FARMER: Finding Interesting Rule Groups in Biological Datasets

Gao Cong * Xin Xu Feng Pan Anthony K. H. Tung
National University of Singapore. email: {panfeng,congc,gao,xin, atung}@comp.nus.edu.sg
*Contact Author

Abstract

The growth of bioinformatics has resulted in datasets with new characteristics. These datasets typically contain a large number of columns and 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.

Association rules can reveal biological relevant associations between genes and environmental/categories to identify gene regulation pathways. However, most existing association rule mining algorithms have an exponential dependence on the number of columns. Moreover, the number of association rules generated from bioinformatic datasets are enormous due to the combinatorial explosion of frequent itemsets. In this paper, we describe a new algorithm called FARMER that is specially designed to discover interesting rule groups by identifying their upper bounds and lower bounds from biological datasets. FARMER exploits all user specified constraints including minimum support, minimum confidence and minimum chi-square to support efficient pruning. Several experiments on real bioinformatics datasets show that FARMER is orders of magnitude better than previous association rule mining algorithms.

1 Introduction

With recent advance in DNA chip-based technologies [22], we can now measure the expression level of thousands of genes in cell simultaneously, which results in a large amount of high-dimensional 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.

One special association rule takes the form of \( LHS \rightarrow C \), where \( LHS \) is a set of items and \( C \) is a class label. Let \( \text{sup}(LHS) \) be the number of rows containing \( LHS \) in the given database. Then the confidence of the rule will be \( \frac{\text{sup}(LHS \cup C)}{\text{sup}(LHS)} \). The user-specified constraints such as minimum support (a statement of generality) and minimum confidence (a statement of predictability) are often imposed on mining such association rules.

Recent studies have shown that such association rules are very useful in the analysis of microarray data. For example, it is shown in [9, 12] that classifiers built from association rules are rather accurate in identifying cancerous cell. Due to their relative simplicity, they are more easily interpreted by biologists, providing great help in the search for gene predictors of the sample categories.

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 had been developed for association rule mining [1, 14, 11, 19, 18, 2, 20, 27], 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 the datasets taken from the bioinformatics domain (or other domains with similar data characteristics), \( i \) can be in the range of tens of thousands. These high dimensional bioinformatics datasets with thousands of items 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.

A large number of long frequent itemsets may be discovered from microarray datasets due to their large number of columns. As a result, enormous number of association rules may be generated due to the combinatorial explosion of frequent itemsets [4]. For example, given a one row dataset with 5 columns and a label: \( \{a, b, c, d, e, \text{Cancer}\} \), we could have 31 rules of the form \( LHS \rightarrow \text{Cancer} \) since any combination of \( a, b, c, d, e \) could be the \( LHS \) for the rule. These 31 rules are all covered by the same row and have the same confidence (100%). Such a large set of rules contain a lot of redundancy and are difficult to interpret. Instead of generating all 31 rules, we propose to discover these rules as a rule group whose consequent is Cancer, which can be identified by a unique upper bound and a set of lower bounds. The upper bound of a rule group is the rule with the most specific \( LHS \) among the rules. In this case, the upper bound rule is \( abed \rightarrow \text{Cancer} \). 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 (\( a \rightarrow \text{Cancer}, b \rightarrow \text{Cancer}, e \rightarrow \text{Cancer}, d \rightarrow \text{Cancer}, and e \rightarrow \text{Cancer} \)). 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 \( abed \rightarrow \text{Cancer} \) with confidence 90% and \( ab \rightarrow \text{Cancer} \) with confidence 95%, it is obvious that \( ab \) is a better indicator of Cancer since \( ab \rightarrow \text{Cancer} \) has a higher confidence and all rows covering \( abed \rightarrow \text{Cancer} \) must cover \( ab \rightarrow \text{Cancer} \). With \( ab \rightarrow \text{Cancer} \), rule
In this paper, we describe a novel algorithm called FARMER, that is specially designed to mine interesting rule groups whose consequent is a specified class label. Our target datasets are microarray data which have large numbers of columns and relatively small numbers of rows. FARMER discovers upper bounds of interesting rule groups from datasets by performing depth-first row-wise enumeration instead of the usual column-wise approach taken by existing rule mining algorithms. Our attempt to do so is a bold one as it would be unthinkable to perform row enumeration instead of column enumeration in most circumstances. The 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. As experiments will show later, this unconventional approach, together with the pruning methods, produces good results when mining microarray datasets. Our algorithm substantially outperforms other rule mining algorithms described in [3], [27](CHARM) and [25](CLOSET+).

