the database only once in the first iteration. Although the AprioriTid algorithm reduces the number of scanning of the database, its performance is inferior to that of the Apriori algorithm. However, it shows that both the Apriori algorithm and the AprioriTid algorithm are superior to the AIS and SETM algorithms [3].

Park et al. proposed the DHP (standing for Direct Hashing and Pruning) algorithm [8] for efficient generation of frequent itemsets. The DHP is a hash-based algorithm and is especially effective for the generation of frequent $2 - itemsets$. Based on the Apriori algorithm, the DHP algorithm uses an efficient approach to trim the number of candidate $2 - itemsets$. As a result, the number of candidate $2 - itemsets$ generated by the DHP algorithm is much smaller than those generated by the Apriori and the AprioriTid algorithms [8].

Yen and Chen proposed a DLG algorithm [11] and an EDM algorithm for discovering association rules. In the EDM algorithm [12], inner-product operations are performed for counting the support of an itemset. While in the DLG algorithm, logic AND operations are used to count the support. Both algorithms avoid generating candidate itemsets, which is a very time consuming operation. Roberto proposed a Max-Miner algorithm [10] for efficiently identifying long frequent itemsets which, in turn, can be used to generate other frequent itemsets.

**Table 1. The truth table for AND, OR, and XOR operations**

| $V_1$ | 1 | 1 | 0 | 0 |
|---|---|---|---|---|
| $V_2$ | 1 | 0 | 1 | 0 |
| $V_l$ AND $V_2$ | 1 | 0 | 0 | 0 |
| $V_1$ OR $V_2$ | 1 | 1 | 1 | 0 |
| $V_1$ XOR $V_2$ | 0 | 1 | 1 | 0 |

## 3. The Boolean Algorithm

The Boolean algorithm mines association rules in two steps. In the first step, the frequent itemsets is identified. Then in the second step, the association rules based on the identified frequent itemsets are generated. Because the Boolean algorithm is based on AND, OR and XOR logical operations, we show, for convenience, the truth table of these operations in Table 1.

In the following, we first present the generation of frequent itemsets in section 3.1. Then, we describe the generation of association rules in section 3.2. For better understanding of the following discussions, we use Example 1 as

Database $D$

| TID | Items |
|---|---|
| 100 | ACD |
| 200 | BCE |
| 300 | ABCE |
| 400 | BE |

Item Set $I$

| ABCDE |
|---|

Minimum support: $40\%$
Minimum Confidence: $50\%$

**Figure 1. A sample database for data mining**

a sample database.

Example 1: Figure 1 shows a database $D$ with four sales transactions; each transaction is a subset of the item set $I$ which contains five items; the minimum support and the minimum confidence are $40\%$ and $50\%$ respectively.

## 3.1. Generation of Frequent Itemsets

The Boolean algorithm generates frequent itemsets through several iterations based on two tables, the item table and the transaction table. Section 3.1.1 describes the construction of an initial item table and an initial transaction table. Then, Section 3.1.2 describes the process of generating frequent $k - itemsets$.

### 3.1.1 Initializing Item Table and Transaction Table

The item table in the $k$th iteration ($TI_k$) is a $p \times n$ table, where $p$ is the number of the frequent $k - itemsets$ and $n$ is the number of items in $I$. Each column in $TI_k$ represents a data item in item set $I$,while each row in $TI_k$ represents a frequent $k - itemset$. An initial item table $TI_1$, is an $n \times n$ identity matrix, where $n$ is the number of items in $I$.

A transaction table in the $k$th iteration ($TT_k$) is a $p \times m$ table, where $p$ is the number of frequent $k - itemsets$ and $m$ is the number of transactions in $D$. Each column in $TT_k$ represents a transaction in $D$. Each row in $TT_k$ records the transactions that contain the corresponding itemset. $TT_k$ $[i, j]$ takes on value 1 if the $j$th transaction in $TT_k$ contains the $i$th frequent $k - itemset$ in $TI_k$,; otherwise, $TT_k[i, j]$ takes on value 0. We need to scan database $D$ once in order to construct the initial transaction table $TT_1$.

