FARMER: Finding Interesting Rule Groups in Microarray Datasets

Gao Cong, Anthony K. H. Tung*†,
Xin Xu, Feng Pan
Dept. of Computer Science
Natl. University of Singapore
{conggao,atung,xuxin,panfeng}@comp.nus.edu.sg

Jiong Yang
Dept. of Computer Science,
University of Illinois, Urbana Champaign
jioyang@cs.uiuc.edu

ABSTRACT

Microarray datasets typically contain large number of columns but small number of rows. Association rules have been proved to be useful in analyzing such datasets. However, most existing association rule mining algorithms are unable to efficiently handle datasets with large number of columns. Moreover, the number of association rules generated from such datasets is enormous due to the large number of possible column combinations.

In this paper, we describe a new algorithm called FARMER that is specially designed to discover association rules from microarray datasets. Instead of finding individual association rules, FARMER finds **interesting rule groups** which are essentially a set of rules that are generated from the same set of rows. Unlike conventional rule mining algorithms, FARMER searches for interesting rules in the row enumeration space and exploits all user-specified constraints including minimum support, confidence and chi-square to support efficient pruning. Several experiments on real bioinformatics datasets show that FARMER is orders of magnitude faster than previous association rule mining algorithms.

1. INTRODUCTION

With recent advances in DNA chip-based technologies, we can now measure the expression levels of thousands of genes in cell simultaneously resulting in a large amount of high-dimension data. These microarray datasets typically have a large number of columns but a small number of rows. For example, many gene expression datasets may contain up to 10,000-100,000 columns but only 100-1000 rows.

Recent studies have shown that association rules are very useful in the analysis of microarray data. Due to their relative simplicity, they are more easily interpreted by biologists. Association rules can be applied in the following two scenarios: (1) it is shown in [9, 13] that classifiers built from association rules are rather accurate in identifying cancerous cell; (2) it is suggested in [7] that association rules can be used to build gene networks since they can capture the associations among genes.

In this paper, we focus on a special type of association rule which takes the form of $LHS \rightarrow C$, where LHS is a set of items and C is a class label. We use the term "support of A" to refer to the number of rows containing A in the database and denote this number as $sup(A)$. The probability of the rule being true is referred to as "the confidence of the rule" and is computed as $sup(LHS \cup C)/sup(LHS)$. The number of rows in the database that match the rule is defined as "the support of the rule". User-specified constraints such as minimum support (a statement of generality) and minimum confidence (a statement of predictive ability) are often imposed on mining such association rules.

Microarray datasets pose a great challenge for existing rule mining algorithms in both runtime and the number of discovered rules. While there are a large number of algorithms that have been developed for association rule mining [1, 11, 18, 23], their basic approaches are all column enumeration in which combinations of columns are tested systematically to search for association rules. Such an approach is unsuitable for microarray datasets. This is because if i is the maximum length of a row in a dataset, the search space based on column enumeration could be as large as 2^i. Previous column enumeration methods work well for datasets with small average row length (usually $i < 100$). However, for micorarray datasets, i can be in the range of tens of thousands. These high-dimension bioinformatics datasets with thousands of columns render most of the existing algorithms impractical.

On the other hand, the number of rows in such datasets is typically in the order of hundreds to a thousand. If m is the number of rows, the size of the row enumeration space will be 2^m. In our application domain (e.g., microarray datasets), the size of the row enumeration space is much less than the size of the column enumeration space. Therefore, it seems reasonable to devise the algorithm that does not perform column enumeration but *row enumeration*. To the best of our knowledge, none of existing studies investigate the possibilities of discovering rules by row enumeration.

A large number of long frequent itemsets may be discovered from microarray datasets due to their large number of columns. As a result, large number of association rules may be generated due to the combinatorial explosion of frequent

*Contact Author

†This work was supported in part by NUS ARF grant R252-000-121-112 and R252-000-142-112.

SIGMOD 2004 June 13-18, 2004, Paris, France.

itemsets [3]. For example, given a dataset with one row, five columns and one class label: $\{a, b, c, d, e, Cancer\}$, we could have 31 rules of the form $LHS \rightarrow Cancer$ since any combination of a, b, c, d and e could be the LHS for the rule. These 31 rules all cover the same row and have the same confidence (100%). Such a large set of rules contains a lot of redundancy and is difficult to interpret. Instead of generating all 31 rules, we propose to discover these rules as a rule group whose consequent is Cancer, and which can be identified by a unique **upper bound** plus a set of **lower bounds**. The upper bound of a rule group is the rule with the most specific LHS among the rules. In our example, the upper bound rule is $abcde \rightarrow Cancer$. The lower bounds of the rule group are the rules with the most general LHS in the rule group. For our example, the rule group has 5 lower bounds ($a \rightarrow Cancer$, $b \rightarrow Cancer$, $c \rightarrow Cancer$, $d \rightarrow Cancer$, and $e \rightarrow Cancer$). Given the upper bound and the lower bounds of the rule group, other rules within the group can be easily derived.

We further reduce the number of rules by finding interesting rule groups only. Consider two rules $abcd \rightarrow Cancer$ with confidence 90% and $ab \rightarrow Cancer$ with confidence 95%, it is obvious that ab is a better indicator of $Cancer$ since $ab \rightarrow Cancer$ has a higher confidence and all rows covered by $abcd \rightarrow Cancer$ must be covered by $ab \rightarrow Cancer$. With $ab \rightarrow Cancer$, rule $abcd \rightarrow Cancer$ is not interesting[1].

In this paper, we describe a novel algorithm FARMER [2], that is specially designed to mine interesting rule groups from microarray datasets. FARMER discovers upper bounds of interesting rule groups by performing depth-first row-wise enumeration instead of the usual column-wise approach taken by existing rule mining algorithms. This basic idea is combined with efficient search pruning strategies based on user-specified thresholds (minimum support, minimum confidence and minimum chi-square value), yielding a highly optimized algorithm. We also describe an efficient algorithm for computing the lower bounds. Our experiments show that FARMER substantially outperforms other rule mining algorithms described in [2], [23](CHARM) and [21](CLOSET+) To further illustrate the usefulness of the discovered interesting rule groups in biology, we build a simple classifier based on these interesting rule groups, which outperforms the well-known CBA [14] and SVM [12] on 5 real-life datasets.

The rest of this paper is organized as follows: In the next section, we will introduce some preliminaries and give our problem definitions. The FARMER algorithm will be explained in Section 3. Experimental results will be given in Section 4 on real-life microarray datasets. Section 5 introduces some of the related work for this paper. We will conclude our discussion in Section 6.

2. PRELIMINARY

In this section, we introduce some basic notations and concepts that are useful for further discussion.

2.1 The Basics

Dataset: the dataset (or table) D consists of a set of rows, R=$\{r_1, ..., r_n\}$. Let I=$\{i_1, i_2, ..., i_m\}$ be the complete set of items of D, and $C = \{C_1, C_2, ..., C_k\}$ be the complete set of class labels of D, then each row $r_i \in R$ consists of one or more items from I and a class label from C.

As an example, Figure 1(a) shows a dataset where items are represented with alphabets from 'a' to 't'. There are altogether 5 rows, r_1,...,r_5, in the dataset, the first three of which are labeled C while the other two are labeled $\neg C$. To simplify the notation, we use the *row id set* to represent a set of rows and the *item id set* to represent a set of items. For instance, "234" denotes the row set $\{r_2, r_3, r_4\}$, and "acf" denotes the itemset $\{a, c, f\}$.

Given a set of items $I' \subseteq I$, we define the **row support set**, denoted $\mathcal{R}(I') \subseteq R$, as the largest set of rows that contain I'. Likewise, given a set of rows $R' \subseteq R$, we define **item support set**, denoted $\mathcal{I}(R') \subseteq I$, as the largest set of items that are common among the rows in R'.

EXAMPLE 1. $\mathcal{R}(I')$ **and** $\mathcal{I}(R')$
Consider again the table in Figure 1(a). Let I' be the itemset $\{a, e, h\}$, then $\mathcal{R}(I') = \{r_2, r_3, r_4\}$. Let R' be the row set $\{r_2, r_3\}$, then $\mathcal{I}(R')$=$\{a, e, h\}$ since this is the largest itemset that occurs in both r_2 and r_3. □

Association Rule: an **association rule** γ, or just **rule** for short, from dataset D takes the form of $A \rightarrow C$, where $A \subseteq I$ is the antecedent and C is the consequent (here, it is a class label). The **support** of γ is defined as the $|\mathcal{R}(A \cup C)|$, and its **confidence** is $|\mathcal{R}(A \cup C)|/|\mathcal{R}(A)|$. We denote the antecedent of γ as $\gamma.A$, the consequent as $\gamma.C$, the support as $\gamma.sup$, the confidence as $\gamma.conf$ and the chi-square value is $\gamma.chi$.

As discussed in the introduction, in real biological applications, people are often interested in rules with a specified consequent C that meet specified thresholds, like minimum support and minimum confidence.

2.2 Interesting Rule Groups (IRGs)

The interesting rule group is a concept which helps to reduce the number of rules discovered by identifying rules that come from the same set of rows and clustering them conceptually into one entity.