The Dataset

The dataset (or table) D consists of a set of rows, \( R = \{r_1, \ldots, r_n\} \), and \( C = \{C_1, C_2, \ldots, C_k\} \) be the complete set of class labels of D, then each row \( r_j \in R \) consists of one or more items from \( I \) and a class label from \( C \).

As an example, Figure 1(a) shows a dataset whose items are represented with alphabets from ‘a’ to ‘t’. There are altogether 5 rows, \( r_1 \ldots r_5 \), in the table, the first three of which are labelled ‘C’ while the other two are labelled ‘A’. 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, \( \{234\} \) denote the row set \( \{r_2, r_3, r_4\} \), and \( \{a, c, f\} \) denote the itemset \( \{a, c, f\} \).

**Item/Row Support Set**

Given a set of rows \( R' \) and a set of items \( I' \), we will define item support set and row support set as follows.

**Definition 2.1** Item Support Set, \( \mathcal{R}(I') \)

Given a set of items \( I' \subseteq I \), we use \( \mathcal{R}(I') \subseteq R \) denote the largest set of rows that contain \( I' \).

**Definition 2.2** Row Support Set, \( \mathcal{I}(R') \)

Given a set of rows \( R' \subseteq R \), we use \( \mathcal{I}(R') \subseteq I \) to denote 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 \( I' = \{a, c, h\} \), then \( \mathcal{R}(I') = \{r_2, r_3, r_4\} \). Let \( R' = \) the row set \( \{r_2, r_3\} \), then \( \mathcal{I}(R') = \{a, c, h\} \) since this is the largest itemset that occurs in both \( r_2 \) and \( r_3 \).

**Association Rule**

An association rule \( \gamma \), 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 \( \gamma \) is defined as the \( |\mathcal{R}(A \cup C)| \), and its confidence is \( \frac{|\mathcal{R}(A \cup C)|}{|\mathcal{R}(A)|} \). We denote the antecedent of \( \gamma \) as \( \gamma_A \), the consequent as \( \gamma_C \), the support as \( \gamma_{\text{sup}} \), the confidence as \( \gamma_{\text{conf}} \) and the chi-square value is \( \gamma_{\text{chi}} \).

As discussed in 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 help to reduce the number of rules discovered by identifying rules that come from the same set of rows and clustering them conceptually into interesting groups.

**Definition 2.3** Rule Group

Let \( D \) be the dataset with itemset \( I \) and \( C \) be the specified class label. \( G = \{A_1 \rightarrow C|A_1 \subseteq I\} \) is a rule group with antecedent support set \( R \) and consequent \( C \) iff \( \forall A_1 \rightarrow C \in G \), \( \mathcal{R}(A_1 \cup C) \subseteq \mathcal{R}(A_1) \). (2) if \( \forall A_1 \rightarrow C \in G \), \( \mathcal{R}(A_1) = R, A_1 \rightarrow C \in G \).

**Definition 2.4** the Upper Bound of a Rule Group

Let \( G = \{A_1 \rightarrow C\} \) be a rule group of dataset \( D \). Rule \( \gamma_{\text{ub}} \in G \) (\( \gamma_{\text{ub}}: A_1 \rightarrow C \)) is an upper bound of \( G \) if there exists no \( \gamma' \in G \) such that \( \forall A' \rightarrow C \) such that \( A' \supseteq A_1 \).

**Definition 2.5** the Lower Bound of a Rule Group

Let \( G = \{A_1 \rightarrow C\} \) be a rule group of dataset \( D \). Rule \( \gamma_{\text{lb}} \in G \) (\( \gamma_{\text{lb}}: A_1 \rightarrow C \)) is a lower bound of \( G \) if there exists no \( \gamma' \in G \) such that \( A' \supseteq A_1 \).