Finally, we use a column vector $C_k$ to store the reference count of all frequent $k - itemsets$ in the $k$th iteration. The reference count on a $k - itemset$ can be obtained by counting the number of $l$s in the corresponding row of $TT_k$. Hereafter, we refer to the concatenation (by put them together side by side) of the item table ($TI$), transaction table ($TT$), and the reference count $C$ as the

**Table 2. Notations**

| | |
|---|---|
| $TI_k$ | Item table at the $k$th iteration |
| $TT_k$ | Transaction table at the $k$th iteration |
| $C_k$ | Column vector of reference count in the $k$th iteration |
| $TIC_k$ | Concatenation of $TI_k$ and $C_k$ |
| $TITTC_k$ | Concatenation of $TI_k$, $TT_k$, and $C_k$ |
| $TIC$ | The union of all $TIC_k$, $k$= 1, $\cdots$, to the last iteration |
| $TAR$ | Table of association rules |

item/transaction/count table (abbreviated as $TITTC$). In the Boolean algorithm, we use $TITTC_k$ to denote the corresponding item/transaction/count table of the algorithm in the $k$th iteration. The notations used in the Boolean algorithm are shown in Table 2. In Table 2, the $TIC$ is the union (by treating each row in $TIC_k$ as an element and the $TIC_k$ as a set, for any $k$) of all $TIC_k$. The $TAR$ table will contain the derived association rules after the mining process completes.

Figure 2(a) shows the initial tables for example 1, which consist of $TI_1$, $TT_1$, $C_1$ and their concatenation $TITTC_1$. According to example 1, the minimum number of transactions needed for a frequent itemset is $40\% * 5$, which equals to 2. Based on the initial $TITTC_1$, we prune the row corresponding to itemset $\{D\}$ whose reference count does not satisfy the minimum support requirement. $TITTC_1$ that contains only frequent $1 - itemsets$ is shown in Figure 2(b). After generation of frequent $1 - itemsets$, the concatenation of the item table and the count vector (i.e., $TIC_1$,), as shown in Figure 2(c), is retained in $TIC$ for the generation of association rules.

The detailed algorithm for generating frequent $1 - itemsets$ is shown in Figure 3.

### 3.1.2 Generation of Frequent $k - itemsets$

Frequent $k - itemsets$ can be generated through the following iteration:

Repeat

1. Get a pair of different rows ($itemsets$) in $TI_{k-1}$.
2. Apply OR operation on these two rows to get a new temporary itemset. If the temporary itemset contains more than $k$ different items or is already produced by a previous OR operation, go to step l (i.e., ignore this new itemset); otherwise, go to step 3 (i.e., a new $k - itemset$ has been found).
3. Apply AND operation on the two rows of $TT_{k-1}$ which correspond to the rows of step 2. The result shows which transactions contain this new $k -$

(a) The Initial $TITTC$ Table

| | A | B | C | D | E | T100 | T200 | T300 | T400 | Count |
|---|---|---|---|---|---|---|---|---|---|---|
| A | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 2 |
| B | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 3 |
| C | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 3 |
| D | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| E | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 3 |

(b) The $TITTC_1$ table

| | A | B | C | D | E | T100 | T200 | T300 | T400 | Count |
|---|---|---|---|---|---|---|---|---|---|---|
| A | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 2 |
| B | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 3 |
| C | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 3 |
| E | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 3 |

(c) The $TIC_1$ Table

| | A | B | C | D | E | count |
|---|---|---|---|---|---|---|
| A | 1 | 0 | 0 | 0 | 0 | 2 |
| B | 0 | 1 | 0 | 0 | 0 | 3 |
| C | 0 | 0 | 1 | 0 | 0 | 3 |
| E | 0 | 0 | 0 | 0 | 1 | 3 |

