Answering the Most Correlated N Association Rules Efficiently

Jun Sese and Shinichi Morishita

Department of Complexity Science and Engineering
Graduate School of Frontier Science, University of Tokyo
{sesejun,moris}@gi.k.u-tokyo.ac.jp

Abstract. Many algorithms have been proposed for computing association rules using the support-confidence framework. One drawback of this framework is its weakness in expressing the notion of correlation. We propose an efficient algorithm for mining association rules that uses statistical metrics to determine correlation. The simple application of conventional techniques developed for the support-confidence framework is not possible, since functions for correlation do not meet the anti-monotonicity property that is crucial to traditional methods. In this paper, we propose the heuristics for the vertical decomposition of a database, for pruning unproductive itemsets, and for traversing a set-enumeration tree of itemsets that is tailored to the calculation of the N most significant association rules, where N can be specified by the user. We experimentally compared the combination of these three techniques with the previous statistical approach. Our tests confirmed that the computational performance improves by several orders of magnitude.

1 Introduction

A great deal of research has examined the analysis of association rules [2]. Most of these studies have proposed efficient algorithms for computing association rules so that both support and confidence are sufficiently high. However, several researchers have remarked that one drawback of the support and confidence framework is its weakness in expressing the notion of correlation [6,1,10,11]. For instance, in practice, the analysis of scientific data calls for a method of discovering correlations among various phenomena, even when the database is noisy. The number of parameters taken into account can sometimes be in the millions. Such cases require an efficient way of selecting combinations of parameters (items) that are highly correlated with the phenomena of interest. For instance, in the human genome, the number of point-wise mutations in human DNA sequences is estimated to be in the millions. However, there is a need to discover combinations of mutations that are strongly correlated with common diseases, even in the presence of large amounts of noise. Such new applications demand fast algorithms for handling large datasets with millions of items, and for mining association rules that produce significant correlations.

T. Elomaa et al. (Eds.): PKDD, LNAI 2431, pp. 410–422, 2002.

Table 1. Transactions and Association Rules

a	b	c	d	e
1	1	1	1	1
1	1	0	1	1
1	0	1	0	1
1	0	0	0	1
0	1	1	0	1
0	1	0	0	1
0	0	1	1	1
0	0	0	1	1

(A) Examples of Transactions

$I \Rightarrow C$	support	confidence	correlated?
$\{x\} \Rightarrow \{y\}$	25%	50%	No
$(x, y \in \{a, b, c, d\}, x \neq y)$			
Many Rules with Singleton Sets			
$\{a, b\} \Rightarrow \{d\}$	25%	100%	Yes
$\{a\} \Rightarrow \{e\}$	50%	100%	No

(B) Examples of Association Rules

In the following, we present an example that illustrates the difference between correlation and the traditional criteria: support and confidence.

Motivating Example. In Table 1(A) taken from [11], each row except the first represents an itemset, and each column denotes an item. "1" indicates the presence of an item in the row, while "0" indicates the absence of an item. For example, the fourth row expresses $\{a, c, e\}$. Let I be an itemset, and let $Pr(I)$ denote the ratio of the number of transactions that include I to the number of all transactions. In our current example, $Pr(\{a, b\}) = 25\%$ and $Pr(\{a, b, c\}) = 12.5\%$. Association rules are in the form $I_1 \Rightarrow I_2$, where I_1 and I_2 are disjoint itemsets.

The *support* for rule $I_1 \Rightarrow I_2$ is the fraction of a transaction that contains both I_1 and I_2, namely, $Pr(I_1 \cup I_2)$. The *confidence* of rule $I_1 \Rightarrow I_2$ is the fraction of a transaction containing I_1 that also contains I_2, namely, $Pr(I_1 \cup I_2 | I_1)$. Table 1(B) shows some association rules derived from the database in Table 1(A). From Table 1(B), we may conclude that rule $\{a\} \Rightarrow \{e\}$ is the most valuable, since both its support and confidence are the highest.

Statistically speaking, the first and third rules do not make sense, because in each rule the assumptive itemset, say I, and the conclusive itemset, say C, are independent; that is, $Pr(I \cup C) = Pr(I) \times Pr(C)$. Conversely, in the second rule, the assumption and conclusion are highly and positively correlated.

This example suggests that the usefulness of association rules should be measured by the significance of the correlation between the assumption and the conclusion. For this purpose, the chi-squared value is typically used, because of its solid grounding in statistics. The larger the chi-squared value, the higher is the correlation.