**Lemma 2.1** Given a rule group \( G \) with the consequent \( C \) and the antecedent support set \( R \), it has a unique upper bound \( \gamma_{\text{ub}} \) (\( A \rightarrow C \)).

**Proof:** Assume there exists another upper bound \( \gamma' : A' \rightarrow C \) in \( G \) such that \( A' \neq A \) and \( A' \subseteq A \). Let \( A'' = A \cup A' \), then \( A'' \rightarrow C \in G \) and \( A'' \supseteq A \), thus \( \forall \gamma' : A' \rightarrow C \) can not be a 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 \( \gamma_{\text{ub}} \).
Definition 2.6

C, ..., aeh

Proof:

They want to.

For users to recognize all the rule group members as and when liberately leave these constraint parameters out of our definition and satisfy user specified constraints including $\gamma$. It is obvious that all rules in the same rule group have the same support, confidence and chi square value since they are same support, confidence and chi square value since they are.

Lemma 2.2 Suppose rule group $G$ with the consequent $C$ and antecedent support set $R$ has an upper bound $A_u \rightarrow C$ and a lower bound $A_l \rightarrow C$. Rule $\gamma(A \rightarrow C)$, where $A \subset A_u$ and $A \supseteq A_l$ must be a member of $G$.

Proof: Since $A \subset A_u$, $R(A) \supseteq R(A_u)$. Likewise, $R(A) \subseteq R(A_l)$. Since $R(A_l) = R(A_u) = R$, $R(A) = R$. So $\gamma(A \rightarrow C)$ belongs to $G$.

Lemma 2.3

Interesting Rule Group (IRG)

A rule group $G$ with upper bound $\gamma_u$, $\gamma_u$ con $f < \gamma_u$ con $f$. For brevity, we will use the abbreviation IRG to refer to interesting rule group.

Our algorithm FARMER is designed for finding IRGs that satisfy user specified constraints including minimum support, minimum confidence and minimum chi square value. We deliberately leave these constraint parameters out of our definition of IRG to avoid restricting the definition of interestingness to these measures. FARMER finds the upper bounds of all IRGs first, 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

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 number $r_m$ 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 example 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 algorithm FARMER 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 item sets, such as [18, 20], 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 when we do not apply any punning technique (this tree is not physically built as a data structure). Each node $X$ of the enumeration tree corresponds to a combination of rows $R'$ and is labelled with $I(R')$ that is the antecedent of the upper bound of a rule group identified at this node. For example, node "$12n$" corresponds to the row combination $\{r_1, r_2\}$ and "$aln$" indicates that $I(r_1, r_2) = \{a, l\}$. An upper bound $al \rightarrow C$ can be discovered at node "$12n$". This is correct because of the following lemma.

Lemma 3.1 Let $X$ be a subset of rows from the original table, then $I(X) \rightarrow C$ must be the upper bound of the rule group $G$ whose antecedent support set is $R(I(X))$ and consequent is $C$.

Proof: First, according to definition 2.3, $I(X) \rightarrow C$ belongs to rule group $G$ with antecedent support set $R(I(X))$ and consequent $C$. Second, assume $I(X) \rightarrow C$ is not the upper bound of $G$, then there must exist an item $i$ such that $\not\in I(X)$, $I(X) \cup \{i\} \rightarrow C$ belongs to $G$. So $R(I(X)) = R(I(X) + \{i\})$. Since rows in $X$ contains all items of $I(X)$, $X \subseteq R(I(X))$, or $X \subseteq R(I(X) \cup \{i\})$. This means that $i$ is also found in every row of $X$ which contradicts the definition that $I(X)$ is the largest set of items that are found in every row of $X$. So $I(X) \rightarrow C$ is the upper bound of the rule group with antecedent support set $R(I(X))$.

FARMER performs a depth first search on the enumeration tree by moving along the edges of the tree. By imposing an order $ORD$, in which the rows with consequent $C$ are ordered BEFORE the rows without consequent $C$, on the set of row tuples in the transposed table actually represent the items in the original table.