**Figure 2. Generation of frequent $1-itemsets$**

```
// Generating initial $TITTC_1$, and $TIC_1$
// Suppose there are $n$ items in $I$ and $m$ transactions in $D$

Initialize $TI_1$, by setting $TI_1[i,j] = 1$ if $i = j$, otherwise
   $TI_1[i,j] = 0$;
Initialize $TT_1$, by setting $TT_1[i,j] = 0$, for $i = 1, \cdots, n$,
   and $j = 1, \cdots, m$;
Initialize $C_1$, by setting $C_1[i] = 0$ for $i = 1, \cdots, n$;
for( $i = 1; i <= m; ++i$) do begin
    read the $i$th transaction from the database;
        for ( $j = 1; j <= n; ++j$) do begin
            if (the $i$th transaction contains the $j$th item)
              $TT_1[j,i] = 1$;
end
for( $i = 1; i <= n; ++i$) do begin
    for ($j = 1; j <= m; ++j$) do begin
        $C_1[i] = C_1[i] + TT_1[i,j]$;
    end
end
for( $i = 1; i <= n; ++i$) do begin
    if ($C_1[i] <$ min_support $\times$ number of total transactions
    in the database) then do begin
        eliminate row $TI_1[i]$ and $TT_1[i]$;
        discard $C_1[i]$;
    end
end
concatenate $TI_1$, $TT_1$, and $C_1$, to form $TITTC_1$ for next frequent $2-$
$itemsets$ generation;
concatenate $TI_1$, and $C_1$, to form $TIC_1$ for association rules generation;
```

**Figure 3. Algorithm for generating initial $TITTC_1$, and $TIC_1$**

```
// Algorithm for generation of frequent $k-itemsets$
// Suppose, after the iteration on computing $(k-l)-itemsets$, $TI_{k-1}$
// has $p$ rows and $n$ columns; $TT_{k-1}$ has $p$ rows and $q$ columns; count vector
// $C$ has $p$ elements.

if ($p = 1$) then stop   else $y = 1$;
for( $i = 1; i <= (p-1); ++i$) do begin
    for( $x = i+1; x <= p; ++x$) do begin
        $I\_count = 0$;
        for( $j = 1; j <= n; ++j$) do begin
            $TI_k[y,j] = TI_{k-1}[i,j]$  OR  $TI_{k-1}[x,j]$;
            $I\_count = I\_Count + TI_k[y,j]$;
        end
        found = 0;
        for( $z = 1; z < y$ and I_count $= k$ and found $= 0; ++z$) do begin
            if( row vector $TI_k[z] =$ row vector $TI_k[y]$) then found $= 1$;
        end
        $T\_count = 0$;
        if( found $= 0$ and $I\_count = k$) then do begin

            for( $j = 1; j <= q; ++q$) do begin
              $TT_k[y,j] = TT_{k-1}[i,j]$ AND $TT_{k-1}[x,j]$;
              $T\_count = T\_count + TT_k[y,j]$;
            end
        end
        if ($T\_count >=$ min_support $*$ number of transactions in the database)
            then do begin
              $C_k[y] = T\_count$;
              y=y+l;
            end
    end
end
concatenate $TI_k$, $TT_k$, and $C_k$ to form $TITTC_k$ for next frequent $(k+l)-$
$itemsets$ generation;
concatenate $TI_k$ and $C_k$ to form $TIC_k$ for generating association rules;
```

**Figure 4. Algorithm for generation of frequent $k-itemsets$**

$itemset$. We then count the number of 1s in the result to get the reference count of this new $k-itemset$. If the count is less than the number of transactions required by the minimum support, the new $k-itemset$ is discarded; Otherwise, the new k-itemset, the AND result and the reference count are inserted into $TI_k$, $TT_k$ and $C_k$ respectively.

Until (no new pair of rows in $TI_{k-1}$ left without being processed)

After the generation of frequent $k-itemsets$, the item table of the $k-itemsets$ and its corresponding reference count vector are kept in $TIC$ for generating association rules. Figure 4 shows the algorithm for generation of frequent $k-itemsets$. In Figure 5, we use an example to illustrate this algorithm.