Related Work. To address this problem, application of the Apriori algorithm has been investigated [6,1,11]. This algorithm proposed by Brin *et al.* [6] enumerates large itemsets first and then selects correlated itemsets based on the chi-squared measure. However, the algorithm does not discard the first non-correlated rule in Table 1(B) because of its use of the support threshold. To avoid the use of a support threshold, Aggarwal and Yu [1] proposed the genera-

tion of a *strongly collective itemset* that requires correlation among the items of any subset. However, the new algorithm discards the second rule in Table 1(B) because the algorithm might be too restrictive to output some desired rules.

To calculate the optimal solution of association rules using common statistical measures, Bayardo and Agrawal [5] presented a method of approaching optimal solutions by scanning what is called a support/confidence border. To further exploit Bayardo's idea by combining it with the traversing itemset lattices developed by Agrawal and Srikant [3], we [11] proposed an algorithm called the AprioriSMP. The novel aspect of AprioriSMP is its ability to estimate a tight upper bound on the statistical metric associated with any superset of an itemset. AprioriSMP uses these upper bounds to prune unproductive supersets while traversing itemset lattices. In our example, if the chi-squared value is selected as the statistical metric, AprioriSMP prioritizes the second rule over the first according to their chi-squared values.

Several papers [13,7,18,4,9,16] have focused on methods to improve Apriori. These methods effectively use a specific property of the support-confidence framework, called the *anti-monotonicity* of the support function. For example, for any $I \subseteq J, Pr(I) \geq Pr(J)$, we are allowed to discard any superset of itemset I when $Pr(I)$ is strictly less than the given minimum threshold. By contrast, statistical metrics are not anti-monotonic. Therefore, whether we can improve the performance of AprioriSMP using the ideas of these traditional methods is not a trivial question.

Main Result. To overcome this difficulty, we propose a new algorithm, TidalSMP. Table 2 shows the differences between Apriori, AprioriSMP, and our TidalSMP algorithm. TidalSMP is the effective ensemble of three techniques: a vertical layout database to accelerate the database scan and calculate the statistical measure, null elimination to eliminate unproductive itemsets quickly, and set-enumeration tree (or SE-tree in short) traversal to generate candidate itemsets rapidly. Tests indicate that TidalSMP accelerates performance by several orders of magnitude over AprioriSMP, even for difficult cases such as rules with very low support. Furthermore, even in the presence of significant noise involving the most correlated rule, the execution time of the algorithm is independent of the amount of noise.

Table 2. Comparative table with TidalSMP

	Apriori	AprioriSMP	TidalSMP
Layout	Horizontal	Horizontal	Vertical
Traverse	Lattice	Lattice	Set-Enumeration tree
Pruning	Threshold	Upper-bound	Upper-bound and Null Elimination
Evaluation	Support	Statistical Measure	Statistical Measure

2 Preliminaries

TidalSMP frequently uses a method to estimate a tight upper bound of the statistical indices, such as the chi-squared values, entropy gain, or correlation coefficient. For the sake of simplicity in this paper, we present the chi-squared value method.

Chi-squared Values.

Definition 1 Let $I \Rightarrow C$ be an association rule, D be a set of transactions, and n be the number of transactions in D. Let O_{ij} where $(i,j) \in \{I, \bar{I}\} \times \{C, \bar{C}\}$ denotes the number of transactions that contain both i and j. Let O_i where $i \in \{I, \bar{I}, C, \bar{C}\}$ denote the number of transactions that contain i. For instance, $O_{I\bar{C}}$ represents the number of transactions that contain I, but do not include C. O_I represents the number of transactions that contain I, which is equal to the sum of the values in the row $O_{IC} + O_{I\bar{C}}$. Let x, y, and m denote O_I, O_{IC} and O_C, respectively. For each pair $(i,j) \in \{I, \bar{I}\} \times \{C, \bar{C}\}$, we calculate an expectation under the assumption of independence: $E_{ij} = n \times O_i/n \times O_j/n$. The chi-squared value expressed as $chi(x,y)$ is the normalized deviation of observation from expectation; namely,

$$chi(x,y) = \sum_{i \in \{I,\bar{I}\}, j \in \{C,\bar{C}\}} \frac{(O_{ij} - E_{ij})^2}{E_{ij}}.$$

In this definition, each O_{ij} must be non-negative; hence, $0 \leq y \leq x$, and $0 \leq m - y \leq n - x$. $chi(x,y)$ is defined for $0 < x < n$ and $0 < m < n$. We extend the domain of $chi(x,y)$ to include $(0,0)$ and (n,m), and define $chi(0,0) = chi(n,m) = 0$. Incidentally, it is often helpful to explicitly state that x and y are determined by I, and we define $x = x(I)$ and $y = y(I)$.