DEFINITION 2.1. **Rule Group**
Let D be a dataset with itemset I and C be a specified class label. $G = \{A_i \rightarrow C | A_i \subseteq I\}$ is a rule group with antecedent support set R and consequent C, iff (1) $\forall A_i \rightarrow C \in G$, $\mathcal{R}(A_i) = R$, and (2) $\forall \mathcal{R}(A_i) = R$, $A_i \rightarrow C \in G$. Rule $\gamma_u \in G$ (γ_u: $A_u \rightarrow C$) is an **upper bound** *of G iff there exists no $\gamma' \in G$ (γ':$A' \rightarrow C$) such that $A' \supset A_u$. Rule $\gamma_l \in G$ (γ_l: $A_l \rightarrow C$) is a* **lower bound** *of G iff there exists no $\gamma' \in G$ (γ': $A' \rightarrow C$) such that $A' \subset A_l$.* □

LEMMA 2.1. *Given a rule group G with the consequent C and the antecedent support set R, it has a unique upper bound γ (γ: $A \rightarrow C$).*
Proof: *Assume there exists another upper bound $\gamma'(A' \rightarrow C) \in G$ such that $A' \neq A$ and $A' \not\subseteq A$. Let $A'' = A \cup A'$. Because of $\mathcal{R}(A') = \mathcal{R}(A) = R$, we get $\mathcal{R}(A'') = R$, and then $A'' \rightarrow C \in G$ and $A'' \supset A$. Therefore, $\gamma(A \rightarrow C)$ cannot be an upper bound of G. So the upper bound of a rule group must be unique.* □

Based on lemma 2.1, a rule group G can be represented with its unique upper bound γ_u.

[1]Rules like $abcd \rightarrow Cancer$ are simply pruned off in methods like CBA [14] when they are building classifier with association rules.

[2]FARMER stands for Finding Interesting Association Rule Groups by Enumeration of Rows.

i	r_i	class
1	a,b,c,l,o,s	C
2	a,d,e,h,p,l,r	C
3	a,c,e,h,o,q,t	C
4	a,e,f,h,p,r	¬C
5	b,d,f,g,l,q,s,t	¬C

(a) Example Table

i_j	$R(i_j)$	
	C	$\neg C$
a	1,2,3	4
b	1	5
c	1,3	
d	2	5
e	2,3	4
f		4,5
g		5
h	2,3	4
l	1,2	5
o	1,3	
p	2	4
q	3	5
r	2	4
s	1	5
t	3	5

(b) Transposed Table, TT

Figure 1: Running Example

i_j	$R(i_j)$	
	C	$\neg C$
a	1,2,3	4
e	2,3	4
h	2,3	4

Figure 2: $TT|_{\{2,3\}}$

EXAMPLE 2. **Rule Group**
A running example is shown in Figure 2 in which $\mathcal{R}(\{e\}) = \mathcal{R}(\{h\}) = \mathcal{R}(\{ae\}) = \mathcal{R}(\{ah\}) = \mathcal{R}(\{eh\}) = \mathcal{R}(\{aeh\}) = \{r_2, r_3, r_4\}$. *They make up a rule group* $\{e \to C, h \to C, ..., aeh \to C\}$ *of consequent C, with the upper bound* $aeh \to C$ *and the lower bounds* $e \to C$ *and* $h \to C$. □

It is obvious that all rules in the same rule group have the same support, confidence and chi-square value since they are essentially derived from the same subset of rows. Based on the upper bound and all the lower bounds of a rule group, we can identify its remaining members according to the lemma below.

LEMMA 2.2. *Suppose rule group G with the consequent C and antecedent support set R has an upper bound* $A_u \to C$ *and a lower bound* $A_l \to C$. *Rule* $\gamma(A \to C)$, *where* $A \subset A_u$ *and* $A \supset A_l$, *must be a member of G.*
Proof: *Since* $A \subset A_u$, $\mathcal{R}(A) \supseteq \mathcal{R}(A_u)$. *Likewise,* $\mathcal{R}(A) \subseteq \mathcal{R}(A_l)$. *Since* $\mathcal{R}(A_l) = \mathcal{R}(A_u) = R$, $\mathcal{R}(A) = R$. *So* $\gamma(A \to C)$ *belongs to G.* □

DEFINITION 2.2. **Interesting Rule Group (IRG)**
A rule group G with upper bound γ_u *is an interesting rule group iff for any rule group with upper bound* $\gamma'_u \subset \gamma_u$, $\gamma'_u.conf < \gamma_u.conf$. *For brevity, we will use the abbreviation* ***IRG*** *to refer to interesting rule group.* □

Our algorithm FARMER is designed to find IRGs that satisfy user-specified constraints including *minimum support*, *minimum confidence* and *minimum chi-square value* [3]. FARMER finds the upper bounds of all IRGs first, and then gathers their lower bounds. This makes it possible for users to recognize all the rule group members as and when they want to.

3. THE FARMER ALGORITHM

[3]Other constraints such as lift, conviction, entropy gain, gini and correlation coefficient can be handled similarly

To illustrate our algorithm, we first give a running example (Figure 1). Table TT (Figure 1(b)) is a transposed version of the example table (Figure 1(a)). In TT, the items become the row ids while the row ids become the items. A row id r_m in the original table will appear in tuple i_n of TT if and only if the item i_n occurs in the row r_m of the original table. For instance, since item d occurs in row r_2 and r_5 of the original table, row ids "2" and "5" occur in tuple d of TT. To avoid confusion, we hereafter refer to the rows in the transposed table as **tuples** while referring to those in the original table as **rows**.

We provide a conceptual explanation of FARMER algorithm to discover upper bounds of interesting rule groups in Section 3.1, the pruning strategies in Section 3.2, and the implementation details in Section 3.3. In Section 3.4, we describe subroutine MineLB of FARMER to discover the lower bounds of interesting rule groups.

3.1 Enumeration

Unlike existing column-wise rule mining algorithms which perform their search by enumeration of columns [18], FARMER performs search by enumeration of row sets to find interesting rule groups with consequent C. Figure 3 illustrates the enumeration tree which represents the search of FARMER **conceptually** for the interesting rule groups in the absence of any pruning strategies. Each node X of the enumeration tree corresponds to a combination of rows R' and is labeled with $\mathcal{I}(R')$ that is the antecedent of the upper bound of a rule group identified at this node. For example, node "12" corresponds to the row combination $\{r_1, r_2\}$ and "al" indicates that $\mathcal{I}(\{r_1, r_2\}) = \{a, l\}$. An upper bound $al \to C$ can be discovered at node "12". This is correct because of the following lemma.

LEMMA 3.1. *Let X be a subset of rows from the original table, then* $\mathcal{I}(X) \to C$ *must be the upper bound of the rule group G whose antecedent support set is* $\mathcal{R}(\mathcal{I}(X))$ *and consequent is C.*
Proof: *First, according to Definition 2.1,* $\mathcal{I}(X) \to C$ *belongs to rule group G with antecedent support set* $\mathcal{R}(\mathcal{I}(X))$ *and consequent C. Second, assume that* $\mathcal{I}(X) \to C$ *is not the upper bound of G, then there must exist an item i such that* $i \notin \mathcal{I}(X)$, *and* $\mathcal{I}(X) \cup \{i\} \to C$ *belongs to G. So we get* $\mathcal{R}(\mathcal{I}(X)) = \mathcal{R}(\mathcal{I}(X) \cup \{i\})$. *Since rows in X contain all items of* $\mathcal{I}(X)$, *we get* $X \subseteq \mathcal{R}(\mathcal{I}(X))$, *and then* $X \subseteq \mathcal{R}(\mathcal{I}(X) \cup \{i\})$. *This means that i is also found in every row of X, which contradicts the definition that* $\mathcal{I}(X)$ *is the largest set of items that are found in every row of X. So* $\mathcal{I}(X) \to C$ *is the upper bound of the rule group with antecedent support set* $\mathcal{R}(\mathcal{I}(X))$. □

FARMER performs a depth-first search on the enumeration tree by moving along the edges of the tree. By imposing an order $\mathcal{ORD}$, in which the rows with consequent C are ordered BEFORE the rows without consequent C(this is done to support efficient pruning which will be explained later), we are able to perform a systematic search by enumerating the combinations of rows based on the order $\mathcal{ORD}$. For example, let "$1 \prec 2 \prec 3 \prec 4 \prec 5$" according to $\mathcal{ORD}$, the order of search in Figure 3 will be {"1", "12", "123", "1234", "12345", "1235",...,"45", "5"} in absence of any optimization and pruning strategies. Note that the order also serves for confidence pruning purpose (explained in section 3.2.3).

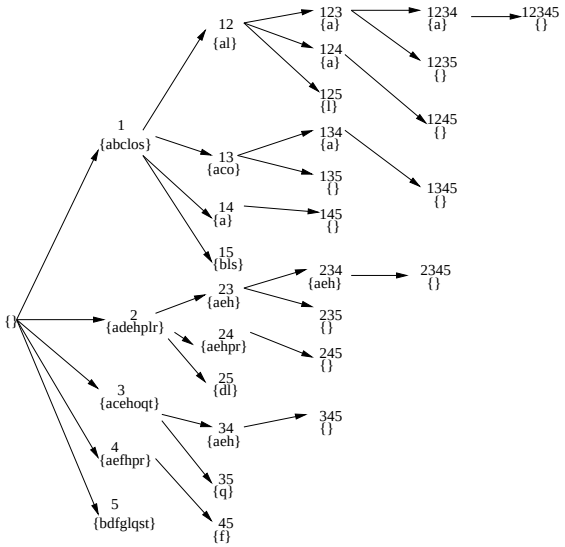


Figure 3: The Row Enumeration Tree.

Next, we prove that the complete rule groups can be discovered by a complete row enumeration.