Continued from example 1, by performing OR and AND operations on $TI_1$, $TT_1$, (in Figure 2(b)) respectively, we derive table $TITTC_2$ in Figure 5(a). It is noted that the itemset $\{$AE$\}$ is not in table $TITTC_2$ because it fails to satisfy the minimum support requirement. Similarly, we derive table $TITTC_3$ from $TITTC_2$, which contains only one frequent $3-itemset$, as is shown in Figure 5(d).

After the iteration completes, the final $TIC$, which is needed for generation of the association rules, is shown in

(a) The $TITTC_2$ Table

| | A | B | C | D | E | T100 | T200 | T300 | T400 | Count |
|---|---|---|---|---|---|---|---|---|---|---|
| AC | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 2 |
| BC | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 2 |
| BE | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 3 |
| CE | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 2 |

(b) The $TIC_2$ Table

| | A | B | C | D | E | Count |
|---|---|---|---|---|---|---|
| AC | 1 | 0 | 1 | 0 | 0 | 2 |
| BC | 0 | 1 | 1 | 0 | 0 | 2 |
| BE | 0 | 1 | 0 | 0 | 1 | 3 |
| CE | 0 | 0 | 1 | 0 | 1 | 2 |

(c) The $TITTC_3$ Table

| | A | B | C | D | E | T100 | T200 | T300 | T400 | Count |
|---|---|---|---|---|---|---|---|---|---|---|
| BCE | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 2 |

(d) The $TIC_3$ Table

| | A | B | C | D | E | Count |
|---|---|---|---|---|---|---|
| BCE | 0 | 1 | 1 | 0 | 1 | 2 |

**Figure 5. Example for generating frequent $k-itemsets$**

| | A | B | C | D | E | Count |
|---|---|---|---|---|---|---|
| A | 1 | 0 | 0 | 0 | 0 | 2 |
| B | 0 | 1 | 0 | 0 | 0 | 3 |
| C | 0 | 0 | 1 | 0 | 0 | 3 |
| E | 0 | 0 | 0 | 0 | 1 | 3 |
| AC | 1 | 0 | 1 | 0 | 0 | 2 |
| BC | 0 | 1 | 1 | 0 | 0 | 2 |
| BE | 0 | 1 | 0 | 0 | 1 | 3 |
| CE | 0 | 0 | 1 | 0 | 1 | 2 |
| BCE | 0 | 1 | 1 | 0 | 1 | 2 |

**Figure 6. The final $TIC$ table**

Figure 6.

## 3.2. Generation of Association Rules

In this section, we present the way that Boolean algorithm mines association rules from the final frequent itemsets table.

As discussed in Section 1, if the association rule $X \rightarrow Y$ holds, then all $X$, $Y$ and $X \cup Y$ must be frequent itemsets. Since $X \cup Y$ contains both $X$ and $Y$, we can infer that if a frequent itemset is not a subset of any another frequent itemset in $TIC$ then it can be neither an antecedent nor a consequent of any association rule. This observation is the foundation of our Boolean algorithm in mining association rules. To expedite the mining process, we can first eliminate, from the $TIC$ table, those frequent itemsets that are not subset of any other frequent itemsets. However, we have found that only frequent $1-itemsets$ are candidates for the elimination.

Based on the above-mentioned observation, the Boolean algorithm mines an association rule by first identifying the potential antecedent and potential consequent, and then validate if such a rule satisfies the minimum confidence requirement. The basic ideas for mining association rules are described in the following.

1. Eliminate the rows of $1-itemsets$ from $TIC$ which have no opportunity to be an antecedent or a consequent of any association rules. The Boolean algorithm simply counts the occurrences of 1s in each column of $TI$. A $1-itemset$ whose corresponding column has only one $1$ in the whole column should be eliminated.

2. Apply an AND operation on two rows, say $X$ and $Z$, of $TI$ in $TIC$, and then compare the result with the one which has less number of items, assuming it is $X$. If they are equal, then the frequent itemset with less number of items(i.e., $X$ in this case) is a potential antecedent.