Table 3. Notation Table for the Chi-squared Value

	C	$\bar{C}$	$\sum row$
I	$O_{IC} = y$	$O_{I\bar{C}}$	$O_I = x$
$\bar{I}$	$O_{\bar{I}C}$	$O_{\bar{I}\bar{C}}$	$O_{\bar{I}} = n - x$
$\sum column$	$O_C = m$	$O_{\bar{C}} = n - m$	n

Theorem 1 [11] For any $J \supseteq I$,

$$chi(x(J), y(J)) \leq \max\{chi(y(I), y(I)), chi(x(I) - y(I), 0)\}.$$

For any I, $0 = chi(0,0) \leq chi(x(I), y(I))$.

Definition 2 $u(I) = \max\{chi(y(I), y(I)), chi(x(I) - y(I), 0)\}$.

Theorem 1 states that for any $J \supseteq I$, $chi(x(J), y(J))$ is bounded by $u(I)$. It is easy to see that $u(I)$ is tight in the sense that there could exist $J \supseteq I$ such that $chi(x(J), y(J)) = u(I)$.

Optimization Problem and AprioriSMP. Many practical applications generate a large number of association rules whose support value is more than the user-specified threshold. To resolve this problem, we use the chi-squared value and define an optimization problem that searches for the N most significant rules.

Optimization Problem: For a fixed conclusion C, compute the optimal association rules of the form $I \Rightarrow C$ that maximize the chi-squared value, or list the N most significant solutions.

This problem is NP-hard if we treat the maximum number of items in an itemset as a variable [11]. In real applications, however, the maximum number is usually bounded by a constant; hence, the problem is tractable from the viewpoint of computational complexity. AprioriSMP [11] is a feasible solution of the optimization problem. There is plenty of room to improve the performance of AprioriSMP because AprioriSMP inherits the itemset generation method of Apriori. To clarify the problem, we define $\mathcal{P}_k$ and $\mathcal{Q}_k$ as follows.

Definition 3 Let τ denote the temporarily Nth best chi-squared value during the computation. An itemset is called a *k-itemset* if it contains k distinct items. We call an itemset I *promising* if $u(I) \geq \tau$ because a superset of I may provide a chi-squared value no less than τ. Let $\mathcal{P}_k$ denote the set of all promising k-itemsets. We use calligraphic letters to express collections of itemsets. We call an itemset I *potentially promising* if every proper subset of I is promising. Let $\mathcal{Q}_k$ denote the set of potentially promising k-itemsets.

We now focus on the Nth best chi-squared value τ. In the k-itemset generation step, AprioriSMP generates all the potentially promising k-itemsets $\mathcal{Q}_k$, calculates their chi-squared values and τ, and then selects promising itemsets $\mathcal{P}_k$ whose chi-squared values are more than τ. This procedure requires us to calculate the chi-squared value of all the itemsets in $\mathcal{Q}_k$; hence, the cost of scanning transactions to calculate chi-squared values is high. Moreover, AprioriSMP calculates chi-squared values of all 1-itemsets in $\mathcal{Q}_1$.

3 Vertical Layout and Null Elimination

In order to reduce the cost of scanning transactions and reduce the size of $\mathcal{Q}_1$, we select a set of transactions in the form of a vertical layout because almost all the layouts used to find association rules depend on the anti-monotonicity of the evaluative function. Furthermore, some researchers [8,15,17] have recently cited the advantage of vertical layouts that maintain a set of transactions containing the item, for each item. In the following, we show the benefit of using vertical layouts to improve the performance of AprioriSMP.

Fig. 1(A) illustrates the bit-vector layout for a set of transactions in which each column denotes an item, while each row represents a transaction. We use letters of the alphabet to denote items, while numerals denote transactions. We decompose the bit-vector layout into vertical layout in Fig. 1(C) despite the horizontal layout in Fig. 1(B) introduced by Agrawal and Srikant.

Vertical Layout In order to express the conclusion for the optimization problem, let us select an item *obj* with special properties.

Definition 4 We call a fixed conclusion item the *objective item*. Let *obj* denote the objective item.