LEMMA 3.2. *By enumerating all possible row combinations on the row enumeration tree, we can obtain the complete set of upper bounds and the corresponding complete set of rule groups in the dataset.*
Proof: *With Lemma 2.1, we know that each rule group can be represented by a unique upper bound. Based on the definition of rule group (Definition 2.1), all possible antecedent support sets of rule groups can be obtained by enumerating all possible row combinations. Each antecedent support set X corresponds to a rule group with upper bound "$\mathcal{I}(X) \rightarrow C$". Hence the proof.* □

It is obvious that a complete traversal of the row enumeration tree is not efficient. Various pruning techniques will be introduced to prune off unnecessary searches in the next section. We will next introduce the framework of our algorithm for discovering the upper bounds of rule groups. We first introduce two concepts.

DEFINITION 3.1. **Conditional Transposed Table ($TT|_X$)** *Given the transposed table TT (used at the root of the enumeration tree), a X-conditional transposed table ($TT|_X$) at node X (X is the row combination at this node) is a subset of tuples from TT such that for each tuple t of TT that $t \supseteq X$, there exists a tuple $t' = t$ in $TT|_X$.* □

EXAMPLE 3. *Let TT be the transposed table in Figure 1(b) and let $X = \{2, 3\}$. The X-conditional transposed table, $TT|_X$ is shown in Figure 2.* □

DEFINITION 3.2. **Enumeration Candidate List ($TT|_X.E$)** *Let $TT|_X$ be the X-conditional transposed table and $r_{min} \in X$ be the row id with the lowest $\mathcal{ORD}$ order in row combination X. Let $E_P = \{r | r \succ_{ORD} r_{min} \ \wedge \ r \in \mathcal{R}(C)\}$ (all rows of consequent C ordered after r_{min}), and $E_N = \{r | r \succ_{ORD} r_{min} \ \wedge \ r \in \mathcal{R}(\neg C)\}$ (all rows with class C ordered after r_{min}). The enumeration candidate list for $TT|_X$, denoted as $TT|_X.E$, is defined to be $E_P \cup E_N$.* □

Notation	Description
$TT\|_X.E$	enumeration candidates;
$TT\|_X.E_P$	enumeration candidates with label C;
$TT\|_X.E_N$	enumeration candidates without label C;
$TT\|_X.Y$	enumeration candidates that occur in each tuple of $TT\|_X$.

Figure 4: Notations for Conditional Transposed Table

In the rest of this paper, we will use the notations in Figure 3.1 to describe various operations on the conditional transposed table, $TT|_X$.

Our formal algorithm is shown in Figure 5. FARMER involves recursive computations of conditional transposed tables by performing a depth-first traversal of the row enumeration tree. Each computed conditional table represents a node in the enumeration tree of Figure 3. For example, the $\{2, 3\}$-conditional table is computed at node "23". After initialization, FARMER calls the subroutine $MineIRGs$ to recursively generate X-conditional tables.

The subroutine $MineIRGs$ takes in four parameters at node X: $TT'|_X$, sup_p, sup_n and IRG. $TT'|_X$ is the X-conditional transposed table at node X with enumeration candidates $TT'|_X.E_P$ and $TT'|_X.E_N$. sup_p is the number of identified rows that contain $\mathcal{I}(X) \cup C$ while sup_n is the number of identified rows that contain $\mathcal{I}(X) \cup \neg C$ before scanning $TT'|_X$. IRG stores the upper bounds of interesting rule groups discovered so far.

Steps 1, 2, 4 and 5 in the subroutine $MineIRGs$ perform the pruning. They are extremely important for the efficiency of FARMER Algorithm and will be explained in the next subsection. Step 3 scans the table $TT'|_X$. Step 6 moves on into the next level enumerations in the search tree. Step 7 checks whether $\mathcal{I}(X) \rightarrow C$ is the upper bound of an IRG that satisfies the user-specified constraints before inserting it into IRG. Note that step 7 must be performed after step 6 (the reason will be clear later). We first prove the correctness of the two steps by two lemmas as follows:

LEMMA 3.3. $TT|_X|_{r_i} = TT|_{X+r_i}$, $r_i \in TT|_X.E$. □

Lemma 3.3 is useful for explaining Step 6. It simply states that a $X + r_i$ conditional transposed table can be computed from a X conditional transposed table $TT|_X$ in the next level search after node X.

Lemma 3.1 ensures that at Step 7 only upper bounds of rule groups are possibly inserted into IRG. To determine whether an upper bound γ discovered at node X represents an interesting rule group satisfying user-specified constraints, we need to compare $\gamma.conf$ with all $\gamma'.conf$, where $\gamma'.A \subset \gamma.A$ and γ' satisfies user specified constraints. FARMER ensures that all such γ' have already been discovered and kept in IRG at Step 7 by lemma 3.4 below.

LEMMA 3.4. *Let $\gamma : \mathcal{I}(X) \rightarrow C$ be the upper bound rule discovered at node X. The rule group with upper bound $\gamma' : A' \rightarrow C$ such that $A' \subset \mathcal{I}(X)$ can always be discovered at the descendent nodes of node X or in an earlier enumeration.*
Proof: *Since $A' \subset \mathcal{I}(X)$, and γ' and γ are the upper bounds of two different rule groups, we see $\mathcal{R}(A') \supset \mathcal{R}(\mathcal{I}(X)) \supseteq X$. Let $RS = \{r | r \in \mathcal{R}(A') \ \wedge \ r \notin X\}$ and $r_{min} \in X$ be the row with the lowest $\mathcal{ORD}$ rank in row set X. If $\exists r' \in RS$ such that $r' \prec r_{min}$, then node $\mathcal{R}(A')$ is traversed before node X; otherwise node $\mathcal{R}(A')$ is traversed at a descendent node of node X.* □

Algorithm FARMER
Input: table D, specified consequent C, $minsup$, $minconf$, and $minchi$.
Output: interesting rule groups with consequent C satisfying minimum measure thresholds.
Method:

1. *Initialization:* Let TT be the transposed table of ORD ordered D; $IRG = \emptyset$.
2. *Mine Interesting Rule Groups:* MineIRGs($TT|_\emptyset$, 0, 0, IRG).
3. *Mine Lower Bounds of Interesting Rule Groups:* Optional.

Subroutine: MineIRGs($TT'|_X$, sup_p, sup_n, IRG).
Parameters:

$TT'|_X$: a X-conditional transposed table;

sup_p and sup_n: support parameters;

IRG: the set of discovered interesting rule groups;

Method:

1. **Apply Pruning 2: If** $\mathcal{I}(X) \rightarrow C$ is already identified, **then** return.
2. **Apply Pruning 3: If** prunable with the loose upper bounds of support or confidence, **then** return.
3. Scan $TT'|_X$ and count the frequency of occurrences for each enumeration candidate, $r_i \in TT'|_X.E$,
   Let $U_p \subseteq TT'|_X.E_P$ be the set of rows from $TT'|_X.E_P$ which occur in at least one tuple of $TT'|_X$;
   Let $U_n \subseteq TT'|_X.E_N$ be the set of rows from $TT'|_X.E_N$ which occur in at least one tuple $TT'|_X$;
   Let $Y_p \subset TT'|_X.E_P$ be the set of rows from $TT'|_X.E_P$ found in every tuple of $TT'|_X$;
   Let $Y_n \subset TT'|_X.E_N$ be the set of rows from $TT'|_X.E_N$ found in every tuple of $TT'|_X$;
   $sup_p = sup_p + |Y_P|$ ($|\mathcal{R}(\mathcal{I}(X) \cup C)|$);
   $sup_n = sup_n + |Y_N|$ ($|\mathcal{R}(\mathcal{I}(X) \cup \neg C)|$);
4. **Apply Pruning 3: If** prunable with one of the three tight upper bounds, **then** return.
5. **Apply Pruning 1:** Update enumeration candidate list, $TT'|_X.E_P = U_P - Y_P$, $TT'|_X.E_N = U_N - Y_N$.
6. **for** each $r_i \in TT'|_X.E$ **do**
   if $r_i \in \mathcal{R}(C)$ **then**
   $TT'|_X|_{r_i}.E_P = \{r_j | r_j \in TT'|_X.E_P \wedge r_j \succ_{ORD} r_i\}$; $TT'|_X|_{r_i}.E_N = TT'|_X.E_N$;
   $a = sup_p + 1$; $b = sup_n$;
   else
   $TT'|_X|_{r_i}.E_P = \varnothing$; $TT'|_X|_{r_i}.E_N = \{r_j | r_j \in TT'|_X.E_N \wedge r_j \succ_{ORD} r_i\}$;
   $a = sup_p$; $b = sup_n + 1$;
   $MineIRGs(TT'|_X|_{r_i}, a, b, IRG)$;
7. Let $conf = (sup_p)/(sup_p + sup_n)$;
   If ($sup_p \geq minsup$) $\wedge$ ($conf \geq minconf$) $\wedge$ ($chi(sup_p, sup_p + sup_n) \geq minchi$) **then**
   if $\forall\gamma$, $(\gamma \in IRG) \wedge (\gamma.A \subset \mathcal{I}(X)) \Rightarrow (conf > \gamma.conf)$
   then add upper bound rule $\mathcal{I}(X) \rightarrow C$ into IRG.

Figure 5: The FARMER Algorithm

Step 7 is done after Step 6 to ensure that all descendant nodes of X are explored before determining whether the upper bound rule γ at X is an IRG. Together with Lemma 3.2, we know that the complete and correct set of interesting rule groups will be in IRG.

