Parallel Branch-and-Bound Graph Search for Correlated Association Rules

Shinichi Morishita[*]
University of Tokyo
moris@is.s.u-tokyo.ac.jp

Akihiro Nakaya[†]
University of Tokyo
nakaya@ims.u-tokyo.ac.jp

Abstract

There have been proposed efficient ways of enumerating all the association rules that are interesting with respect to support, confidence, or other measures. In contrast, we examine the optimization problem of computing the optimal association rule that maximizes the significance of the correlation between the assumption and the conclusion of the rule. We propose a parallel branch-and-bound graph search algorithm tailored to this problem. The key features of the design are (1) novel branch-and-bound heuristics, and (2) a rule of rewriting conjunctions that avoids maintaining the list of visited nodes. Experiments on two different types of large-scale shared-memory multi-processors confirm that the speed-up of the computation time scales almost linearly with the number of processors, and the size of search space could be dramatically reduced by the branch-and-bound heuristics.

1 Introduction

Many organizations are seeking strategies for processing or interpreting massive amounts of data that will inspire new marketing strategies or lead to the next generation of scientific discoveries. In response to those demands, in recent years, decision support systems and data mining systems have rapidly attracted strong interests, and numerous optimization techniques for computing decision trees, clusters, and association rules have been proposed. Among those techniques, the development of efficient ways of computing association rules has attracted considerable attention.

[*]Address: Office 301, 7th Building, Department of Information Science, Faculty of Science, University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan. Phone&FAX: +81-3-5841-4116.

[†]Address: Department of Genome Knowledge Discovery System (Hitachi), Institute of Medical Science, University of Tokyo 4-6-1, Shirokane-dai, Minato Ward, Tokyo 108-8639, Japan. Phone: +81-3-5449-5767. FAX: +81-3-5449-5568.

Association Rules

Given a set of records, an association rule is an expression of the form $X \Rightarrow Y$, where X and Y are tests on records, and X and Y are called the *assumption* and the *conclusion*, respectively. Consider the market basket analysis problem [1]. An example of an association rule is: "50% of customers who purchase bread also buy butter; 20% of customers purchase both bread and butter." We will describe the rule by

$$(Bread = 1) \Rightarrow (Butter = 1).$$

We call 50% the *confidence* of the rule and 20% the *support* of the rule.

The significance of an association rule has been evaluated by support and confidence [1, 2]. Higher support implies that the coverage of the rule is sufficiently large, while higher confidence shows that the prediction accuracy of using the assumption X as a test for inferring the conclusion Y is sufficient. In their pioneering work, Agrawal et al. [1, 2] define that an association rule is interesting if its support and confidence are no less than given thresholds, and they propose *Apriori* algorithm that enumerates all the interesting association rules. The idea of Apriori algorithm has been explored by many researchers [2, 8, 9, 10, 11, 12].

Motivating Example

Higher support and higher confidence, however, are not necessarily sufficient for evaluating the correlation between the assumption and the conclusion of an association rule. Brin et al. [5] address this problem, and the following example illustrates this issue.

Example 1.1 Consider the super market basket analysis problem [1]. Let $Bread, Butter$ and $Battery$ be Boolean attributes. Suppose that the support and the confidence of the following rule are 29% and 48.3%, respectively:

$$(Bread = 1) \wedge (Butter = 1) \Rightarrow (Battery = 1),$$

which means that customers who purchase both bread and butter may also buy batteries. This implication differs from our common sense, but the support and the confidence are fairly high, and hence one may conclude that the rule presents some unknown behavior of the customers. From a statistical viewpoint, however, we also ought to look at the negative implication that when customers who do not purchase both bread and butter may also buy batteries. In Table 1, which is called a contingency table, the row $(Bread = 1) \wedge (Butter = 1)$ and the row $not((Bread = 1) \wedge (Butter = 1))$ show the number of customers who do and do not meet $(Bread = 1) \wedge (Butter = 1)$, while the column $(Battery = 1)$ and the column $not(Battery = 1)$ shows their corresponding numbers, similarly.

Note that $Battery = 1$ holds for 50% of all the customers, which is higher than 48.3%, and hence customers satisfying $(Bread = 1) \wedge (Butter = 1)$ are less

	$(Battery = 1)$	$not(Battery = 1)$	Sum
$(Bread = 1) \wedge (Butter = 1)$	29	31	60
$not((Bread = 1) \wedge (Butter = 1))$	21	19	40
Sum	50	50	100

Table 1: Contingency Table

likely to meet $Battery = 1$. Thus there is a slight negative correlation between $(Bread = 1) \wedge (Butter = 1)$ and $Battery = 1$, though it is not significant. ∎

The above example suggests that we should measure the statistical significance of the correlation between the assumption and the conclusion. To measure the significance of correlation, the χ^2 value has usually been applied to the contingency table associated with the rule. The benefit of using the χ^2 value is that we can evaluate the significance of an association rule by a single value rather than multiple values such as support and confidence. All association rules can be ordered by their χ^2 values. We are then interested in finding the optimal association rule that maximizes the χ^2 value. Or we want to list the best n association rules in descending order of χ^2 value. We can also provide a cutoff value — say, at the 95% significance level — for χ^2, and then we can enumerate all the association rules whose χ^2 values are no less than that threshold. We will consider those problems, and we call an association rule *correlated* if its χ^2 value is optimal, sub-optimal or no less than a given threshold value.

Related Work

Brin et al.[5] have studied this problem from a slightly different aspect. Instead of finding correlated association rules, they focus on the computation of a set of primitive tests that are not independent by the chi-squared test. Using the strategy of Apriori algorithm [2], they present an algorithm for enumerating all the sets of primitive tests that are not independent, but the algorithm is not intended to compute correlated association rules.