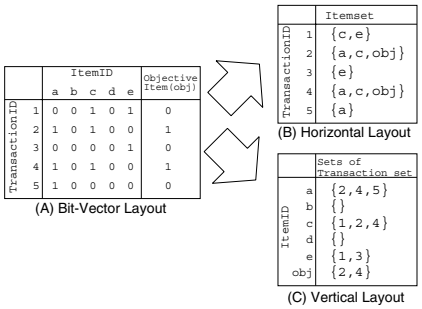

Fig. 1. Database layouts

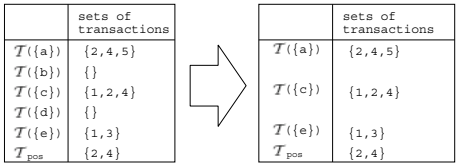

Fig. 2. Null elimination

Since the conclusion is fixed, let us focus on rules in the form $I \Rightarrow \{obj\}$. For itemset I, we need to calculate $chi(x(I), y(I))$. Since $x(I)$ ($y(I)$, respectively) is the number of transactions that contain I ($I \cup \{obj\}$), we need to develop an efficient way of listing transactions that contain I or $I \cup \{obj\}$. For this purpose, we introduce several terms.

Definition 5 Let $\mathcal{T}$ denote a set of transactions. We denote a set of transactions using the calligraphic letter $\mathcal{T}$. Let I be an itemset. Let $\mathcal{T}(I)$ denote $\{T | T$ is a transaction, and $I \subseteq T\}$, which is the set of transactions that contain I. Let *obj* be an objective item, and let T be a transaction. T is called *positive* (or *negative*) if $obj \in T (obj \notin T)$. $\mathcal{T}_{pos}$ and $\mathcal{T}_{neg}$ are then defined:

$$\mathcal{T}_{pos} = \{T \in \mathcal{T} | T \text{ is a positive transaction.}\}$$
$$\mathcal{T}_{neg} = \{T \in \mathcal{T} | T \text{ is a negative transaction.}\}$$

Observe that $\mathcal{T}_{pos}(I)$ is exactly the set of transactions that contain $I \cup \{obj\}$. Consequently, we have $x(I) = |\mathcal{T}(I)|, y(I) = |\mathcal{T}_{pos}(I)|$.

In Fig. 1(C), let $\mathcal{T}$ be $\{1, 2, 3, 4, 5\}$. Then $\mathcal{T}(\{a\}) = \{2, 4, 5\}$, $\mathcal{T}_{pos}(\{a\}) = \{2, 4\}$, and $\mathcal{T}_{neg}(\{a\}) = \{5\}$.

Initializing the Vertical Layout and Null Elimination. Not using the support-confidence framework makes AprioriSMP generate all 1-itemsets. The

following observation, however, helps us to discard useless itemsets in $\mathcal{Q}_1$ without eliminating productive itemsets.

Observation 1 (Null Elimination) Let I be an itemset such that $\mathcal{T}(I)$ is empty. $chi(x(I), y(I))$ is minimum; hence, it is safe to eliminate I from consideration.

Proof Let I be an itemset such that $\mathcal{T}(I)$ is the empty set. It is immediately apparent that $x(I) = |\mathcal{T}(I)| = 0$ and $y(I) = |\mathcal{T}_{pos}(I)| = 0$, which implies that $chi(x(I), y(I)) = chi(0, 0) = 0$. Since $chi(0, 0)$ is the minimum from Theorem 1, $chi(x(I), y(I))$ is no greater than $chi(x, y)$ for any x, y.

For instance, Fig. 2 shows how $\{b\}$ and $\{d\}$ are eliminated. In practice, this procedure is effective for reducing the size of the initial vertical layout. Incidentally, in Fig. 2, $\mathcal{T}_{pos}$ is displayed as a substitute for its equivalent value $\mathcal{T}(\{obj\})$.

Incremental Generation of Vertical Layouts. We now show how to generate vertical layouts incrementally without scanning all the transactions from scratch. To be more precise, given $\mathcal{T}(I_1)$ and $\mathcal{T}(I_2)$, which were computed in the previous steps, we develop an efficient way of computing $\mathcal{T}(I_1 \cup I_2)$ and $\mathcal{T}_{pos}(I_1 \cup I_2)$ from $\mathcal{T}(I_1)$ and $\mathcal{T}(I_2)$ instead of the whole transaction $\mathcal{T}$. The first step is the representation of $\mathcal{T}(I_i)$ as the union of $\mathcal{T}_{pos}(I_i)$ and $\mathcal{T}_{neg}(I_i)$. It is fairly straightforward to prove the following property.