Note that Step 6 implicitly does some pruning since it is possible that the enumeration candidate list is empty, i.e. $TT'|_X.E = \emptyset$. It can be observed from the enumeration tree that there exist some combinations of rows, X, such that $\mathcal{I}(X) = \emptyset$ (an example is node "134"). This implies that there is no item existing in all the rows in X. When this happens, $TT'|_X.E$ is empty and no further enumeration will be performed.

3.2 Pruning Strategies

We next look at the pruning techniques that are used in FARMER, which are essential for the efficiency. Our emphasis here is to show that our pruning steps do not prune off any interesting rule groups while preventing unnecessary traversals of the enumeration tree. Combining this with our earlier explanations on how all interesting rule groups are enumerated in FARMER without the pruning steps, the correctness of our algorithm will be obvious.

3.2.1 Pruning Strategy 1

Pruning strategy 1 is implemented at Step 5 of MineIRGs by pruning $TT|_X.Y$, the set of enumeration candidate rows that occur in all tuples of the $TT|_X$. We partition $TT|_X.Y$ to two subsets: Y_p with consequent C and Y_n without. The intuitive reason for the pruning is that we obtain the same set of upper bound rules along the branch X WITHOUT such rows. The correctness of such a pruning strategy is due to the following lemma.

LEMMA 3.5. *Let $TT'|_X$ be a X-conditional transposed table. Given any subset R', $R' \subset TT'|_X.E$, we have $\mathcal{I}(X \cup R') = \mathcal{I}(X \cup TT'|_X.Y \cup R')$.*
Proof: *By definition, $\mathcal{I}(X \cup R')$ contains a set of items which occur in every row of $(X \cup R')$. Suppose candidate $y \in TT'|_X.Y$ (y occurs in every tuple of $TT'|_X$), then either $y \in X \cup R'$ (if $y \in R'$) or y occurs in every tuple of the $TT'|_{X \cup R'}$ (if $y \notin R'$). In either case, $\mathcal{I}(X \cup R') = \mathcal{I}(X \cup R' \cup \{y\})$. Thus, $\mathcal{I}(X \cup R') = \mathcal{I}(X \cup TT'|_X.Y \cup R')$.* □

With Lemma 3.5, we can safely delete the rows in $TT'|_X.Y$ from the enumeration candidate list $TT'|_X.E$.

EXAMPLE 4. *Consider $TT|_{\{2,3\}}$, the conditional transposed table in Figure 2. Since enumeration candidate row 4 occurs in every tuples of $TT|_{\{2,3\}}$, we can conclude that $\mathcal{I}(\{2,3\}) = \mathcal{I}(\{2,3,4\}) = \{a, e, h\}$. Thus, we need not traverse node "234" and create $TT|_{\{2,3,4\}}$. Row 4 can be safely deleted from $TT|_{\{2,3\}}.E$.* □

Since $\mathcal{I}(\{2,3,4\}) = \mathcal{I}(\{2,3\})$, the upper bound rule is identified at node "23" and node "234" is redundant. We say that node "234" is compressed to node "23".

We argue here that Lemma 3.4 still holds after applying pruning strategy 1. Without applying pruning strategy 1, for each node X, $A' \rightarrow C$, where $A' \subset \mathcal{I}(X)$, is identified at a node X', which is traversed before node X or is a descendent node of node X. With pruning strategy 1, X' might be compressed to a node X'' ($X'' \subset X'$ *and* $\mathcal{I}(X'') = \mathcal{I}(X') = A'$), and we can see node X'' is either traversed before the subtree rooted at node X, or inside this subtree.

3.2.2 Pruning Strategy 2

This pruning strategy is implemented at Step 1 of MineIRGs. It will stop searching the subtree rooted at node X if the upper bound rule $\mathcal{I}(X) \rightarrow C$ was already discovered previously in the enumeration tree because this implies that any rules to be discovered at the descendants of node X would have been discovered too.

LEMMA 3.6. *Suppose pruning strategy 1 is utilized in the search. Let $TT'|_X$ be the conditional transposed table of the current node X. All upper bounds to be discovered in the subtree rooted at node X must have already been discovered if there exists such a row r' that satisfies the following conditions: (1)$r' \notin X$; (2)$r' \notin TT'|_X.E$; (3)for any ancestor node X_i of node X, $r' \notin TT|_{X_i}.Y$ (pruned by strategy 1); and (4)r' occurs in each tuple of $TT'|_X$.*

Proof: *Let $X = \{r_1, r_2, ..., r_m\}$, where $r_1 \prec_{ORD} r_2 \prec_{ORD} ... \prec_{ORD} r_m$. Suppose that there is a node $X''(X'' = X \cup \{r'\})$, we can have the following properties: (1) $\mathcal{I}(X) = \mathcal{I}(X'')$; (2) $r' \prec_{ORD} r_m$, since $r' \notin TT'|_X.E$ and $r' \notin X$; (3) $TT'|_X.E = TT'|_{X''}.E$.*

X'' is either enumerated or compressed to a node X_C, where $\mathcal{I}(X_C) = \mathcal{I}(X'')$ and $TT'|_{X''}.E \subseteq TT'|_{X_C}.E$. We can prove that either node X'' or node X_C is traversed before node X by considering the following two cases: (1) If $r' \prec_{ORD} r_1$, node X'' or node X_C falls in the subtree rooted at node $\{r'\}$, which is traversed before node X. (2) If row ids in X'' follow the order $r_1 \prec_{ORD} r_2 \prec_{ORD} ... \prec_{ORD} r_t \prec_{ORD} r' \prec_{ORD} r_{t+1} \prec_{ORD} ... \prec_{ORD} r_m$, node X'' or node X_C falls in the subtree rooted at node $X' = \{r_1, ..., r_t, r'\}$, which is also traversed before node X. Because $TT'|_X.E = TT'|_{X''}.E$ and $TT'|_{X''}.E \subseteq TT'|_{X_C}.E$, we can conclude that all upper bounds to be discovered in the subtree rooted at node X must have already been discovered earlier in the subtree rooted at node X'' or node X_C. □

In the implementation of pruning strategy 2, the existence of such a r' can be efficiently detected by a process called **back counting** without scanning the whole $TT'|_X$. Details are explained in section 3.3.

EXAMPLE 5. *Consider node "23" in Figure 3 where the upper bound rule $\{a, e, h\} \rightarrow C$ is identified for the first time. When it comes to node "34", we notice that row "2" occurs in every tuple of $TT|_{\{3,4\}}$, "2" $\notin TT|_{\{3,4\}}.E$, and "2" $\notin TT|_{\{3\}}.Y$. So we conclude that all upper bounds to be discovered down node "34" have already been discovered before ($\mathcal{I}(\{3,4\}) = \mathcal{I}(\{2,3\}) = \{a, e, h\}$. $\mathcal{I}(\{3,4,5\}) = \emptyset$). We can prune the search down node "34".* □

3.2.3 Pruning Strategy 3

Pruning strategy 3 performs pruning by utilizing the user-specified thresholds, *minsup*, *minconf* and *minchi*. We estimate the upper bounds of the measures for the subtree rooted at the current node X. If the estimated upper bound at X is below the user-specified threshold, we stop searching down node X. A important thing to note here is that our pruning strategy is highly dependent on the order ORD which rank all rows with consequent C before rows with consequent $\neg C$.

Pruning strategy 3 consists of 3 parts: pruning using confidence upper bound, pruning using support upper bound and pruning using chi-square upper bound. This strategy is executed separately at Step 2 and Step 4 (Figure 5). At Step 2, we will perform pruning using the two loose upper bounds of support and confidence that can be calculated BEFORE scanning $TT'|_X$. At Step 4 we calculate the three tight upper bounds of support, confidence and chi-square value AFTER scanning $TT'|_X$.

For clarity, we will use the notations in Figure 3.2.3 to explain our pruning strategy here.

Notation	Description
X	the current enumeration node;
γ	the upper bound rule $\mathcal{I}(X) \rightarrow C$ at node X;
X'	the immediate parent node of X;
γ'	the upper bound rule $\mathcal{I}(X') \rightarrow C$ at node X';
r_m	a row id such that $TT|_X = TT|_{X'}|_{r_m}$;

Figure 6: Notations for Search Pruning

Pruning Using Support Upper Bound

We have two support upper bounds for the rule groups identified at the subtree rooted at node X: the tight support upper bound U_{s1} (after scanning $TT'|_X$) and the loose support upper bound U_{s2} (before scanning $TT'|_X$). If the estimated upper bound is less than the minimum support *minsup*, the subtree can be pruned.

If r_m has consequent C:
$U_{s1} = \gamma'.sup + 1 + MAX(|TT'|_X.E_P \cap t|)$, $t \in TT'|_X$;
$U_{s2} = \gamma'.sup + 1 + |TT'|_X.E_P|$;

If r_m has consequent $\neg C$ then $U_{s1} = U_{s2} = \gamma'.sup$;

LEMMA 3.7. *U_{s1} and U_{s2} are the support upper bounds for the upper bound rules discovered in subtree rooted at node X.*

Proof: *Because of the $\mathcal{ORD}$ order (Definition 3.2), if the consequent of r_m is $\neg C$, the enumeration candidates of nodes down node X will also have consequent $\neg C$. The support can not increase down node X, so the support of upper bounds discovered in the subtree rooted at node X is less than $\gamma'.sup$. If r_m has consequent C, for node X and its descendent nodes, the maximum increase of support from $\gamma'.sup$ must come from the number of enumeration candidates with consequent C ($|TT'|_X.E_P|$) at node X plus 1 (1 for r_m)(U_{s2}), or more strictly, from the maximum number of enumeration candidates with consequent C within a tuple of $TT'|_X$ ($MAX(|TT'|_X.E_P \cap t|)$, $t \in TT'|_X$) plus 1 (U_{s1}).* □

Note that we need to scan $TT'|_X$ to get U_{s1} while U_{s2} can be obtained directly from the parameters sup_p and X passed by the parent node.