Example 1.2 Let us consider the market basket analysis problem again. Suppose that $(Spaghetti = 1)$, $(Tabasco = 1)$, and $(Battery = 1)$ are not independent, because $(Spaghetti = 1)$ and $(Tabasco = 1)$ are correlated. We however cannot conclude that $(Spaghetti = 1) \wedge (Tabasco = 1) \Rightarrow (Battery = 1)$ is a correlated association rule, since the assumption and the conclusion may not be correlated at all. ∎

One may try to use Brin et al.'s algorithm to enumerate instances of $X \cup Y$ that are not independent and then try to derive correlated association rules. But there could be numerous instances of $X \cup Y$ from which no correlated association rules could be created, because even if primitive tests in X are correlated, X and Y are not correlated at all.

To keep the computation efficient, Brin et al. use a minimum support threshold as a pruning criteria. In practice, selecting a minimum support threshold requires some considerations, because using a higher threshold often results in pruning important patterns with lower support, while using a lower threshold might produce a huge amount of patterns, which is computationally costly. From the viewpoint of statistics, only the χ^2 value is essential, and hence Brin et al. discuss the possibility of avoiding the heuristics of using the minimum support threshold. We will work in this direction.

Overview

We define our problem more formally. Given a set of Boolean attributes, we select B as a special attribute and call it an *objective* attribute, while we call all the other attributes *conditional*. We use conditional attributes in the assumption of a rule. Consider all the association rules of the form

$$(A_1 = v_1) \wedge \ldots \wedge (A_k = v_k) \Rightarrow (B = 1),$$

where $v_i = 0$ or 1. We first remark that it is NP-hard to compute the optimal conjunction in the assumption that maximizes the χ^2 value. One may try to modify Apriori algorithm to compute the optimal conjunction, but this approach may not be promising, because Apriori algorithm is designed to enumerate all the possible association rules of interest, while our optimization problem targets the optimal conjunction or sub-optimal ones.

To cope with such optimization problems, one common approach is an iterative improvement graph search algorithm that initially selects a candidate conjunction by using a greedy algorithm and then tries to improve the ensemble of candidate conjunctions by a local search heuristic; that is, from a conjunction we generate a *neighboring* conjunction that is obtained by replacing one primitive test with another, by deleting a test, or by inserting a new test. Figure 1(a) represents the search space of all conjunctions by an undirected graph in which a pair of neighboring conjunctions is connected by an edge. Starting from the initial conjunction represented by the square dot, we want to search the graph without visiting the same node more than once. Figure 1(b) illustrates such an example.

To accelerate the performance of graph search, parallelizing the search has been studied for various discrete optimization problems [3, 6]. We will exploit this approach for searching the optimal conjunction. To avoid the repetition of visiting the same node, conventional graph search algorithms maintain the list of visited nodes [3, 6], which however could be a severe bottleneck of parallel search. We instead propose a rule of rewriting a conjunction to others. We first apply the rewriting rule to the initial conjunction to obtain child conjunctions, and then we repeat application of the rule to descendant conjunctions so that we can visit every conjunction just once without maintaining the list of visited conjunctions. Moreover, each application of the rewriting rule can be well parallelized.

If the initial conjunction is empty, it is rather trivial to build such a search tree. For instance, we can create one child of a conjunction by inserting one

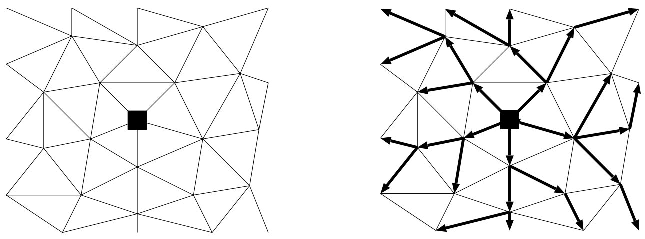


Figure 1: The figure (a) shows the search space of conjunctions. The figure (b) shows the distributed search tree rooted at the square black dot.

primitive test. In general, however, an arbitrary conjunction could be selected as the initial conjunction, and we need to create a neighboring conjunction by using one of replacement, deletion, or insertion, which makes the extraction of a search tree non-trivial.

To reduce the size of the search tree, we develop a branch-and-bound heuristics appropriate for the significance of correlation. We also develop implementation techniques such as materialization of projections and maintenance of distributed priority queues.

2 Preliminaries

Attributes, Records, and Primitive Tests

The domain of a Boolean attribute is $\{0, 1\}$, where 0 and 1 represent true and false, respectively. Let B be a Boolean attribute, let t denote a record (tuple), and let $t[B]$ be the value for attribute B. A *primitive* test has the form $B = v$ where v is either 0 or 1. A record t meets $B = v$ if $t[B] = v$. A conjunction of primitive tests $t_1, t_2, \ldots, t_k$ is of the form $t_1 \wedge t_2 \wedge \ldots \wedge t_k$. A record t meets a conjunction of primitive tests if t satisfies all the primitive tests. We simply call primitive tests and conjunctions *tests.*

Association Rules

From a given set of Boolean attributes, we select one as a special attribute and call it the *objective* attribute. We call all the other attributes *conditional.* Let B be the objective attribute. An *association rule* has the form:

$$(A_1 = v_1) \wedge \ldots \wedge (A_k = v_k) \Rightarrow (B = v),$$

where $A_i (i = 1, \ldots, k)$ is an conditional attribute, and each of v_i and v is either 0 or 1. For instance, $(Bread = 1) \wedge (Butter = 1) \Rightarrow (Battery = 1)$ is an association rule.

	Y is true.	Y is false	Sum of Row
X is true.	$\|R_1^t\|(=y)$	$\|R_1^f\|(=x-y)$	$\|R_1\|(=x)$
X is false.	$\|R_2^t\|(=m-y)$	$\|R_2^f\|(=n-x-(m-y))$	$\|R_2\|(=n-x)$
Sum of Column	$\|R^t\|(=m)$	$\|R^f\|(=n-m)$	$\|R\|(=n)$

Table 2: Contingency Table

Consider association rule $X \Rightarrow Y$. Let R be a set of records over $\mathcal{R}$, and let $|R|$ denote the number of records in R. Let R_1 be the set of records that meet the assumption X, and let R_2 denote $R - R_1$. We call a record t *positive* (*negative*, resp.) if t satisfies (does not meet) the conclusion Y. Let R^t and R^f denote the set of positive and negative records in R, respectively. Table 2 summarizes numbers of records that meet each condition. Since R is given and fixed, we assume that $|R|$, $|R^t|$, and $|R^f|$ are constants, but $|R_1^t|$, $|R_2^t|$, $|R_2^f|$, and $|R_2^f|$ may vary according to the choice of the assumption X. Let n and m denote $|R|$ and $|R^t|$ respectively, then $|R^f| = n - m$. Let x and y denote $|R_1|$ and $|R_1^t|$ respectively. Observe that if we specify the values of x and y, the values of all the other variables are determined.