3. Apply an XOR operation on the two rows chosen in step $2$. The result, denoted as $Y$, of the XOR operation constitutes a potential consequent with $X$ being its corresponding antecedent.

4. If support($Z$)/support($X$) is greater than or equal to the minimum confidence, then the association rule $X \rightarrow Y$ is generated.

Repeat step $2$ through step $4$ for any combination of $X$ and $Z$ until no new rules is found. We create a table of association rules, called $TAR$, to store the antecedent $X$, consequent $Y$, support and confidence for each association rule $X \rightarrow Y$. Figure 7 shows the resultant TAR for example 1, and Figure 8 shows the detailed algorithm for the generation of association rules.

Take the third row of $TAR$, which read as $\{B\} \rightarrow \{E\}$, as an example. It is derived from row B and row BE in the $TIC$ table of Figure 6. By performing bit-wised AND operation on the row $\{B\}$ and the row $\{BE\}$, we get a binary vector $(01000)$, which is exactly the same as that of row $\{B\}$. Therefore, $\{B\}$ is an antecedent. By performing a bit-wised XOR operation on row $\{BE\}$ and row $\{B\}$, we get itemset $\{E\}$ as the corresponding consequent. A rule of $\{B\} \rightarrow \{E\}$ is therefore discovered. By checking the counts of $\{BE\}$ and $\{B\}$ in $TIC$ and performing the necessary computations, we get the support and the confidence of this rule, which is $75\%$ and $100\%$ respectively.

## 4. Experiments and Results

To evaluate the performance of the Boolean algorithm, we perform several experiments on Sun SPARC 20 workstation with CPU clock rate 70 MHz, 32 MB of main memory,

| Antecedent | Consequent | Support | Confidence |
|---|---|---|---|
| A | C | 50% | 100% |
| B | C | 50% | 67% |
| B | E | 75% | 100% |
| B | CE | 50% | 67% |
| C | A | 50% | 67% |
| C | B | 50% | 67% |
| C | E | 50% | 67% |
| C | BE | 50% | 67% |
| E | B | 75% | 100% |
| E | C | 50% | 67% |
| E | BC | 50% | 67% |
| BC | E | 50% | 100% |
| BE | C | 50% | 67% |
| CE | B | 50% | 100% |

**Figure 7. $TAR$ for example 1**

and running SUNOS 4.1.3_Ul. Data resided on a SPARC file system.

We first describe two implementations of the Boolean algorithm in Section 4.1. Then, we describe the method for generating synthetic data for experiments in section 4.2. The performance comparisons are given in section 4.3.

## 4.1. Implementations of the Boolean Algorithm

It is noted that, although, we use tables as data structures in the Boolean algorithm, these data structures can actually be implemented more efficiently using bit-wised vectors or sparse matrix technique. In the following, we present two implementations for the Boolean algorithm, called the BitStream approach and the Sparse-Matrix approach. We use C as the implementation language.

### 4.1.1 The BitStream Approach

Since each entry of both item table and transaction table can take only 1 or 0 as its value, we use one bit to represent an entry in both the item table and the transaction table. This approach significantly reduces the memory size and computation time for AND/OR operations compared with those of the pure table approach.