Observation 2 Let I be an itemset. $\mathcal{T}(I) = \mathcal{T}_{pos}(I) \cup \mathcal{T}_{neg}(I)$, $\mathcal{T}_{pos}(I) = \mathcal{T}(I) \cap \mathcal{T}_{pos}$, and $\mathcal{T}_{neg}(I) = \mathcal{T}(I) - \mathcal{T}_{pos}$

For instance, in Fig. 3, when $\mathcal{T} = \{1, 2, 3, 4, 5\}$, $\mathcal{T}(\{a\}) = \{2, 4, 5\}$, $\mathcal{T}_{pos}(\{a\}) = \{2, 4\}$, and $\mathcal{T}_{neg}(\{a\}) = \{5\}$. The following observation is helpful for computing $\mathcal{T}_{pos}(I_1 \cup I_2)$ efficiently.

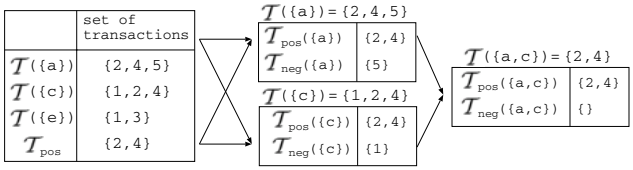

Fig. 3. Incremental Itemset Generation

Observation 3 Let I_1 and I_2 be itemsets.
$\mathcal{T}_{pos}(I_1 \cup I_2) = \mathcal{T}_{pos}(I_1) \cap \mathcal{T}_{pos}(I_2)$ and $\mathcal{T}_{neg}(I_1 \cup I_2) = \mathcal{T}_{neg}(I_1) \cap \mathcal{T}_{neg}(I_2)$

Combining Observations 2 and 3 allows us to calculate vertical layouts incrementally. Fig. 3 illustrates this process. We also note that the scanning cost of each k-itemset decreases as k increases during the computation because $|\mathcal{T}_{pos}(I_1 \cup I_2)| \leq \min\{|\mathcal{T}_{pos}(I_1)|, |\mathcal{T}_{pos}(I_2)|\}$ and $|\mathcal{T}_{neg}(I_1 \cup I_2)| \leq \min\{|\mathcal{T}_{neg}(I_1)|, |\mathcal{T}_{neg}(I_2)|\}$.

4 Set-Enumeration Tree Traversal

Let us consider whether an Apriori-like lattice traversal is useful for creating candidates according to statistical metrics. Note that there are many ways to generate one itemset by merging smaller itemsets; for instance, $\{a, b, c\}$ can be obtained by joining $\{a, b\}$ and $\{b, c\}$, or $\{a, b\}$ and $\{a, c\}$. Multiple choices for generating candidate itemsets may be detrimental to the overall performance when the number of long candidate itemsets becomes huge, since the cost of scanning itemsets could be enormous. Furthermore, scanning itemsets might be meaningless, because the Nth best chi-squared value τ would be changed as the need arises and the itemsets might be unproductive when they were scanned. To settle the former problem, Bayardo [4] proposed using set-enumeration trees [14] in order to mine long-pattern association rules in the support-confidence framework. To resolve both problems, we present a method with a set-enumeration tree (SE-tree) tailored to the statistical-metric framework.

Set-Enumeration Tree. The SE-tree search framework is a systematic and complete tree expansion procedure for searching through the power set of a given set B. The idea is to first impose a total order on the elements of B. The root node of the tree will enumerate the empty set. The children of a node N will enumerate those sets that can be formed by appending a single element of B to N, with the restriction that this single element must follow every element already in N according to the total order. For example, a fully expanded SE-tree over a set of four elements, where each element of the set is denoted by its position in the ordering, appears in Fig. 4.

Fig. 4 illustrates an SE-tree. In the tree, beginning at the bottom root $\{\}$, there exists a unique path to each node. Thus, $\{a, b, c\}$ can be obtained by appending a, b, and c to $\{\}$. Conversely, in the complete lattice, there are multiple ways of generating $\{a, b, c\}$; for instance, joining $\{a, b\}$ and $\{b, c\}$ or $\{a, b\}$ and $\{a, c\}$. Overall, when considering the generation of candidate itemsets, SE-trees are simpler to use than complete lattices.

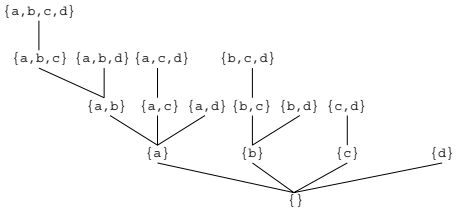

Fig. 4. SE-tree over $\{a, b, c, d\}$

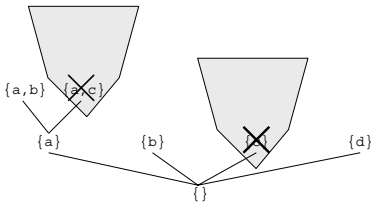

Fig. 5. Pruning branches

Pruning Branches.