Pruning Using Confidence Upper Bound

Similarly, we estimate two confidence upper bounds for the subtree rooted at node X, the tight confidence upper bound U_{c1} and the loose confidence upper bound U_{c2}. If the estimated upper bound is less than minimum confidence *minconf*, the subtree rooted at node X can be pruned.

Given U_{s1} and U_{s2}, the two confidence upper bounds of subtree rooted at node X, U_{c1}(tight) and U_{c2}(loose), are:
$U_{c1} = U_{s1}/(U_{s1} + |\mathcal{R}(\gamma.A \cup \neg C)|)$;
$U_{c2} = U_{s2}/(U_{s2} + |\mathcal{R}(\gamma'.A \cup \neg C)|)$ (r_m has consequent C);
$U_{c2} = U_{s2}/(U_{s2} + |\mathcal{R}(\gamma'.A \cup \neg C)| + 1)$ (r_m has consequent $\neg C$).

LEMMA 3.8. *U_{c1} and U_{c2} are the confidence upper bounds for the rules discovered in the subtree rooted at node X.*

Proof: *For a rule γ'' discovered in subtree rooted at node X, its confidence is computed as $|\mathcal{R}(\gamma''.A \cup C)|/(|\mathcal{R}(\gamma''.A \cup C)| + |\mathcal{R}(\gamma''.A \cup \neg C)|)$. This expression can be simplified as $x/(x+y)$, where $x = |\mathcal{R}(\gamma''.A \cup C)|$ and $y = |\mathcal{R}(\gamma''.A \cup \neg C)|$. This value is maximized by choosing the maximum value for x (U_{s1} and U_{s2}) and minimum value for y. Suppose rule γ is discovered at node X. For any rule γ'' discovered under the enumeration tree under node X, $\gamma''.A \subset \gamma.A$ because of pruning strategy 1, so we can see $|\mathcal{R}(\gamma''.A \cup \neg C)| \geq |\mathcal{R}(\gamma.A \cup$*

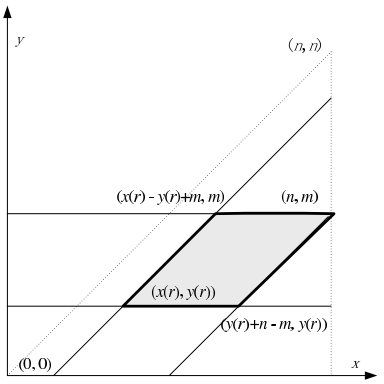


Figure 7: The Possible Chi-square Variables

$\neg C)|$. *Thus the minimum value for* y *is* $|\mathcal{R}(\gamma.A \cup \neg C)|$ *or loosely at* $|\mathcal{R}(\gamma'.A \cup \neg C)| + 1$*(if* r_m *has no consequent* C*) and* $|\mathcal{R}(\gamma'.A \cup \neg C)|$ *(if* r_m *has consequent* C*).* □

EXAMPLE 6. *Suppose minimum confidence* $minconf = 95\%$*. At node "134", the discovered upper bound rule is "*$a \rightarrow C$*" with confidence* $0.75 < 0.95$*. Since row 4 has no consequent* C*, any descendent enumeration will only reduce the confidence. Thus we can stop next level searching.*

Pruning Using Chi-Square Upper Bound

The chi-square value of an association rule is the normalized deviation of the observed values from the expected values.

Let γ be a rule in the form of $A \rightarrow C$ of dataset D, n be the number of rows in D, and m be the number of instances with consequent C in D. The four observed values for chi-square value computation are listed in the following table. For example, $O_{A\neg C}$ represents the number of rows that contain A but do not contain C. Let $x = O_A$ and $y = O_{AC}$. Since m and n are constants, the chi-square value is determined by x and y only and we get chi-square function $chi(x, y)$.

	C	$\neg C$	Total
A	$O_{AC} = y$	$O_{A\neg C}$	$O_A = x$
$\neg A$	$O_{\neg AC}$	$O_{\neg A\neg C}$	$O_{\neg A} = n - x$
Total	$O_C = m$	$O_{\neg C} = n - m$	n

The following lemma gives an estimation of upper bound of chi square value for rules down the node X.

LEMMA 3.9. *Suppose rule* γ *is discovered at enumeration node* X*. The chi-square upper bound for the upper bound rules discovered at the subtree rooted as node* X *is:* $max\{chi(x(\gamma)-y(\gamma)+m, m), chi(y(\gamma)+n-m, y(\gamma)), chi(x(\gamma), y(\gamma))\}$.

Proof: *Suppose rule* γ' $(A' \rightarrow C)$ *is identified in the subtree rooted at node* X*,* $x' = O_{A'}$ *and* $y' = O_{A'C}$*. Since* $O(A) = |\mathcal{R}(A)|$ *and* $A' \subset A$*. The followings are satisfied.*

1) $x \leq x' \leq n$ $(|\mathcal{R}(A)| \leq |\mathcal{R}(A')|)$

2) $y \leq y' \leq m$ $(|\mathcal{R}(A \cup C)| \leq |\mathcal{R}(A' \cup C)|)$

3) $y' \leq x'$ $(|\mathcal{R}(A' \cup C)| \leq |\mathcal{R}(A')|)$

4) $n - m \geq x' - y' \geq x - y$ $(|\mathcal{R}(A' \cup \neg C)| \geq |\mathcal{R}(A \cup \neg C)|)$

The value pair $(x'(\gamma'), y'(\gamma'))$ *falls in the gray parallelogram* $(x(\gamma), y(\gamma))$, $(x(\gamma)-y(\gamma)+m, m)$, (n, m), $(y(\gamma)+n-m, y(\gamma)$ *(Figure 7). Since the chi-square function* $chi(x, y)$ *is a convex function [15], which is maximized at one of its vertexes, and* $chi(n, m) = 0$ *(please refer to [15]), we only need to consider the remaining three vertexes.* □

3.3 Implementation

In the implementation of FARMER, we use memory pointers [4] to point at the relevant tuples in the in-memory transposed table to simulate the conditional transposed table. Our implementation assumes that despite the high dimensionality, the microarray datasets that we are trying to handle are still sufficiently small to be loaded completely into the main memory. This is true for many gene expression datasets which have small number of rows.

Following is the running example. Suppose the current node is node "1" (Figure 8(a)), and $minsup = 1$. The in-memory transposed table is shown on the right hand side of the figure. Memory pointers are organized into **conditional pointer lists**.

In Figure 8(a), the "1"-conditional pointer list (at the top left corner of the figure) has 6 entries in the form of $< f_i, Pos >$ which indicates the tuple (f_i) that contains r_1 and the position of r_1 within the tuple (Pos). For example, the entry $< a, 1 >$ indicates that row r_1 is contained in the tuple 'a' at position 1. We can extend the "1"-conditional transposed table $TT'|_{\{1\}}$ by following the Pos. During one full scan of the transposed table, FARMER also generates the conditional pointer lists for other rows (i.e. r_2, r_3, r_4 and r_5). However, the generated "2"-conditional pointer list is slightly different in that it contains an entry for each tuple that contains r_2 BUT NOT r_1. For example, although the tuple 'a' contains r_2, it does not appear in the "2"-conditional pointer list. It will be inserted subsequently as we will see later.

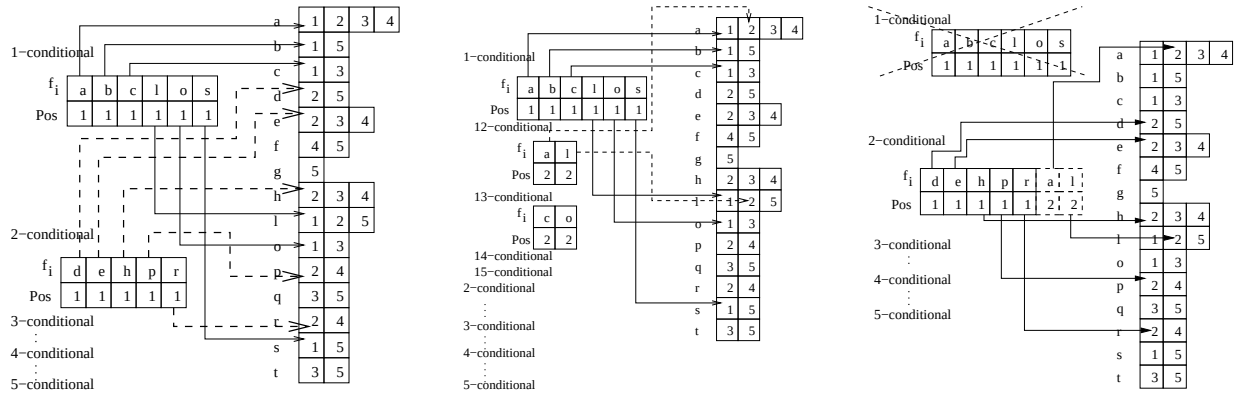
A further scan through the "1"-conditional pointer list will allow us to generate the "12", "13", "14" and "15" conditional pointer lists. Figure 8(b) shows the state of memory pointers when we are processing node $\{1, 2\}$.