Chi-squared Value

The chi-squared value is a normalized deviation of observation from expectation. Table 2 presents observed numbers of records. Expected numbers are calculated as follows: In the entire relation, the probability that a positive record occurs is $\frac{|R^t|}{|R|} = \frac{m}{n}$. Since the observed number of records satisfying X is $|R_1|$, the expected number of records meeting both X and Y is $|R_1|$ times $\frac{m}{n}$. Table 3 presents expected numbers of records. The chi-squared value is defined as the

	Y is true.	Y is false
X is true.	$\|R_1\|\frac{m}{n}$	$\|R_1\|\frac{n-m}{n}$
X is false.	$\|R_2\|\frac{m}{n}$	$\|R_2\|\frac{n-m}{n}$

Table 3: Expected Numbers of Records

total of the squared difference between the observed number and the expected number divided by the expected number for each cell; that is,

$$\frac{(|R_1^t|-|R_1|\frac{m}{n})^2}{|R_1|\frac{m}{n}} + \frac{(|R_1^f|-|R_1|\frac{n-m}{n})^2}{|R_1|\frac{n-m}{n}} + \frac{(|R_2^t|-|R_2|\frac{m}{n})^2}{|R_2|\frac{m}{n}} + \frac{(|R_2^f|-|R_2|\frac{n-m}{n})^2}{|R_2|\frac{n-m}{n}}.$$

Since all the variables are determined by x and y, we will refer the above formula by $\chi^2(x,y)$. If X and Y are independent, the observed number is equal to the expected number (in this case, $\frac{y}{x} = \frac{m}{n}$), and therefore $\chi^2(x,y)$ is equal to 0. In the chi-squared test, if $\chi^2(x,y)$ is greater than a cutoff value – say, at the 95% significance level —, we reject the independence assumption.

Convexity of Function

Let $\phi(x,y)$ be a function that is defined on $(x,y) \in D$. $\phi(x,y)$ is a *convex* function on D if for any (x_1,y_1) and (x_2,y_2) in D and any $0 \leq \lambda \leq 1$,

$$\phi(\lambda(x_1,y_1) + (1-\lambda)(x_2,y_2)) \leq \lambda\phi(x_1,y_1) + (1-\lambda)\phi(x_2,y_2).$$

Let $(x_3,y_3) = \lambda(x_1,y_1)+(1-\lambda)(x_2,y_2)$, then $\phi(x_3,y_3) \leq \max(\phi(x_1,y_1),\phi(x_2,y_2))$.

Proposition 2.1 $\chi^2(x,y)$ is a convex function defined on $0 \leq y \leq x$.

Proof: For any δ_1 and δ_2, define $V = \delta_1 x + \delta_2 y$. Prove $\partial^2\chi^2(x,y)/\partial V^2 \geq 0$. ∎

The convexity of $\chi^2(x,y)$ is crucial to prove the intractability of computing the optimal conjunction. We also use the convexity to derive an effective branch-and-bound heuristics.

Theorem 2.1 Let S denote a set of conjunctions that use conditional attributes, and Y be the objective conclusion. It is NP-hard to find the optimal conjunction $X \in S$ such that the chi-squared value of $X \Rightarrow Y$ is maximum.

Proof: The case for the entropy value is proved in [7]. In the proof, the convexity of the entropy function is essentially used. The argument carries over to the case for the chi-squared value, because the chi-squared function is also convex. ∎

3 Parallel Branch-and-Bound Graph Search

Search Space as an Undirected Graph

Let V denote the set of all conjunctions that use conditional attributes. A conjunction C_1 is *adjacent to* another conjunction C_2 if C_1 is obtained by replacing a primitive test in C_2 with another, by deleting a primitive test in C_2, or inserting a new one to C_2.

Example 3.1 Let C be the conjunction $(A_1 = 1) \wedge (A_2 = 0) \wedge (A_3 = 1)$. C is adjacent to $(A_1 = 1) \wedge (A_2 = 0) \wedge (A_4 = 0)$, because $(A_3 = 1)$ in C is replaced by $(A_4 = 0)$. Also, C is adjacent to $(A_1 = 1) \wedge (A_3 = 1)$ and $(A_1 = 1) \wedge (A_2 = 0) \wedge (A_3 = 1) \wedge (A_5 = 1)$. ∎

Let E denote the set of undirected edges between pairs of adjacent nodes in V; that is, $E = \{(C_1,C_2) \mid C_1 \text{ is adjacent to } C_2\}$. The undirected graph (V,E) represents the search space of all conjunctions. We call (V,E) the *undirected graph* of *adjacency*. Figure 1(a) in Section 1 presents an example. We define the *distance* between nodes v and u by the length of the shortest path between v and u. Put another way, the distance shows the minimum number of operations on primitive tests to generate u from v.

Requirements on Search Tree

Suppose that we are given an arbitrary node $t_1 \wedge \ldots \wedge t_k$ in the search space (V, E) as the initial conjunction. To realize the local search strategy starting from $t_1 \wedge \ldots \wedge t_k$, we need to generate a search tree rooted at $t_1 \wedge \ldots \wedge t_k$ such that (1) the depth from $t_1 \wedge \ldots \wedge t_k$ to any node v in the tree is equal to the distance between $t_1 \wedge \ldots \wedge t_k$ and v in (V, E), and (2) each conjunction is enumerated to appear just once in the tree. For instance, Figure 1(b) illustrates such a search tree rooted at the square black dot.

To build a search tree, we first present a way of creating a unique path from the root of the initial conjunction to the node of any conjunction. We then show how to assemble all the paths into a search tree.