Definition 6 Assume that all the items in an itemset are listed in a fixed total order. Let J be an itemset of m items. Let branch(J) denote the set, $\{I | I$ be an

itemset of at least m items, and the set of the first m items in I be equal to $J\}$, which is called the *branch* rooted at J.

Two branches in Fig. 4 are:

branch$(\{a,b\}) = \{\{a,b\},\{a,b,c\},\{a,b,c,d\},\{a,b,d\}\}$
branch$(\{b\}) = \{\{b\},\{b,c\},\{b,d\},\{b,c,d\}\}$

Note that branch(J) does not include all the supersets of J. For example, $\{a,b\}$ is a superset of $\{b\}$, but is not a member of branch$(\{b\})$.

We now introduce a method of pruning the branches on an SE-tree using the upper bounds of statistical values. Theorem 1 leads us to the following observation.

Observation 4 If $\tau > u(J)$, for any I in branch(J), $\tau > u(I)$.

This is because any I in branch(J) is a superset of J. This property enables us to prune all the itemsets in branch(J) at once. For example, in Fig. 5, when $\tau > u(\{c\})$, branch$(\{c\})$ can be pruned altogether. Furthermore, since $\tau > u(\{c\}) \geq u(\{a,c\})$, we can also eliminate branch$(\{a,c\})$.

Itemset Generation. We now describe how to generate new itemsets.

Definition 7 Assume that all the items are ordered. Let I be an itemset. Let head(I) (tail(I), respectively) denote the minimum (maximum) item of I, and let us call the item *head* (*tail*).

For example, when $I = \{b,c,d\}$, head$(I) = b$ and tail$(I) = d$.

SE-tree traversal generates a new set of $(k+1)$-itemsets from the set of k-itemsets $\mathcal{Q}_k$ and the set of 1-itemsets $\mathcal{B}_1$. It selects $Q \in \mathcal{Q}_k$ and $B \in \mathcal{B}_1$ such that tail$(Q) <$ head(B), and it generates a $(k+1)$-itemset by appending B to Q. For instance, let us consider the case when $\mathcal{Q}_2 = \{\{a,b\}\}$ and $\mathcal{B}_1 = \{\{a\},\{b\},\{d\}\}$. Of all possible pairs of $Q \in \mathcal{Q}_2$ and $B \in \mathcal{B}_1$, $Q = \{a,b\}$ and $B = \{d\}$ meet the condition that tail$(Q) <$ head(B). Hence, we put $\{\{a,b,d\}\}$, the appendage of B to Q, into $\mathcal{Q}_3$.

5 Pseudo-code for TidalSMP

We now provide pseudo-code descriptions of TidalSMP in Fig. 6-8. Let τ be the temporarily Nth best chi-squared value. Fig. 6 presents the overall structure of TidalSMP. Fig. 7 shows pseudo-codes for vertical layout decomposition and null elimination, respectively. The program in Fig. 7 makes the decomposition database conform to the vertical layout and generates 1-itemset candidates following Observation 1. The code in Fig. 8 performs a level-wise generation of the SE-tree including the calculation of the intersection of two positive or negative itemsets according to Observation 3. Whenever a new itemset is created, its number of transactions, chi-squared value, and upper bound of the chi-squared value are calculated and stored to avoid duplicate computation.

```
TidalSMP
  k := 1;
  (Q_1, L) =TidalSMP-init;
  // L: list of top N chi-squared values
  B_1 := Q_1; k++;
  repeat begin
    (Q_{k+1}, B_1, L) :=
      SE-tree Traversal(Q_k, B_1, L); k++;
  end until Q_k = φ;
  Return τ with its
  corresponding itemset;
```

Fig. 6. Pseudo-Code of TidalSMP

```
TidalSMP-init
  L := list of top N values in
    {chi(x(I), y(I))|I is a 1-itemset},
  τ := Nth best in L;
  for each J ∈ {I|I is a 1-itemset,
      T(I) ≠ φ, τ ≤ u(I)}
      // Null-Elimination
    Put J into Q_1;
    Calculate T_pos(J) and T_neg(J);
  end
  Return Q_1 and L;
```