Finally, we show the state of conditional pointer lists after node $\{1\}$ and all its descendants have been processed (Figure 8(c)). Since all enumerations involving row r_1 have been either processed or pruned off, the entries in the "1"-conditional pointer list are moved into the remaining conditional pointer lists. The entries in the "2"-conditional pointer list will be moved to the other conditional pointer lists after node $\{2\}$ and its descendants are processed, and so on.

Throughout all the enumerations described above, we need to implement our three pruning strategies. The implementation of strategies 1 and 3 is straightforward. For pruning strategy 2, we do a back scan through the conditional list to see whether there exists some row that satisfies the condition of Lemma 3.6. For example at node "2" in Figure 8(c), we scan from the position of each pointer to the head of each tuple, instead of scanning the transposed table from the position of each pointer to the end of each tuple. In this example, there is no row that satisfies the pruning condition of Lemma 3.6. Such an implementation is proven to be efficient for our purpose as shown in our experiments.

3.4 Finding Lower Bounds

In this section, we describe the algorithm, MineLB, which is designed to find the lower bounds of a rule group. Since a rule group has a unique upper bound and the consequent of a rule group is fixed, the problem can be regarded as generating the lower bounds for the antecedent of the upper bound rule. This antecedent could be regarded as a closed set (Definition 3.3) and the problem can be solved as long



(a) Node {1} (b) Node {1,2} (c) Node {2}

Figure 8: Conditional Pointer Lists

as we can generate the lower bounds of a closed set.

DEFINITION 3.3. **Closed Set**
*Let D be the dataset with itemset I and row set R. A ($A \subseteq I$) is a closed set of dataset D, iff there is no $A' \supset A$ such that $\mathcal{R}(A) = \mathcal{R}(A')$. A_l, $A_l \subseteq A$, is a **lower bound** of closed set A, iff $\mathcal{R}(A_l) = \mathcal{R}(A)$ and there is no $A' \subset A_l$ such that $\mathcal{R}(A') = \mathcal{R}(A)$.* □

MineLB is an incremental algorithm that is initialized with one closed set A, the antecedent of an upper bound rule, $A \rightarrow C$ for a rule group. It then updates the lower bounds of A incrementally whenever a new closed set A' is added, where $A' \subset A$ and A' is the antecedent of the newly added upper bound $A' \rightarrow C$. In this way, MineLB keeps track of the latest lower bounds of A. MineLB is based on the following lemma.

LEMMA 3.10. *Let A be the closed set whose lower bounds will be updated recursively and F be the set of closed sets that are already added. Let $A.\Gamma$ be the current collection of lower bounds for A. When a new closed set $A' \subset A$ is added, $A.\Gamma$ is divided into two groups, $A.\Gamma 1$ and $A.\Gamma 2$, where $A.\Gamma 1 = \{l_i | l_i \in A.\Gamma \wedge l_i \subseteq A'\}$, $A.\Gamma 2 = A.\Gamma - A.\Gamma 1$. Then the newly generated lower bounds of A must be in the form of $l_1 \cup \{i\}$, where $l_1 \in A.\Gamma 1$, $i \in A - A'$.*
Proof:
Suppose l is a newly generated lower bound of A.
(1) we prove $l \supset l_1$. Since $\mathcal{R}(l) = \mathcal{R}(A)$ (Definition 2.1) before A' is added, there must exist a $l_i \in A.\Gamma$ such that $l_i \subset l \subset A$. If $l_i \in A.\Gamma 2$, l can not be a new lower bound, since $l_i \in A.\Gamma 2$ is still a lower bound of A after A' is added. So $l \supset l_1$, $l_1 \in A.\Gamma 1$.
(2) Obviously, the newly generated lower bound must contain at least one item from the set $(A - A')$.
(3) $l' = l_1 \cup \{i\}$ is a bound for A after adding A', where $l_1 \in A.\Gamma 1$, $i \in A$ and $i \notin A'$. Before A' is added, $l' = l_1 \cup \{i\}$ is a bound, so for any $X \in F$, $l' \not\subseteq X$. After A' is added, $l' \not\subseteq A'$ because $i \notin A'$. So, $l' = l_1 \cup \{i\}$ is a new bound for A after adding A'.
Based on (1), (2) and (3), we come to the conclusion that the newly generated lower bound for A after inserting A' takes the form of $l_1 \cup \{i\}$, where $l_1 \in A.\Gamma 1$ and $i \in (A - A')$. □

Itemset $l_1 \cup \{i\}$ described in Lemma 3.10 is a candidate lower bound of A after A' is added. If $l_1 \cup \{i\}$ does not cover any $l_2 \in A.\Gamma 2$ and other candidates, $l_1 \cup \{i\}$ is a new lower bound of A after A' is added. MineLB adopts bit vector for the above computation. Thus $A.\Gamma$ can be updated efficiently. The detailed algorithm is illustrated in Figure 9.

We can ensure that the closed sets (those that cover all the longest closed set $A' \subset A$) obtained at Step 2 are sufficient for the correctness of MineLB because of Lemma 3.11

LEMMA 3.11. *If a closed set $A1 \subset A$ is already added and the collection of A's lower bounds, $A.\Gamma$, is already updated, $A.\Gamma$ will not change after adding closed set $A2$, $A2 \subset A1$.*
Proof:
After $A1 \subset A$ is added, $A.\Gamma$ is updated so that no $l_i \in A.\Gamma$ can satisfy $l_i \subseteq A1$. So no $l_i \in A.\Gamma$ can satisfy $l_i \subseteq A2$, $A2 \subset A1$. Since $A2$ will not cover any $l_i \in A.\Gamma$, $A.\Gamma$ will not change, according to Lemma 3.10. □

EXAMPLE 7. **Finding Lower Bound**
Given an upper bound rule with antecedent $A = abcde$ and two rows, $r_1 : abcf$ and $r_2 : cdeg$, the lower bounds $A.\Gamma$ of A can be determined as follows:
1)Initialize the set of lower bounds $A.\Gamma = \{a, b, c, d, e\}$;
2)add "abc" ($= \mathcal{I}(r_1) \cap A$): We get $A.\Gamma 1 = \{a, b, c\}$ and $A.\Gamma 2 = \{d, e\}$. Since all the candidate lower bounds, "ad", "ae", "bd", "be", "cd", "ce" cover a lower bound from $A.\Gamma 2$, no new lower bounds are generated. So $A.\Gamma = \{d, e\}$;
3)add "cde" ($= \mathcal{I}(r_2) \cap A$): We get $A.\Gamma 1 = \{d, e\}$ and $A.\Gamma 2 = \emptyset$. The candidate lower bounds are "ad", "bd", "ae" and "be". Because none of them is covered by another candidate and $A.\Gamma 2 = \emptyset$, $A.\Gamma = \{ad, bd, ae, be\}$. □

4. PERFORMANCE STUDIES

In this section, we will look at both the efficiency of FARMER and the usefulness of the discovered IRGs. All our experiments were performed on a PC with a Pentium IV 2.4 Ghz CPU, 1GB RAM and a 80GB hard disk. Algorithms were coded in Standard C.

Datasets: The 5 datasets are the clinical data on lung cancer (LC)[4], breast cancer (BC) [5], prostate cancer (PC) [6],

[4]http://www.chestsurg.org
[5]http://www.rii.com/publications/default.htm
[6]http://www-genome.wi.mit.edu/mpr/prostate

Subroutine: MineLB(Table:D, upper bound rule: γ).

1. $A = \gamma.A$; $A.\Gamma = \{i | i \in A\}$; $\Sigma = \emptyset$;
2. **for** each row r_{id} of D that $r_{id} \notin \mathcal{R}(A)$:
   if $(\mathcal{I}(r_{id}) \cap A) \subset A$ **then** add $(\mathcal{I}(r_{id}) \cap A)$ to Σ;
3. **for** each closed set $A' \in \Sigma$:
   $A.\Gamma 1 = A.\Gamma 2 = \emptyset$;
   for each lower bound $l_i \in A.\Gamma$:
   if $l_i \subseteq A'$ **then** add l_i to $A.\Gamma 1$;
   else add l_i to $A.\Gamma 2$;
   $CandiSet = \emptyset$;
   for each $l_i \in A.\Gamma 1$ and each $i \in A$ && $i \notin A'$:
   add candidate $l_i \cup \{i\}$ to $CandiSet$;
   $A.\Gamma = A.\Gamma 2$;
   for each candidate $c_i \in CandiSet$
   if c_i does not cover any $l_i \in A.\Gamma 2$ and c_i does not cover any other $c_j \in CandiSet$
   then add c_i to $A.\Gamma$
4. Output $A.\Gamma$.

Figure 9: MineLB

ALL-AML leukemia (ALL) [7], and colon tumor (CT) [8]. In such datasets, the rows represent clinical samples while the columns represent the activity levels of genes/proteins in the samples. There are two categories of samples in these datasets.

dataset	# row	# col	class 1	class 0	#row of class 1
BC	97	24481	relapse	nonrelapse	46
LC	181	12533	MPM	ADCA	31
CT	62	2000	negative	positive	40
PC	136	12600	tumor	normal	52
ALL	72	7129	ALL	AML	47

Table 1: Microarray Datasets

Table 1 shows the characteristics of these 5 datasets: the number of rows (# row), the number of columns (# col), the two class labels (class 1 and class 0), and the number of rows for class 1 (# class 1). All experiments presented here use the class 1 as the consequent; we have found that using the other consequent consistently yields qualitatively similar results.