Creating a Unique Path from the Initial Conjunction to Any Conjunction

We introduce a way of representing a conjunction uniquely with respect to the initial conjunction. We develop this idea motivated by techniques for enumerating geometric objects [4]. We assume that all the primitive tests are sorted in a total order, and we denote the order by $x_1 < x_2$. For simplicity of presentation, we introduce a dummy test $\perp$ that is strictly smaller than any test t; that is, $\perp < t$. Let S denote the set of all the primitive tests. Let $t_1 \wedge \ldots \wedge t_k$ be the the initial conjunction given. Let C denote an arbitrary conjunction of primitive tests in S. We represent C by a list of primitive tests according to the following steps:

1. If $t_i (1 \leq i \leq k)$ appears in C, place t_i at the i-th position in the list. Otherwise, leave the i-th position open.

2. Sort all the primitive tests that appear in C but are not in $\{t_1, \ldots, t_k\}$, in the ascending order. Let SL denote the sorted list. Select and remove the first primitive test in SL, and assign it to the leftmost open position. Repeat this process until SL becomes to be empty.

Observe that any conjunction can be represented by the unique list of primitive tests, and hence we call it the *canonical* list.

Example 3.2 Let $t_1 \wedge t_2 \wedge t_3 \wedge t_4 \wedge t_5$ be the initial conjunction. Its canonical list is $[t_1, t_2, t_3, t_4, t_5]$. Let $\bigcirc$ denote an open position. The canonical list of $t_4 \wedge a_1 \wedge t_2 \wedge a_2 \wedge t_5 \wedge a_3$, where $a_1 < a_2 < a_3$, is obtained as follows: We first create $[\bigcirc, t_2, \bigcirc, t_4, t_5]$ by placing each t_i of $t_4 \wedge a_1 \wedge t_2 \wedge a_2 \wedge t_5 \wedge a_3$ at the i-th position. We then assign a_1 and a_2 respectively to the first and the third positions, which are open, and we append a_3 at the end of the list. Consequently we have $[a_1, t_2, a_2, t_4, t_5, a_3]$.

The canonical list of $t_4 \wedge a_1 \wedge a_2$, where $a_1 < a_2$, is obtained similarly. We first create $[\bigcirc, \bigcirc, \bigcirc, t_4, \bigcirc]$, and then assign a_1 and a_2 into the first and the second positions, respectively. Thus we obtain $[a_1, a_2, \bigcirc, t_4, \bigcirc]$. ∎

We show how to rewrite the canonical list of the initial conjunction to that of an arbitrary target conjunction, which creates a unique path from the root to any node. Intuitively, we scan two lists together from left to right, and when we find different primitive tests at the same position, we perform one of replacement, deletion, or insertion so that the initial conjunction is transformed into the target conjunction after the scan.

Example 3.3 Consider canonical lists $[t_1, t_2, t_3, t_4, t_5]$ and $[a_1, t_2, a_2, t_4, t_5, a_3]$. Since the two primitive tests at the first position are different, we replace t_1 by a_1. We then see the difference at the third position, and we replace t_3 by a_2. Finally, a_3 at the sixth position of $[a_1, t_2, a_2, t_4, t_5, a_3]$ does not appear in $[t_1, t_2, t_3, t_4, t_5]$, and hence we insert a_3. Consequently we have rewritten $[t_1, t_2, t_3, t_4, t_5]$ to $[a_1, t_2, a_2, t_4, t_5, a_3]$ by the following sequence of operations:

$$[t_1, t_2, t_3, t_4, t_5] \xrightarrow{\text{replacement}} [a_1, t_2, t_3, t_4, t_5] \xrightarrow{\text{replacement}}$$
$$[a_1, t_2, a_2, t_4, t_5] \xrightarrow{\text{insertion}} [a_1, t_2, a_2, t_4, t_5, a_3]$$

We have applied three operations, and the distance between $t_1 \wedge t_2 \wedge t_3 \wedge t_4 \wedge t_5$ and $a_1 \wedge t_2 \wedge a_2 \wedge t_4 \wedge t_5 \wedge a_3$ in the graph of conjunctions is also 3. ∎

There are some issues on sequences that use deletion.

Example 3.4 Consider the following two sequences

- $[t_1, t_2, t_3, t_4, t_5] \xrightarrow{\text{deletion}} [\bigcirc, t_2, t_3, t_4, t_5] \xrightarrow{\text{insertion}} [a_1, t_2, t_3, t_4, t_5]$
- $[t_1, t_2, t_3, t_4, t_5] \xrightarrow{\text{replacement}} [a_1, t_2, t_3, t_4, t_5]$

The second sequence gives the minimum length path in the undirected graph of adjacency. The following two sequences show another issue:

- $[a_1, t_2, t_3, t_4, t_5, t_6] \xrightarrow{\text{deletion}} [a_1, \bigcirc, t_3, t_4, t_5, t_6] \xrightarrow{\text{replacement}} [a_1, \bigcirc, a_2, t_4, t_5, t_6]$
- $[a_1, t_2, t_3, t_4, t_5, t_6] \xrightarrow{\text{replacement}} [a_1, a_2, t_3, t_4, t_5, t_6] \xrightarrow{\text{deletion}} [a_1, t_2, \bigcirc, t_4, t_5, t_6]$

$[a_1, \bigcirc, a_2, t_4, t_5, t_6]$ is not a canonical list, because $\bigcirc$ appears before a_2. ∎

In each case of the above example, we want to derive the second sequence only. We can solve this problem by using the rule that we do not allow replacement nor insertion once deletion is used. We will prove that this restriction does not overlook the canonical list of any conjunction.