Fig. 7. Pseudo-Code of TidalSMP-init

```
SE-tree Traversal( Set of k-itemsets Q_k,
                   Set of 1-itemsets B_1,
                   list of top N values L)
  τ := Nth best in L;
  for each Q ∈ Q_k, B ∈ B_1
          st. tail(Q) < head(B)
    if u(B) < τ; then
      Delete B from B_1;
      // delete one branch
    else
      T_pos(B ∪ Q) := T_pos(B) ∩ T_pos(Q);
      T_neg(B ∪ Q) := T_neg(B) ∩ T_neg(Q);
      Put J into Q_{k+1}
            if T(J) ≠ φ and u(J) ≥ τ;
      L := list of top N values in
                L ∪ {chi(x(J), y(J))};
      τ := Nth best in L;
    end
  end
  Return Q_{k+1}, B_1, τ;
```

Fig. 8. Pseudo-Code of SE-tree Traversal

6 Experimental Results

We evaluated the overall performance of TidalSMP implemented in C++ with a Pentium-III 450-MHz processor and 768 MB of main memory on Linux. We generated a test dataset using the method introduced by Agrawal and Srikant [3]. To intentionally create an optimal association rule, we arbitrarily selected one maximal potentially large itemset, called X, and doubled the probability so that this itemset would be picked during generation of the test dataset. The other itemsets were selected according to the method in [3]. We then selected item c in X as the objective item, making $(X - \{c\}) \Rightarrow \{c\}$ the optimal association rule. The parameters and their default values used in the test dataset generator were as follows:

|D|: Number of transactions
|T|: Average size of transactions
|I|: Average size of maximal and potentially large itemsets
|M|: Number of maximal and potentially large itemsets
W: Number of items

Figs. 9-12 present the experimental results. We used the default parameters |D| = W = 10K, |T| = 20, |I| = 10 and |M| = |D|/10 unless otherwise stated,

and we calculated the association rule with the maximum chi-squared value. Each figure shows the execution time of our algorithms, including both the time required to load the database from a hard disk and the time elapsed using our algorithms. The datasets in the secondary disk were originally stored in an Apriori-like layout. Those datasets were then loaded into the main memory in a vertical layout.

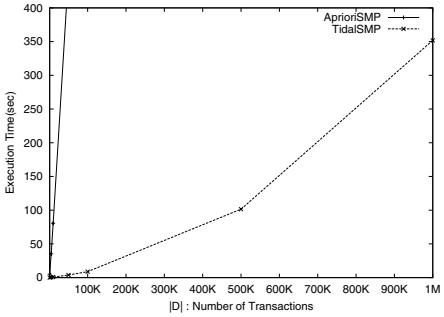

Fig. 9. Scalability of the performance (D≤1M.W50K)

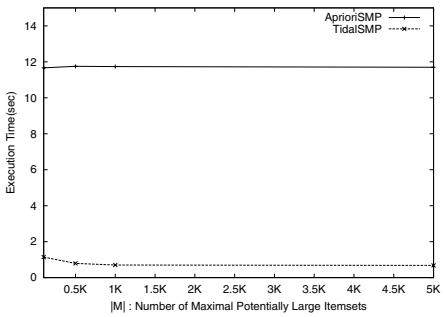

Fig. 10. The performance of low support (D10K.W10K.M≥100)

Scalability. Fig. 9 demonstrates the performance of TidalSMP when |D| ranges from 1K to 1M. TidalSMP accelerates the performance by several orders of magnitude over AprioriSMP. For TidalSMP, the execution time increases quadratically in |D| because the number of occurrences of items in vertical layouts is quadratic in |D|.

Rules with Low Support. We next investigated the performance of mining the optimal association rule $(X - \{c\}) \Rightarrow \{c\}$ with low support. Such a dataset can be obtained by increasing |M|. For instance, when |M| = 5K and |D| = 10K, the average support of the optimal association rule is 0.04% because the probability that X is randomly selected is defined as $2/|\mathrm{M}|(= 0.04\%)$. Similarly, if we set |M| to 0.5K, the support is 0.4%. Fig. 10 shows that the execution time decreases when the support for the statistically optimal association rule also decreases. This might appear to contradict our expectations, but it really could happen because the implementation of the vertical layout is effective in this situation. Note that the lower the support for the optimal rule becomes, the smaller the size of each vertical layout.

Tolerance of Noise. The use of statistical values is expected to allow derivation of the most correlated rule even in the presence of many noisy transactions that are irrelevant to the optimal solution, since noise can naturally be ignored by statistical inference. In order to verify this conjecture, we performed two experiments.