To discretize the datasets, we use two methods. One is the entropy-minimized partition (for CBA and IRG classifier)[9] and the other is the equal-depth partition with 10 buckets. Ideally, we would like to use only the entropy discretized datasets for all experiments since we want to look at the classification performance of IRGs. Unfortunately, the two rule mining algorithms that we want to compare against are unable to run to completion within reasonable time (we ran them for several days without results) on the entropy discretized datasets, although FARMER is still efficient. As a result, we will report our efficiency results based on the equal-depth partitioned data while our classifier is built using the entropy-discretized datasets.

4.1 Efficiency of FARMER

The efficiency of FARMER will first be evaluated. We compare FARMER with the interesting rule mining algorithm in [2]. The algorithm in [2] is the one most related to FARMER in terms of interesting rule definition (but not the same, see related work). To our best knowledge, it is also the most efficient algorithm that exists with the purpose of mining interesting rules of our kind. We denote this algorithm as *ColumnE* since it also adopts column enumeration like most existing rule mining algorithms. We also compare FARMER with the closed set discovery algorithms CHARM [23] and CLOSET+ [21], which are shown to be more efficient than other association rule mining algorithms in many cases. We found that CHARM is always orders of magnitude faster than CLOSET+ on the microarray datasets and thus we do not report the CLOSET+ results here. Note that the runtime of FARMER includes the time for computing both the upper bound and lower bounds of each interesting rule group. Compared with CHARM, FARMER does extra work in: 1)computing the lower bounds of IRGs and 2) identifying the IRGs from all rule groups. Unlike FARMER that discovers both upper bound and lower bounds for each IRG, ColumnE only gets one rule for each IRG.

4.1.1 Varying Minimum Support

The first set of experiments (Figure 10) shows the effect of varying minimum support threshold *minsup*. The graphs plot runtime for the three algorithms at various settings of minimum support. Note that the y-axes in Figure 10 are in logarithmic scale. We set both $minconf$ and *minchi* as ZERO, which disables the pruning with confidence upper bound and the pruning with the chi-square upper bound of FARMER.

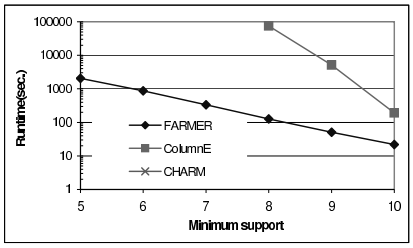
For CHARM, *minsup* represents the least number of rows that the closed sets must match. The runtime of CHARM is not shown in Figures 10(a) and 10(b) because CHARM runs out of memory even at the highest support in Figure 10 on datasets BC and LC.

Figure 10 shows that FARMER is usually 2 to 3 orders of magnitude faster than ColumnE and CHARM (if it can be run). Especially at low minimum support, FARMER outperforms ColumnE and CHARM greatly. This is because the candidate search space for ColumnE and CHARM, dependent on the possible number of column combinations after removing the infrequent items, is orders of magnitude greater than the search space of FARMER, dependent on the possible number of row combinations, on microarray datasets.

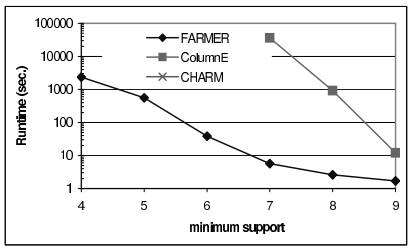
As shown in Figure 10(f), the number of interesting rule groups discovered at a low *minsup* is much larger than that at a high *minsup*. Besides the size of row enumeration space, the number of IRGs also affects the efficiency of FARMER due to two reasons. First, since FARMER discovers IRGs by comparison (see algorithm section, step 7), more time will be spend when the number of IRGs to be compared against increase. Second, the time complexity of computing lower bounds in FARMER is $O(n)$, where n is the number of IRGs. We observe that at high *minsup*, the time used to compute lower bounds takes 5% to 10% of the runtime of FARMER while the time taken at low *minsup* can be up to 20%. ColumnE also does the comparison to get interesting rules while all the runtime of CHARM is used to discover closed sets.

We choose our minimum support such that the runtime of FARMER is around 10 seconds. Although ColumnE and CHARM could perform better that FARMER for higher *minsup*, the absolute time difference however will be less

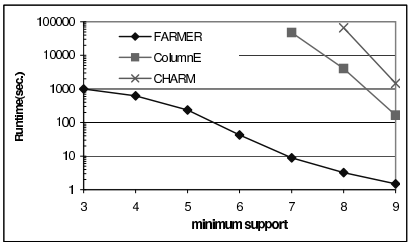
[7] http://www-genome.wi.mit.edu/cgi-bin/cancer
[8] http://microarray.princetion.edu/oncology/affydata
[9] the code is available at http://www.sgi.com/tech/mlc/



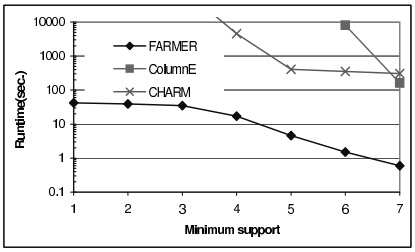
(a) Lung Cancer



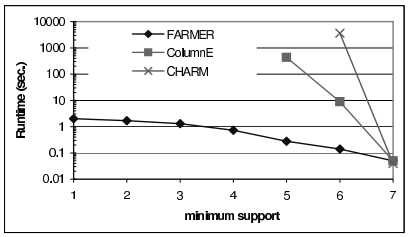
(b) Breast Cancer



(c) Prostate Cancer



(d) ALL-AML leukemia

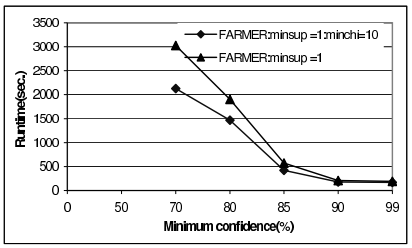


(e) Colon Tumor

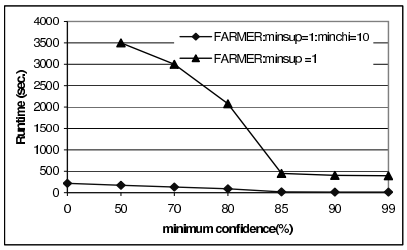
Dataset	minsup	# IRGs	minsup	# IRGs
LC	10	13880	5	1137398
BC	9	1355	4	1405703
PC	9	5234	3	1275095
ALL	7	6921	1	183521
CT	7	356	1	31294

(f) Number of IRGs vs minsup (minchi=0)

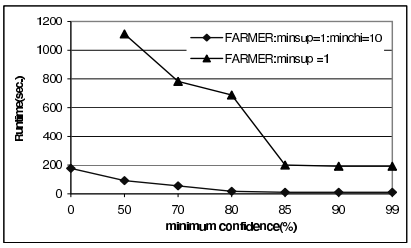
Figure 10: Varying *minsup*



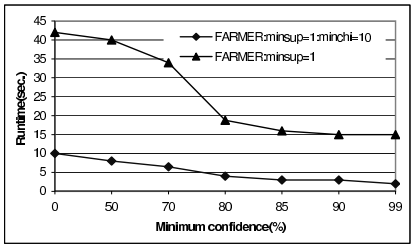
(a) Lung Cancer



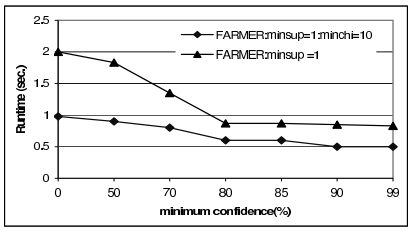
(b) Breast Cancer



(c) Prostate Cancer



(d) ALL-AML leukemia



(e) Colon Tumor

Dataset	minconf	# IRGs	minconf	#IRGs
LC	0.99	37316	0.7	1173020
BC	0.99	214640	0.5	2154705
PC	0.99	184310	0.5	1233525
ALL	0.99	38333	0	183521
CT	0.99	5101	0	31294

(f) Number of IRGs vs minconf (minchi=0)

Figure 11: Varying *minconf*

than 10 seconds and thus is not interesting for comparison. This is negligible compared to the difference in runtime at low *minsup*.

4.1.2 Varying Minimum Confidence

The next set of experiments (Figure 11) shows the effect of varying *minconf* when *minsup* is fixed. The *minchi* pruning is still disabled by setting it to ZERO. For all the parameter settings in Figure 11, CHARM is unable to finish because of insufficient memory after several hours while ColumnE always has a runtime of more than 1 day. This is because we adopt a relative low *minsup* to study the effectiveness of confidence pruning in the experiment. To show the effect of various *minconf* clearly, we do not give the runtime of ColumnE. We will first ignore the lines marked with "minchi=10" here. We set $minsup = 1$, which means that minimum support pruning is almost disabled.

Figure 11 shows that the runtime of FARMER decreases when increasing *minconf* on all the 5 datasets (Figure 11(f) lists the number of IRGs). This shows that it is effective to exploit the confidence constraint for pruning. There is only

a slight decrease in runtime of FARMER when the *minconf* increases from 85% to 99% since there are few upper bound rules with confidence between 85% and 99%. We observe that nearly all IRGs discovered at confidence 85% on these 5 datasets have a 100% confidence. As a result, FARMER does no additional pruning when *minconf* increases from 85% to 99%.

The result that many discovered IRGs have a 100% confidence is interesting and promising. It means that the IRGs are decisive and have good predictability.