Making Canonical Lists Distributable to Arbitrary Multiple Processes

We present a way of distributing the canonical lists of conjunctions to arbitrary multiple processes so that each process can continue to rewrite independently. Consider the following sequence again:

$$[t_1, t_2, t_3, t_4, t_5] \overset{\text{replacement}}{\to} [a_1, t_2, t_3, t_4, t_5] \overset{\text{replacement}}{\to}$$
$$[a_1, t_2, a_2, t_4, t_5] \overset{\text{insertion}}{\to} [a_1, t_2, a_2, t_4, t_5, a_3]$$

Suppose that we assign the third canonical list $[a_1, t_2, a_3, t_4, t_5]$ to one process. We want to avoid giving to the process the history of creating the previous two canonical lists, because in general the history could be lengthy. We rather provide minimum information to the process so that the process can continue to rewrite the canonical list. For instance, it is enough to provide the information that primitive tests up to the third position have been updated, a_2 is the largest primitive test that has been most recently added, and no primitive test has been deleted. With this information, we can then append a_3, which is greater than a_2, at the end of $[a_1, t_2, a_2, t_4, t_5]$.

In general, we add the following auxiliary information to a canonical list $[x_1, \dots, x_n]$, and we represent the extension by $\langle [x_1, \dots, x_n], n, i, max, dmode \rangle$, which we also call a *canonical* list.

- n: The number of primitive tests in the canonical list.
- i: Let j be an index such that $i \le j$. We can update the primitive test at the j-th position in the next step.
- max: max denotes the largest primitive test among all the primitive tests that have been added. In the next step, we need to add a new primitive test that is greater than max when we perform replacement or insertion. For the initial conjunction we set max to $\perp$, where $\perp$ is the dummy test smaller than any primitive test.
- $dmode$: For the initial conjunction, $dmode = 0$. Once deletion is applied, $dmode$ is set to 1. When $dmode = 1$, only deletion is applicable.

Application of replacement, insertion or deletion to is defined as follows:

- **Replacement**: When $i \le n$ and $dmode = 0$, we can replace the j-th ($j \ge i$) primitive test with x such that $max < x$ and $x \notin \{t_1, \dots, t_k\}$.
  $\langle [x_1, \dots, x_n], n, i, max, 0 \rangle \overset{\text{replacement}}{\to}$
  $\langle [x_1, \dots, x_{j-1}, x, x_{j+1}, \dots, x_n], n, j+1, x, 0 \rangle$.
- **Insertion**: When $dmode = 0$, we can insert x such that $max < x$ and $x \notin \{t_1, \dots, t_k\}$ at the end of the list.
  $\langle [x_1, \dots, x_n], n, i, max, 0 \rangle \overset{\text{insertion}}{\to}$
  $\langle [x_1, \dots, x_n, x], n+1, n+2, x, 0 \rangle$.
- **Deletion**: When $i \le n$, we can delete the j-th ($j \ge i$) primitive test, and we set $dmode$ to 1.
  $\langle [x_1, \dots, x_n], n, i, max, dmode \rangle \overset{\text{deletion}}{\to}$
  $\langle [x_1, \dots, x_{j-1}, \bigcirc, x_{j+1}, \dots, x_n], n-1, j+1, max, 1 \rangle$

Table 4 presents two examples of such sequences.

	$\langle [t_1, t_2, t_3, t_4, t_5], 5, 1, \perp, 0 \rangle$		$\langle [t_1, t_2, t_3, t_4, t_5, t_6], 6, 1, \perp, 0 \rangle$
$\overset{\text{replacement}}{\rightarrow}$	$\langle [a_1, t_2, t_3, t_4, t_5], 5, 2, a_1, 0 \rangle$	$\overset{\text{replacement}}{\rightarrow}$	$\langle [a_1, t_2, t_3, t_4, t_5, t_6], 6, 2, a_1, 0 \rangle$
$\overset{\text{replacement}}{\rightarrow}$	$\langle [a_1, t_2, a_2, t_4, t_5], 5, 4, a_2, 0 \rangle$	$\overset{\text{replacement}}{\rightarrow}$	$\langle [a_1, a_2, t_3, t_4, t_5, t_6], 6, 3, a_2, 0 \rangle$
$\overset{\text{insertion}}{\rightarrow}$	$\langle [a_1, t_2, a_2, t_4, t_5, a_3], 6, 7, a_3, 0 \rangle$	$\overset{\text{deletion}}{\rightarrow}$	$\langle [a_1, a_2, \bigcirc, t_4, t_5, t_6], 5, 4, a_2, 1 \rangle$
		$\overset{\text{deletion}}{\rightarrow}$	$\langle [a_1, a_2, \bigcirc, t_4, \bigcirc, t_6], 4, 6, a_2, 1 \rangle$

Table 4: Examples of Distributable Sequences

Theorem 3.1 Let A and B denote a given initial conjunction and an arbitrary conjunction. There exists a unique sequence of application of replacement, insertion or deletion that rewrites the canonical list of A to that of B. Furthermore, the number of instances of application is equal to the distance between A and B in the undirected graph of adjacency. ▮

Proof: The proof is an induction on the number of positions where the two primitive tests disagree in A and B, and let d denote the number. Observe that d is equal to the distance between A and B in the undirected graph of adjacency. We construct a unique sequence of rewriting A into B by applying one of the three operations d times.

Suppose that the initial conjunction A contains n primitive tests in it, and its canonical form is $[t_1, \ldots, t_n]$. Let n_B denote the number of primitive tests in B. In what follows, for simplicity and readability, we assume that A and B denote their canonical forms.

We first consider the base case when $d = 1$. Since $d = 1$, the number of primitive conjunctions n_B is either $n - 1$, n or $n + 1$. In each case, we can generate B from A by the application of deletion, replacement, or insertion.

- When $n_B = n - 1$, B must contain an open position $\bigcirc$, and suppose that $\bigcirc$ is located at the j-th position. Deleting the j-th primitive test in A yields B.
- When $n_B = n$, suppose that A and B disagree at the j-th position. B can be generated by replacing the primitive test at the j-th position in A with that at the j-th position in B.
- When $n_B = n + 1$, A and B are equal except that B has an extra primitive test at the $(n + 1)$-th position, and hence B can be created by inserting the last primitive test of B into A.

When $d > 1$, we consider the three cases below:

- When $n_B < n$, B must contain some open positions, and let j denote the last position where $\bigcirc$ is located. Replace $\bigcirc$ at j-th open position in B with the primitive test at the j-th position in A, and let B' denote the result. By the inductive hypothesis, there exists a unique sequence of $(d - 1)$ operations that rewrite A into B'. Note that B can be rewritten from B' by one operation of deletion.