### 4.1.2 The Sparse-Matrix Approach

In general, an itemset will contain only a small number of items, compared to the number of items in item set $I$. Also, an itemset is usually related to a small number of transactions compared to the large number of transactions in the transaction set $D$. As a result, an item table ($TI_k$) or a transaction table ($TT_k$) are very sparse. We therefore use sparse matrices to implement all item tables and transaction tables. Using this approach, only non-zero entries in the tables need to be considered.

```
// Generation of association rules
// Suppose that, after the generation of frequent itemsets, we have $TIC_1$, $TIC_2$,
//$\cdots$, $TIC_k$ tables, and each table has $num1$, $num2$, $\cdots$, $numk$
//number of rows respectively.
// Each table has $(n+1)$ columns, containing $n$ items and the count vector.
// $Num$ is a column vector which stores the number of rows contained in $TIC_1$,
// $TIC_2$, $\cdots$, and $TIC_k$. That is, $Num[1] = num1$, $Num[2] = num2$,
// $\cdots$, and $Num[k] = numk$.

Perform a UNION operation on $TIC_1, TIC_2, \cdots, TIC_k$ to form $TIC$;
$rows = 0$;
for( $i = 1, i <= k; ++i$) do $rows = rows + Num[k]$;
for( $j = 1; j <= n; ++j$) do begin
    $occurrence = 0$;
    for( $i = 1; i <= rows; ++i$) do $occurrence = occurrence +$
        $TIC[i, j]$;
    if( $occurrence = 1$) then do begin
        eliminate the row of $TIC$ that consists of the $j$th item;
        $rows = rows - 1$;
        $Num[1] = Num[1] - 1$;
    end
end
$p = 1$;
for( $i = 1; i <= k; ++i$) do begin
    $row1 = 1$;
    for($j = 1; j <= i-1; ++j$) do $row1 = row1 + Num[j]$;
    for( $row2 = row1 + Num[i]; row2 <= rows; ++row2$) do begin
        for($j = 1; j <= n; ++j$) do $temp[j] = TIC[row1, j]$ AND
            $TIC[row2, j]$;
        if( row vector of temp = row vector $TIC[row1]$ and
            $TIC[row2, n+1] / TIC[row1, n+1] >=$ min-confidence) then
            do begin
              Store the itemset in $TIC[rowl]$, as the antecedent of the rule,
                in $TAR[p]$;
              for($j = 1; j <= n; ++j$) do $tempj] = TIC[row1, j]$ XOR
                $TIC[row2, j]$;
              Store the itemset in $temp[j]$, as the consequent of the rule, in
                $TAR[p]$;
              Store $TIC[row2, n+1]$ / (number transactions in the database),
                as the support of the rule, in $TAR[p]$;
              Store $TIC[row2, n+1]/TIC[row1, n+1]$, as the confidence
                of the rule, in $TAR[p]$;
              $p = p + 1$;
            end
    end
end
```

**Figure 8. Algorithm for generation of association rules**

## 4.2. Generation of Synthetic Data

We use extensive synthetic data to evaluate our algorithms. The way in generating synthetic data is similar to that of [3]. The definitions of various parameters used in our experiments are summarized in Table 3. We generated six sales transaction databases by setting $N$ = 100, $|D|$ = 600, and $|L|$ = 2000. We chose 3 different values for $|T|$, which are 5, 10, and 20. We also chose 3 different values for $|I|$, which are 2, 4, and 6. Table 4 summarizes the database parameter settings.

## 4.3. Performance Comparisons

We tested the BitStream, Sparse-Matrix, and Apriori approaches over $T5I2$, $T10I2$, $T10I4$, $T20I2$, $T20I4$, and $T20I6$ synthetic databases with minimum supports ranging

**Table 3. Definition of Parameters**

| | |
|---|---|
| $\lvert D\rvert$ | Number of transactions |
| $\lvert T\rvert$ | Average size of the transactions |
| $\lvert I\rvert$ | Average size of the potentially maximal frequent itemsets |
| $\lvert L\rvert$ | Number of potentially maximal frequent itemsets |
| $N$ | Number of items |

**Table 4. Parameter settings**

| Name | $\lvert T\rvert$ | $\lvert I\rvert$ | $\lvert D\rvert$ |
|---|---|---|---|
| $T5.I2.D600$ | 5 | 2 | 600 |
| $Tl0.I2.D600$ | 10 | 2 | 600 |
| $TI0.I4.D600$ | 10 | 4 | 600 |
| $T20.I2.D600$ | 20 | 2 | 600 |
| $T20.I4.D600$ | 20 | 4 | 600 |
| $T20.I6.D600$ | 20 | 6 | 600 |

from 1% to 2%.