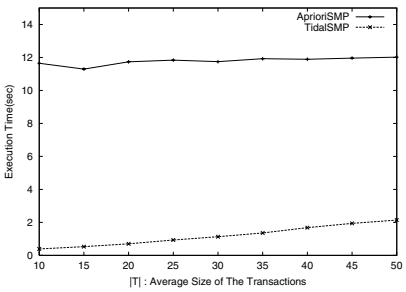

Fig. 11. Tolerance of noise (T≤50.I10.D10K.W10K)

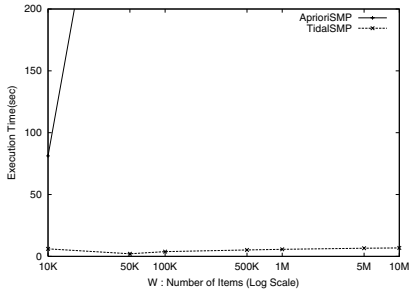

Fig. 12. Tolerance of noise by numerous items (D50K.W≤10M)

First, we intentionally supplied a large number of noisy transactions by increasing |T| to 50 while setting |I| to 10. Fig. 11 shows that the execution time increased moderately and was proportional to |T|.

Next, we considered the case in which we increased the number of items to ten million. Note that the x-axis W is in log scale in 12. Fig. 12 shows that the execution time is independent of the number of items, except when the number of items is less than 10 K. The result suggests that the addition of items irrelevant to the optimal association rule does not have an impact on the overall execution time. One reason for the performance improvement is the quick elimination of unproductive itemsets by null elimination and the dynamic change of the threshold τ.

Both experiments indicated that mining statistically optimal association rules tolerates the presence of noise in datasets.

7 Conclusion

We have presented the heuristics for the vertical decomposition of a database, for pruning unproductive itemsets, and for traversing the SE-tree of itemsets that are tailored to the calculation of association rules with significant statistical metrics. This combination of tree techniques accelerates the overall performance. Therefore, for an experimental database that contains more than 10 million items or a million transactions, our algorithm can efficiently calculate the optimal association rules of the database.

Finding the correlation between itemsets is applicable to various problems. We have been applying this technique to the analysis of correlation between multiple genotypes and the objective phenotype of interest [12].

References

1. C. C. Aggarwal and P. S. Yu. A new framework for itemset generation. In *Proc. of PODS'98*, pp. 18-24, June 1998. 410, 411

2. R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proc. of SIGMOD'93*, pp. 207-216, May 1993. 410
3. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of VLDB'94*, pp. 487-499, Sept. 1994. 412, 419
4. R. J. Bayardo Jr. Efficiently mining long patterns from databases. In *Proc. of SIGMOD'98*, pp. 85-93, June 1998. 412, 417
5. R. J. Bayardo and R. Agrawal Mining the most interesting rules. In *Proc. of SIGKDD'99*, pp. 145-153, Aug. 1999. 412
6. S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *Proc. of SIGMOD'97*, pp. 265-276, May 1997. 410, 411
7. S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket analysis. In *Proc. of SIGMOD'97*, pp. 265-276, May 1997. 412
8. B. Dunkel and N. Soparkar. Data organization and access for efficient data mining. In *Proc. of ICDE'99*, pp. 522-529, March 1999. 414
9. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. of SIGMOD'00*, pp. 1-12, May 2000. 412
10. B. Liu, W. Hsu, and Y. Ma. Pruning and summarizing the discovered associations. In *Proc. of SIGKDD'99*, pp. 125-134, 1999. 410
11. S. Morishita and J. Sese. Traversing lattice itemset with statistical metric pruning. In *Proc. of PODS'00*, pp. 226-236, May 2000. 410, 411, 412, 413, 414
12. A. Nakaya, H. Hishigaki, and S. Morishita. Mining the quantitative trait loci associated with oral glucose tolerance in the OLETF rat. In *Proc. of Pacific Symposium on Biocomputing*, pp. 367-379, Jan. 2000. 421
13. J. S. Park, M. S. Chen, and P. S. Yu. An effective hash-based algorithm for mining association rules. In *Proc. of SIGMOD'95*, pp. 175-186, May 1995. 412
14. R. Rymon. Search through systematic set enumeration. In *Proc. of KR'92*, pp. 539-550, 1992. 417
15. P. Shenoy, J. R. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah. Turbocharging vertical mining of large databases. In *Proc. of SIGMOD'00*, pp. 22-33, May 2000. 414
16. G. I. Webb. Efficient search for association rules. In *Proc. of SIGKDD'00*, pp. 99-107, Aug. 2000. 412
17. M. J. Zaki. Generating non-redundant association rules. In *Proc. of SIGKDD'00*, pp. 34-43, Aug. 2000. 414
18. M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *Proc. of KDD'97*, pp. 343-374, Aug. 1997. 412