- When $n_B = n$, A and B disagree at d positions, and let j denote the last position of disagreement. Let B' denote the result of replacing the primitive test at the j-th position in B with that at the j-th position in A. B' can be obtained from A by $(d-1)$ operations by the inductive hypothesis, and B can be generated from A by one operation of replacement.

- When $n_B > n$, B has extra $(n_B - n)$ primitive tests at the end. Let B' denote the result of deleting the last primitive test from B. By the inductive hypothesis, there is a unique sequence of $(d-1)$ operations to rewrite A into B'. We can obtain B by the application of insertion to B'.

∎

Distributable Search Tree

The *distributable search tree* is a binary tree that displays all the sequences from the initial canonical list to the canonical list of any conjunction. Figure 2 illustrates such an example. Theorem 3.1 implies that any distributable search tree meets the two requirements on search trees; that is, (1) the depth from the root to any node v in the tree is equal to the distance between the root and v in the graph of adjacency, and (2) each conjunction is enumerated to appear just once in the tree.

Furthermore any node in a distributable search tree can be assigned to any process in a flexible manner.

$$\begin{tikzcd}
 & \langle [t_1, t_2, t_3, t_4, t_5], 5, 1, \perp, 0 \rangle \arrow[dl, "\text{deletion}"'] \arrow[d, "\text{replacement}"] \arrow[dr, "\text{insertion}"] & \\
\langle [\bigcirc, t_2, t_3, t_4, t_5], 4, 2, \perp, 1 \rangle \arrow[d, "\text{deletion}"] & \langle [a_1, t_2, t_3, t_4, t_5], 5, 2, a_1, 0 \rangle \arrow[d, "\text{replacement}"] & \langle [t_1, t_2, t_3, t_4, t_5, a_1], 6, 7, a_1, 0 \rangle \arrow[d, "\text{insertion}"] \\
\langle [\bigcirc, t_2, \bigcirc, t_4, t_5], 3, 4, \perp, 1 \rangle \arrow[d, "\text{deletion}"] & \langle [a_1, t_2, a_2, t_4, t_5], 5, 4, a_2, 0 \rangle \arrow[d, "\text{replacement}"] & \langle [t_1, t_2, t_3, t_4, t_5, a_1, a_2], 7, 8, a_2, 0 \rangle \arrow[d, "\text{insertion}"] \\
\langle [\bigcirc, t_2, \bigcirc, \bigcirc, t_5], 2, 5, \perp, 1 \rangle \arrow[d] & \langle [a_1, t_2, a_2, t_4, a_3], 5, 6, a_3, 0 \rangle \arrow[d] & \langle [t_1, t_2, t_3, t_4, t_5, a_1, a_2, a_3], 8, 9, a_3, 0 \rangle \arrow[d] \\
\cdots & \cdots & \cdots
\end{tikzcd}$$

Figure 2: Example of Distributable Search Tree

Best-First Search

Next we discuss how to traverse the distributed search tree. Suppose that we compute an initial conjunction by a greedy algorithm that always makes the choice that looks best at the moment. Next we need to consider how to scan the distributed search tree rooted at the initial conjunction. One may try the depth-first search or the breadth-first search, but when we look for the conjunction that maximizes the chi-squared value, we should make a locally optimal choice in hope that this choice will lead to the global optimal solution. We therefore select the best-first search strategy that expands a node whose chi-squared value is maximum at the moment.

We implement the best-first search by using a priority queue. First we insert the initial conjunction into the empty queue. We repeat the process that we remove the first node v from the queue, compute the chi-squared value of v, update the best chi-squared value if necessary, use the chi-squared value of v to prioritize each child of v, and insert all the chide nodes of v into the queue.

Later in this section we show how to distribute the queue to multiple processes, but before that we show two techniques to improve the performance of the best-first search.

Branch-and-Bound Heuristics

Suppose that we examine a node v in a distributable search tree. The following theorem shows how to compute an upper bound of the best chi-squared value that could be obtained by scanning all the nodes in the subtree rooted at v. If the upper bound is smaller than the optimum chi-squared value at the moment, we can ignore and prune the subtree.

Theorem 3.2 Let v be a node in a distributed tree. Suppose that

$$v = \langle [x_1, \ldots, x_i, x_{i+1}, \ldots, x_k], k, i+1, -, - \rangle.$$

Let a (b, resp.) denote the number of (positive) records that meet $x_1 \wedge \ldots \wedge x_i$. Let w be an arbitrary descendant of v. Note that the conjunction of w contains $x_1 \wedge \ldots \wedge x_i$. Let p (q, resp.) denote the number of (positive) records that meet the conjunction of w. Recall that $\chi^2(p,q)$ is the chi-squared value of w, and we have:

$$\chi^2(p,q) \ \leq \ \max\{\chi^2(b,b), \chi^2(a-b,0)\}.$$

Proof: Let n and m denote respectively the number of records and the number of positive records in the entire relation. Consider the points (a,b) and (p,q) in the two-dimensional Euclidean plane. It is easy to see that (p,q) falls in the convex region whose vertexes are $(0,0)$, (b,b), (a,b), and $(a-b,0)$. To be more precise, we have $0 \leq p \leq a \leq n$, $0 \leq q \leq b \leq m$, $q \leq p$, $b \leq a$, and $(p-q) \leq (a-b)$. When $y/x = m/n$, $\chi^2(x,y)$ is zero and minimum. Because of the convexity of $\chi^2(x,y)$, it follows that $\chi^2(p,q) \leq \max\{\chi^2(b,b), \chi^2(a-b,0)\}$. ∎

Materialized Projections

Let $v = \langle [x_1, \ldots, x_i, x_{i+1}, \ldots, x_{n_1}], n_1, i+1, -, - \rangle$ be an arbitrary node in a distributed tree. Note that any node in the subtree rooted at v must contain all the primitive tests in $\{x_1, \ldots, x_i\}$, because none of $\{x_1, \ldots, x_i\}$ is updated in the subtree. We call the set of records that meet $x_1 \wedge \ldots \wedge x_i$ the *materialized projection* for v.