Figure 9 shows that the Sparse-Matrix approach is superior to the other two approaches. This is due to the fact that the Sparse-Matrix approach takes extremely less amount of time in generating frequent itemsets.

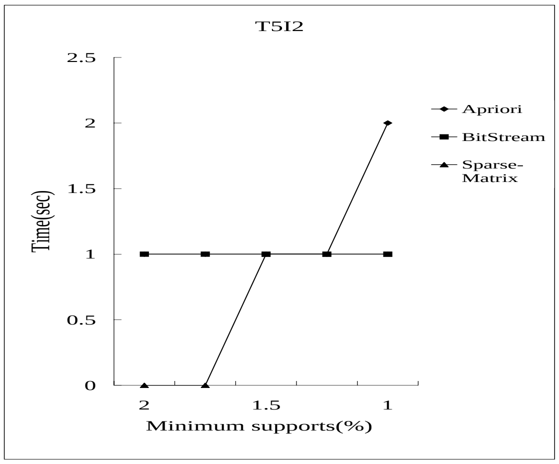
## 5. Conclusions and Future work

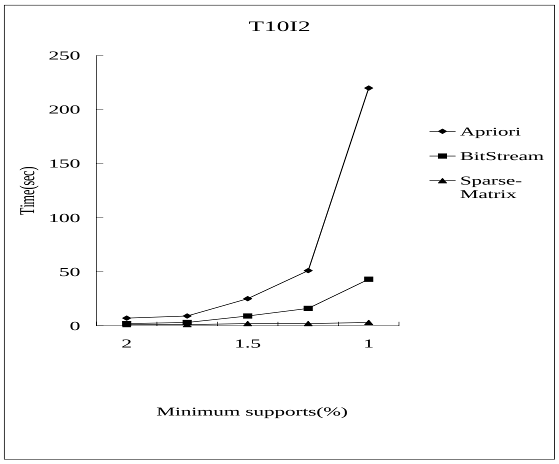
We proposed an effective Boolean algorithm for mining association rules in large sales transaction databases. The major advantage of the Boolean algorithm over the Apriori algorithm is that the Boolean algorithm generates frequent itemsets without constructing candidate itemsets. In contrast, construction of candidate itemsets is required by the Apriori algorithm.

We also presented two efficient implementations for the Boolean algorithm, the BitStream approach and the Sparse-Matrix approach. We conduct several experiments using different synthetic databases. The results show that both the BitStream approach and the Sparse-Matrix approach outperform the Apriori approach in all database settings. Especially, the Sparse-Matrix approach shows a significant performance improvement over that of the Apriori approach.

In the future, we plan to extend this work along the following directions:

1. Utilize the parallel system to efficiently generate frequent itemsets.
2. Extend the Boolean algorithm for data which exhibit concept hierarchy property.