4.1.3 Varying Minimum Chi-Square Value

The last set of experiments was performed to study the effectiveness of the chi-square pruning. Minimum chi-square constraint is usually treated as a supplementary constraint of minimum support and minimum confidence. We set $minchi = 10$ and plot the runtime vs various *minconf* in Figure 11 due to the space limitation, where *minconf* is set the same as in section 4.1.2.

The pruning exploited by constraint $minchi = 10$ is shown to be very effective on datasets BC, PC, CT and ALL. In some cases, the saving can be more than an order of magnitude. The pruning effect is not so obvious on dataset LC. By checking the identified IRGs, we found that discovered IRGs from LC usually have higher chi-square value. If we impose a tighter chi-square constraint by increasing *minchi*, the *minchi* pruning will be more obvious. Due to space constraint, we do not discuss this further.

As can be seen, in all the experiments we conducted, FARMER outperforms ColumnE and CHARM. Moreover, the prunings based on *minsup*, *minconf* and *minchi* are effective. In general, the runtime of FARMER correlates strongly with the number of interesting rule groups that satisfy all of the specified constraints. Our experimental results demonstrate that FARMER is extremely efficient in finding IRGs on datasets with small number of rows and large number of columns.

In additional to these experiments, we also look at how the performance of FARMER varies as the number of rows increase. This is done by replicating each dataset a number of times to generate a new dataset. It is observed that the performance of FARMER still outperform other algorithms even when the datasets are replicated for 5-10 times. Due to lack of space, we refer readers to [6] for these additional experiments.

4.2 Usefulness of IRGs

In order to show the usefulness of the discovered IRGs, we build a simple classifier called **IRG classifier** based on those IRGs that we discovered. Note that our **emphasis** here is not to build a new classifier but to provide some evidence that the discovery of IRGs is at least useful for such purpose.

We will compare our IRG classifier with two well-known classifiers CBA [14] and SVM [12], both available through the Internet. The open-source CBA algorithm (and all competitors we look at in the earlier section) failed to finish running in one week. To go around this problem, we build the CBA classifier by obtaining the frequent rules based on the upper bounds and lower bounds generated by FARMER. Our IRG classifier is similar to CBA but we use IRGs to build classifiers instead of all rules. Due to space limitation, we do not explain the details of the IRG classifier.

For CBA, we set the minimum support threshold as $0.7*$ number of training data of class C for each class C and set the minimum confidence threshold as 0.8 (According to our experiments, if we further lower the minimum confidence threshold, the final CBA classifier is the same); For IRG classifier, we use the same parameters as CBA; For SVM, we always use the default setting of SVM^{light}[12].

dataset	#training	#test	IRG classifier	CBA	SVM
BC	78	19	**78.95%**	57.89%	36.84%
LC	32	149	89.93%	81.88%	**96.64%**
CT	47	15	**93.33%**	73.33%	73.33%
PC	102	34	**88.24%**	82.35%	79.41%
ALL	38	34	64.71%	91.18%	**97.06%**
Average Accuracy			**83.03%**	77.33%	76.66%

Table 2: Classification Results

Table 2 illustrates the percentages of correctly predicted test data for the IRG classifier, CBA and SVM on the 5 microarray datasets. We can see that the IRG classifier has the highest average accuracy. Although SVM performs very well on LC and ALL, it fails on BC. No classifier outperforms the others on all datasets. Our IRG classifier is both efficient and easily understandable and thus could be a good reference tool for biological research.

5. RELATED WORK

Association rule mining has attracted considerable interest since a rule provides a concise and intuitive description of knowledge. It has already been applied on biological data, such as [7, 8, 19].

Association rule can relate gene expressions to their cellular environments or categories, thus available for building accurate classifiers on microarray datasets as in [9, 13]. Moreover, it can discover the relationship between different genes, so that we can infer the function of an individual gene based on its relationship with others [7] and build the gene network. Association rules might reveal more patterns than clustering [5], considering that a gene may belong to many rules while it is usually grouped to one cluster (or a hierarchy of clusters).

There are many proposals about rule mining in the data mining literatures. They can be roughly divided into three classes. The first two classes are related to association rule mining. All existing association rule mining algorithms adopt the column enumeration in the mining process, therefore they are very time-consuming on microarray datasets. The first class of rule mining algorithms identifies the interesting (or optimal) rules with some interestingness measures [2]. The interesting rule discussed in [2] is quite similar to our interesting rule group. However, [2] randomly picks a rule for each rule group while FARMER discovers the upper bound and lower bounds for each interesting rule group.

The second class of rule mining algorithms aims to find all association rules satisfying user-specified constraints by identifying all frequent itemsets at the key step, such as [1, 11]. Recently the concept of closed itemset [18] is proposed to reduce redundant itemsets and rules [22]. Several efficient mining algorithms [18, 23, 21] are proposed to mine frequent closed itemsets. On the other hand, there is some work [16, 20] that investigates incorporating item constraints to reduce the number of frequent itemsets. Other work [3, 15] leverages the item constraint as rule consequent and utilizes

minimum thresholds of confidence, support and other statistic constraints. FARMER differs from these approaches in term of its enumeration method and pruning strategies.

The third class of algorithms aims at mining predictive rules. One example is the decision tree induction algorithm[10]. Alternatively, some work [9, 14] has been done to build classifiers from association rules and has obtained better classification results than decision trees in many cases. It is obvious that these methods are also applicable based on the concept of interesting rule groups.

In a short paper [17], the idea of using row enumeration for mining closed patterns in biological datasets is introduced. The idea is however restricted to finding frequent closed patterns that satisfy a certain support threshold. FARMER on the contrary finds IRGs that satisfy interestingness constraints like *minconf* and *minchi*. The effectiveness of pruning with such constraints is evident in our experiments.

6. CONCLUSIONS

In this paper, we proposed an algorithm called FARMER for finding the interesting rule groups in microarray datasets. FARMER makes use of the special characteristic of microarray datasets to enhance its efficiency. It adopts the novel approach of performing row enumeration instead of the conventional column enumeration so as to overcome the extremely high dimensionality of microarray datasets. Experiments show that FARMER outperforms existing algorithms like CHARM and ColumnE by a large order of magnitude on microarray datasets.

Our IRG classifier built on interesting rule groups demonstrates the usefulness of discovered IRGs. Our experiments showed that it has the highest average accuracy compared with CBA and SVM.

Acknowledgment: We will like to thank the anonymous reviewers from various conferences for their helpful suggestions which have led to great enhancements on the paper. Anthony Tung will like to thank his wife, Monica, for the patience and support through the course of his work.

7. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94)*, pages 487–499, Sept. 1994.

[2] R. J. Bayardo and R. Agrawal. Mining the most intersting rules. In *Proc. of ACM SIGKDD*, 1999.

[3] R. J. Bayardo, R. Agrawal, and D. Gunopulos. Constraint-based rule mining on large, dense data sets. In *Proc. 1999 Int. Conf. Data Engineering (ICDE'99)*.

[4] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'99)*.

[5] Y. Cheng and G. M. Church. Biclustering of expression data. In *Proc of the 8th Intl. Conf. on intelligent Systems for Mocular Biology*, 2000.

[6] G. Cong, A. K. H. Tung, X. Xu, F. Pan, and J. Yang. Farmer: Finding interesting rule groups in microarray datasets. *Technical Report: National University of Singapore*, 2004.

[7] C. Creighton and S. Hanash. Mining gene expression databases for association rules. *Bioinformatics*, 19, 2003.

[8] S. Doddi, A. Marathe, S. Ravi, and D. Torney. Discovery of association rules in medical data. *Med. Inform. Internet. Med.*, 26:25–33, 2001.

[9] G. Dong, X. Zhang, L. Wong, and J. Li. Caep: Classification by aggregating emerging patterns. In *Proc. 2nd Int. Conf. Discovery Science (DS'99)*.

[10] J. Gehrke, R. Ramakrishnan, and V. Ganti. Rainforest: A framework for fast decision tree construction of large datasets. In *Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98)*.

[11] J. Han and J. Pei. Mining frequent patterns by pattern-growth:methodology and implications. *KDD Exploration*, 2, 2000.

[12] T. Joachims. Making large-scale svm learning practical. 1999. svmlight.joachims.org/.

[13] J. Li and L. Wong. Identifying good diagnostic genes or genes groups from gene expression data by using the concept of emerging patterns. *Bioinformatics*, 18:725–734, 2002.

[14] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining (KDD'98)*.

[15] S. Morishita and J. Sese. Traversing itemset lattices with statistical metric prunning. In *Proc. of PODS*, 2002.

[16] R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)*.

[17] F. Pan, G. Cong, A. K. H. Tung, J. Yang, and M. J. Zaki. Carpenter: Finding closed patterns in long biological datasets. In *Proc. 2003 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'03)*, 2003.

[18] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proc. 7th Int. Conf. Database Theory (ICDT'99)*, Jan.

[19] J. L. Pfaltz and C. Taylor. Closed set mining of biological data. In *Workshop on Data Mining in BIoinformatics with (SIGKDD02)*, 2002.

[20] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proc. 1997 Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, 1997.

[21] J. Wang, J. Han, and J. Pei. Closet+: Searching for the best strategies for mining frequent closed itemsets. In *Proc. 2003 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'03)*, 2003.

[22] M. Zaki. Generating non-redundant association rules. In *Proc. 2000 Int. Conf. Knowledge Discovery and Data Mining (KDD'00)*, 2000.

[23] M. Zaki and C. Hsiao. Charm: An efficient algorithm for closed association rule mining. In *Proc. of SIAM on Data Mining*, 2002.