A materilized projection could be very large in practice. To utilize the main memory efficiently, we implemented a materialized projection by creating a bit array of indexes to records in the materialized projection. If the bit of an index is on, the record of the corresponding index belongs to the materialized projection. For instance, the size of a bit array for a large database containing ten million records is $1.25MB$. Such bit arrays may still require large memory space during the execution, especially when the queue becomes to be long during the computation and cannot fit in the main memory. In this case, we put aside nodes with lower priorities, which might not be processed for a while, to the secondary disk at the moment, and later we restore them back to the main memory.

We now discuss the benefit of associating the materialized projection with each node. Let $w = \langle [x_1, \ldots, x_i, y_{i+1}, \ldots, y_{n_2}], n_2, j+1, -, - \rangle$ be a descendant of v. When we compute the chi-squared value of w, we need to count the number of records that satisfy the conjunction of w. It suffices to check if each record in the materialized projection for v also satisfies $y_{i+1} \wedge \ldots \wedge y_{n_2}$. The materialized projection could be much smaller than the entire relation. Since counting the number of records that satisfy a conjunction is the crucial step of the whole computation, the use of materialized projections could reduce the computation time substantially. The materialized projection of each node can be computed in an incremental manner; that is, the materialized projection for a child node is a subset of that for its parent.

Distributing Priority Queue to Multiple Processes

It remains to parallelize the single process version of the best-first search. The key extension is to divide the single queue into multiple disjoint queues and to distribute them to multiple processes. Balancing sizes of queues among multiple processes at run time is rather straightforward, because any node can be processed by any process. Each process maintains its own queue and broadcasts the locally best chi-squared value to the others when the value is updated.

There are a couple of concerns that do not arise previously. The first issue is that broadcasting the update of the locally best chi-squared value may increase the communication overhead between the processes. Another concern is that short delay of the broadcast may slightly deteriorate the overall performance, because the branch-and-bound heuristics uses the best chi-squared value at the moment. Tests however show that updates do not occur so often, and therefore those concerns are not serious in practice.

Listing Best n Conjunctions

We have so far presented an algorithm for computing the optimal conjunction, but it is easy to modify the algorithm to list the best n conjunctions. To this end, we can change to maintain the list of the best n conjunctions instead of the best conjunction. After this modification, the branch-and-bound heuristics still works, because we can use the n-th node instead of the best node to prune the search space according to Theorem 3.2.

4 Experimental Results

Implementation

We implemented our algorithm by using C++ and POSIX thread library. Experiments were performed on two different types of large scaled shared-memory multi-processors. One is Sun Microsystems Ultra Enterprise 10000 with 64 UltraSPARC processors running at 250MHz, 16GB of main memory, and 1MB of L2 cache for each processor, working under Solaris 2.5.1. Another is SGI Origin 2000 with 128 R10000 processors running at 195MHz, 24GB main memory, and 4MB L2 cache for each processor, running under IRIX 6.5SE. We limit the size of main memory to 2GB in order to verify that our implementation uses at most 2GB of main memory. In the case of SGI Origin 2000, since the time to access the remote memory is almost three times larger than the time to access the local memory, we had to implement each thread to keep a local copy of the entire relation to accelerate the overall performance.

Test Data

We randomly generated such a relation that the relation contains one hundred thousand records and the value of an attribute in a record is equal to 1 with a probability of p. We show the experimental results when $p = 0.3$, because in this case, the execution time was at most several hours, and therefore we can measure the speed-up and the effect of the branch-and-bound heuristics in a reasonable amount of time. The relation contains one hundred conditional attributes and one objective attribute. We used one hundred primitive tests of the form $(A = 1)$, where A is a conditional attribute. As the conclusion, we used $(B = 1)$, where B is the objective attribute. We selected the initial conjunction in a greedy manner. We applied our implementation to the test data until the algorithm terminates; that is, all the queues become to be empty, and the optimal conjunction is identified.

Effect of Branch-and-Bound Heuristics

Since there are one hundred primitive tests, the algorithm could generate 2^{100} conjunctions in the worst case. As a result of the branch-and-bound heuristics, however, the algorithm generates much less conjunctions. We have performed the cases when numbers of threads are 1, 2, 4, 8, 16, 32, 64, and 128. The total number of conjunctions inserted into the distributed queues ranges from 24194 to 24463. We have observed that every conjunction examined during the search contains at most four primitive tests. Note that the number of all conjunctions with at most four primitive tests is about 4.3×10^6. This figure again indicates that the branch-and-bound heuristics can drastically reduce the search space.

	Enterprise 10000				Origin 2000			
#(threads)	min	max	avg	s.d.	min	max	avg	s.d.
2	12106	12176	12141	35	11431	12852	12141.5	710.5
4	5544	6473	6070.5	335.7	5843	6382	6093.8	201.3
8	2734	3411	3035.3	226.4	2814	3223	3057.9	128.6
16	884	2085	1523.4	309.7	1364	2106	1528.9	166.8
32	453	1189	764.5	150.3	551	1169	746.0	142.1
64	244	632	383.6	101.0	201	765	382.2	103.0
128	N/A	N/A	N/A	N/A	85	413	191.1	67.9

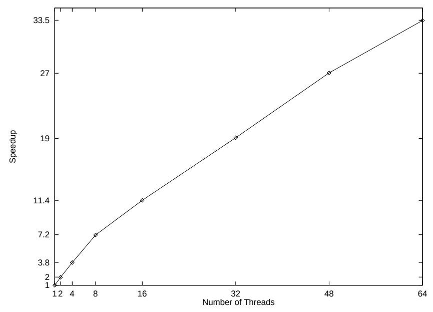
Table 5: Statistics of Numbers of Conjunctions Inserted into Distributed Queues

Effect of Maintaining Distributed Queues

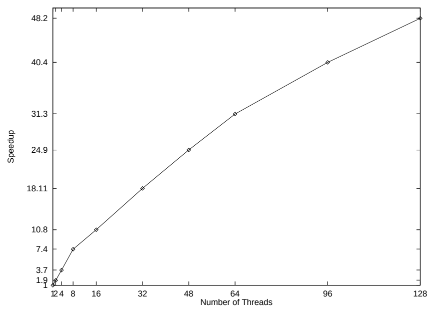
In order to analyze the effect of maintaining distributed queues assigned to multiple processes, Table 5 shows the statistics of the number of conjunctions inserted into each queue. Consider the set of the numbers of conjunctions inserted into all distributed queues. For each number of threads, Table 5 presents the minimum number, the maximum number, the average number, and the standard deviation of the set of those numbers. Observe that the standard deviation of each case is fairly small, which implies that distributing conjunctions to multiple threads works well.