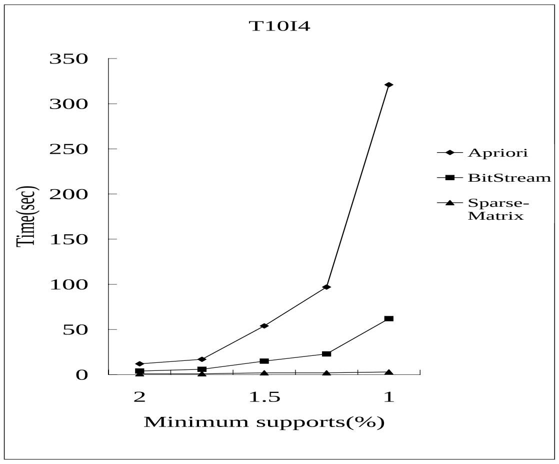
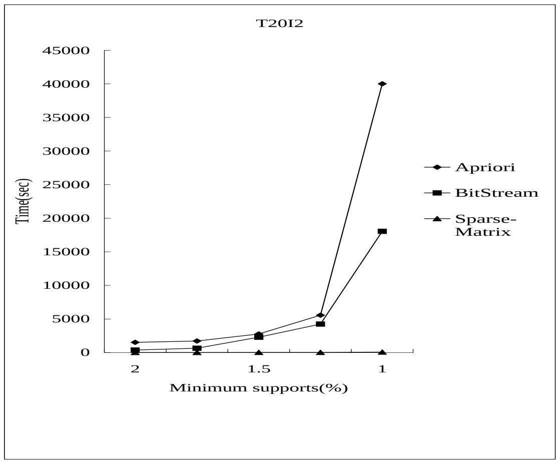
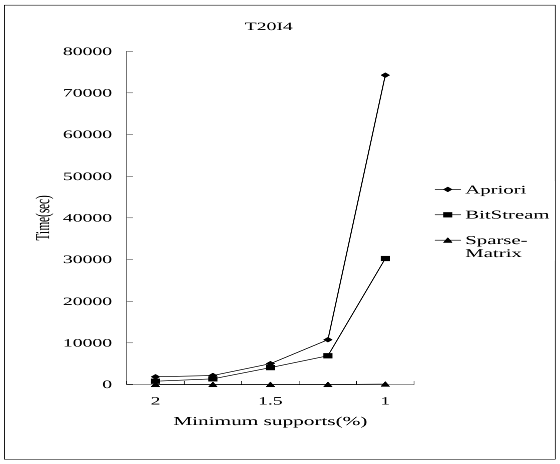


**Figure 9. Execution times**





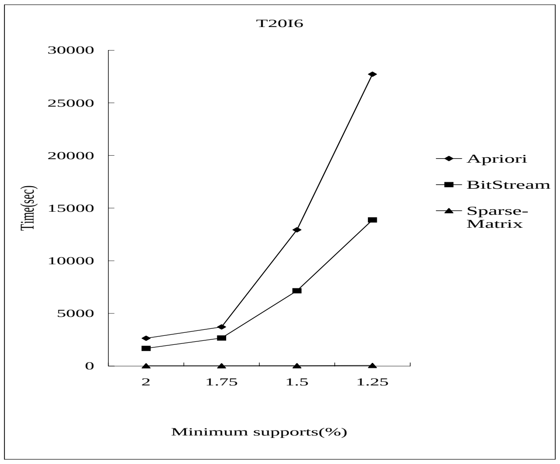


**Figure 9. Execution times(Continued)**

3. Extend the Boolean algorithm for mining association rules in relational databases.

## References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 207–216, May 1993.

[2] R. Agrawal and J. Shafer. Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):962–969, December 1996.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. *Proceedings of the 20th International Conference on Very Large Data Bases*, September 1994.

[4] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. *ACM SIGMOD International Conference on Management of Data*, pages 255–264, May 1997.

[5] M.-S. Chen, J. Han, and P. S. Yu. Data mining: An overview from a database perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):866–882, December 1996.

[6] E.-H. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. *ACM SIGMOD International Conference on Management of Data*, pages 277–288, May 1997.

[7] M. Houtsma and A. Swami. Set-oriented mining for association rules in relational databases. *IEEE 11th International Conference on Data Engineering*, pages 25–33, 1995.

[8] J. S. park, M.-S. Chen, and P. S. Yu. An effective hash based algorithm for mining association rules. *ACM SIGMOD International Conference on Management of Data*, pages 175–186, May 1995.

[9] J. S. Park, M.-S. Chen, and P. S. Yu. Using a hash-based method with transaction trimming for mining association rules. *IEEE Transactions on Knowledge and Data Engineering*, 9(5):813–825, September/October 1997.

[10] B. J. Roberto. Efficiently mining long patterns from databases. *ACM SIGMOD International Conference on Management of Data*, pages 85–93, 1998.

[11] S.-J. Yen and A. Chen. An efficient approach to discovering knowledge from large databases. *Proceedings of the International Conference on Parallel and Distributed Information Systems*, pages 8–18, 1996.

[12] S.-J. Yen and A. Chen. An efficient data mining technique for discovering interesting association rules. *Proceedings of the International Conference and Workshop on Database and Expert System Applications*, pages 664–669, 1997.