Speed-up

The *speed-up ratio* of n threads is defined as the ratio of the execution time of one thread to the execution time of n threads. Figure 3 (a) and (b) present that the speed-up scales almost linearly with the number of threads on both Sun Microsystems Ultra Enterprise 10000 and SGI Origin 2000. Table 6 shows the execution time in seconds.



(a) Sun Microsystems Ultra Enterprise 10000



(b) SGI Origin 2000

Figure 3: Speed-up Ratio

#(threads)	Enterprise 10000		Origin 2000	
	Execution Time	Speed-up Ratio	Execution Time	Speed-up Ratio
1	6,760	1.00	6,503	1.00
64	202	33.47	219	31.30
128	N/A	N/A	135	48.17

Table 6: Execution Time in Seconds and Speed-up Ratio

Optimizing the Objective Criteria

Until the system finds the optimal conjunction, it outputs the optimal conjunction at the moment. Let X and Y denote the assumption and the conclusion of an association rule. Table 7 presents candidates of the optimal conjunction that the system output during the computation when the system was executed as 64 threads on Sun Enterprise 10000.

Assumption X	X is true.		X is false.		χ^2
	Y is true.	Y is false.	Y is true.	Y is false.	
$[t_1, t_2]$	9807	4240	20575	65378	12014.836
$[a_{10}, t_2]$	5962	411	24420	69207	12841.383
$[a_{15}, a_{23}, a_{90}]$	5810	161	24572	69457	13445.606
$[t_1, a_{39}, a_{90}]$	5833	148	24549	69470	13559.018

Table 7: Candidates of Optimal Conjunctions Calculated During the Computation ($[t_1, t_2]$ is the initial conjunction, and $[t_1, a_{39}, a_{90}]$ is the optimal one.)

Relationship between Execution Time and Size of Search Space

We have so far presented the performance of our system applied to the set of one hundred thousand records such that the value of each attribute in a record is equal to 1 with a probability of $p = 0.3$. If we use higher values for the probability p, the number of conjunctions examined increases, and therefore the total execution time also grows. Table 8 summarizes the performance results of executing our algorithm as 32 threads on SUN Ultra Enterprise 10000. The execution time does not always scale to the number of conjunctions examined, since time to handle a longer conjunction with more primitive tests decreases because of the effect of using materialized projections. The execution time also depends on the structure of the search tree. But in general, the growth of the number of conjunctions raises the execution time.

5 Conclusion

We have examined the optimization problem of computing the optimal conjunction maximizing the chi-squared value that indicates the significance of the correlation between the assumption and the conclusion of the rule. Although

p	Execution Time in Seconds	Number of Conjunctions
0.3	354	24,463
0.4	1,396	74,169
0.5	2,525	233,148
0.6	8,261	803,280

Table 8: Relationship between the Performance and the Size of Search Space

this optimization problem is NP-hard, we have introduced a novel data structure called the distributable search tree, and we have presented how to construct this tree and how to speed up the performance of searching the distributable search tree on multiple processes. Our technique carries over to the general cases when we use the entropy function, the gini index, or the correlation coefficient as evaluation criteria.

In Section 4, we use a synthesis data to evaluate the performance of our system. In practice, we have been applying our system to the analysis of multiple factors leading to a common disease such as diabetes, or high blood sugar level. This case poses another problem of finding a conjunction to split data into two classes so that the average of the objective numeric attribute values in one class is substantially higher than that in the other class. It is however NP-hard to find the optimal conjunction that maximizes the interclass variance [7]. Developing an effective branch-and-bound heuristics for this case is an interesting problem.

Acknowledgements

All experimental results are done by using parallel machines at Human Genome Center, Institute of Medical Science, University of Tokyo. This research is partly supported by Grant-in-Aid for Scientific Research on Priority Areas "Discovery Science" from the Ministry of Education, Science and Culture, Japan.

References

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

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of VLDB Conference*, pages 487–499, 1994.

[3] G. Y. Ananth, V. Kumar, and P. Pardalos. Parallel processing of discrete optimization problems. 1993.

[4] D. Avis and K. Fukuda. A basis enumeration algorithm for linear systems with geometric applications. *Applied Mathematics Letters*, 5:39–42, 1991.

[5] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules tocorrelations. In *Proceedings of ACM SIGMOD*, pages 265–276. *SIGMOD Record* 26(2), June 1997.

[6] V. Kumar, A. Grama, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms.* Benjamin Cummings, Nov. 1993.

[7] S. Morishita. On classification and regression. In *Proceedings of Discovery Science, DS'98, Lecture Notes in Artificial Intelligence*, volume 1532, pages 40–57, Dec. 1998.

[8] R. T. Ng, L. V. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained association rules. In *Proceedings of ACM SIGMOD*, pages 13–24, June 1998.

[9] J. S. Park, M.-S. Chen, and P. S. Yu. An effective hash-based algorithm for mining association rules. In *Proceedings of ACM SIGMOD*, pages 175–186, May 1995.

[10] R. J. Bayardo Jr. Efficiently Mining Long Patterns from Databases. In *Proceedings of ACM SIGMOD*, pages 85–93, June 1998.

[11] R. J. Bayardo Jr., R. Agrawal, D. Gunopulos. Constraint-Based Rule Mining in Large, Dense Databases. In *Proceedings of ICDE*, pages 188-197, March 1999.

[12] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proceedings of ACM SIGMOD*, June 1996.