Other constraint such as lift, conviction, entropy gain, gini and correlation coefficient can be handled similarly.
numbers, we are able to perform a systematic search by enumerating the combinations of rows based on the order ORDER. For example, let "1 2 3 4 5" according to ORDER, the order of search in Figure 3 will be {"1"", "2"", "12"", "123"", "1234"", "12345", "1235", "13", "135", "135", "1345", "12345"} in absence of any optimization and pruning strategies. Note that the order also serves for confidence pruning purpose (explained in section 3.2.3).

Next, we prove that the complete rule groups can be discovered by a complete enumeration of row combinations. Following is the lemma.

**Lemma 3.2** By enumerating all possible row combinations as shown in the row enumeration tree of Figure 3, we can obtain the complete set of upper bounds and corresponding the 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 2.3), 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 \(\sum(X) \rightarrow C\).

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 \((T{T}|_X)\)

Given the transposed table \(T{T}\) (used at the root of the enumeration tree), a \(X\)-conditional transposed table \((T{T}|_X)\) at node \(X\) (\(X\) is the row combination at this node) is a subset of tuples from \(T{T}\) such that for each tuple \(t\) of \(T{T}\) that \(t \subseteq X\), there exists a tuple \(t' = t \in T{T}|_X\).

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

**Definition 3.2** Enumeration Candidate List \((TT|_X.E)\)

Let \((T{T}|_X)\) be the \(X\)-conditional transposed table and \(r_{\text{min}} \in X\) be the row id with the lowest ORDER order in row combination \(X\). \(E_P = \{r | r \geq ORDER r_{\text{min}} \& \& r \in R(C)\}\) (all rows of consequent \(C\) ordered after \(r_{\text{min}}\), and \(E_N = \{r | r \geq ORDER r_{\text{min}} \& \& r \in R(\neg C)\}\) (all rows of no consequent \(C\) ordered after \(r_{\text{min}}\)). The enumeration candidate list for \((T{T}|_X)\), denoted as \((T{T}|_X.E)\), is defined to be \(E_P \cup E_N\).

In the remaining paper, we use the following notations to represent the attributes of \((T{T}|_X)\):

- \((T{T}|_X.E)\): enumeration candidates;
- \((T{T}|_X.E_P)\): enumeration candidates with consequent \(C\);
- \((T{T}|_X.E_N)\): enumeration candidates without consequent \(C\);
- \((T{T}|_X.Y)\): enumeration candidates that occurs in each tuple of \((T{T}|_X)\).

Our formal algorithm is shown in Figure 4. FARMER involves recursive computation of conditional transposed tables for performing a depth-first traversal of the row enumeration tree. (Section 3.3 will show that we use memory pointer to generate conditional transposed table instead of constructing conditional transposed table physically). 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 \(\text{MineIRGs}\) to recursively generate \(X\)-conditional tables.

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

Steps 1, 2, 4 and 5 in the subroutine \(\text{MineIRGs}\) perform the pruning. They are extremely important to the efficiency of FARMER Algorithm and will be explained in next subsection. Step 3 scans the table \((T{T}|_X)\). \(\text{MineIRGs}\) outputs a rule if and only if it is an upper bound of an interesting rule group satisfying user specified constraints (\text{minsup}, \text{minconf} and \text{minichi}). This is done at Step 6 which move on into the next level of enumeration in the search tree. Step 7 checks whether \(T(X) \rightarrow C\) is the upper bound of an IRG that satisfies the user specified constraints before inserting it into \(IRG\). Note that step 7 must be after step 6 and the reason will be clear later. We first prove the correctness of the two steps by two lemmas as follows:

**Lemma 3.3** \((T{T}|_X.r_i) = (T{T}|_{X + r_i})\) \(r_i \in (T{T}|_X.E)\).

**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 \((T{T}|_X)\) 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 \(\gamma\) discovered at node \(X\) represents an interesting rule group satisfying user specified constraints, we need to compare \(\gamma . \text{conf} \) with all \(\gamma' . \text{conf}\), where \(\gamma' . A \subset \gamma . A\) and \(\gamma'\) satisfies user specified constraints. FARMER ensures that all such \(\gamma'\) have already been discovered and kept in \(IRG\) at Step 7 by Lemma 3.4 below.
Algorithm FARMER
Input: table $D$, specified consequent $C$, $mins_{up}$, $minconf$, and $minchi$
Output: Interesting rule groups with consequent $C$ satisfying minimum thresholds
Method:
1. Initialization: Let $TT'$ be the transposed table of ORD ordered $D$; $IRG = \emptyset$.
2. Mine Interesting Rule Groups: $\text{MineIRGs}(TT'_A, 0, 0, IRG)$
   Subroutine: $\text{MineIRGs}(TT'|X, sup_p, sup_n, IRG)$
   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 $I(X) \rightarrow C$ is already identified, then return;
2. Apply Pruning 3: If prunable with the loose support or loose confidence upper bound, then return;
3. Scan $TT'|X$ and count the frequency of occurrences for each enumeration candidate, $r_i \in TT'|X_E$. Let $U_p \subseteq TT'|X_E$ be the set of rows from $TT'|X_E$ which occur in at least one tuple of $TT'|X$;
   Let $U_n \subseteq TT'|X_E$ be the set of rows from $TT'|X_E$ which occur in at least one tuple $TT'|X$;
   Let $Y_p \subseteq TT'|X_E$ be the set of rows from $TT'|X_E$ found in every tuple of $TT'|X$;
   Let $Y_n \subseteq TT'|X_E$ be the set of rows from $TT'|X_E$ found in every tuple of $TT'|X$.
   $sup_p = sup_p + |Y_p| (|R(I(X) \cup C)|);
   sup_n = sup_n + |Y_n| (|R(I(X) \cup \neg C)|)$
4. Apply Pruning 3: If prunable with the tight support upper bound or tight confidence upper bound or the chi square upper bound, then return;
5. Apply Pruning 1: Update enumeration candidate list, $TT'|X_E = U_p - Y_p, TT'|X_E = U_n - Y_n$.
6. for each $r_i \in TT'|X_E$ do
   if $r_i \in R(C)$ then $TT'|X|_{r_i,E_R} = \{r_j \mid r_j \in TT'|X|_{E_P \& k \& r_j > ORD r_i}\};
   TT'|x|_{r_i,E_N} = TT'|X|_{E_N};
   a = sup_p + 1; b = sup_n;
   else $\text{MineIRGs}(TT'|X|_{r_i}, a, b, IRG)$;
7. Let $conf = \frac{sup_p}{(sup_p + sup_n)}$;
   if $conf \geq minsup$ and $(conf \geq minconf) \land (chi(sup_p, sup_n) \geq minchi)$ then
   if $\forall r \gamma \in I(RG)$ then add upper bound rule $I(X) \rightarrow C$ into $IRG$;

Figure 4. The FARMER Algorithm

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

Step 7 is implemented after Step 6 to ensure all descendant nodes down $X$ are explored before determining whether the upper bound $\gamma$ at $X$ is $IRG$. 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 represents a form of pruning too 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 $I(X) = \emptyset$ (an example is node “134$'$”). This implies that there is no item which exists in all the rows in $X$. When this occurs, $TT'|X_E$ will be empty and no further enumeration will be performed.

3.2 Pruning Strategy

We next look at the pruning techniques that are used in FARMER to enhance its efficiency. These pruning strategies are essential to the efficiency of algorithm FARMER. Our emphasis here is to show that our pruning steps do not prune off any interesting rule groups while preventing unnecessary traversal of the enumeration tree. Combining this with our earlier explanation 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 $\text{MineIRGs}$ by pruning $TT'|X_E$. The set of enumeration candidate rows that occur in all tuples of the $TT'|X$. We partition $TT'|X \rightarrow Y$ to two subsets: $Y_i$ with consequent $C$ and $Y_n$ without. The intuitive reason for the pruning is that we obtain the same set of rules along the branch $X$ without such rows. The correctness of such a pruning strategy is because of the following lemma.

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

With Lemma 3.5, we can safely delete the rows in $TT'|X_E$ 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 $I((2,3)) = I((2,3,4)) = \{a, c, b\}$. Thus, we need not traverse node “23” and create $TT'|(2,3,4)$. Row 4 can be safely deleted from $TT'|(2,3)$.E.

Since $I((2,3,4)) = I((2,3))$, the upper bound rule is identified at node “23” and node “234$'$” is redundant. We say that node “234$'$” is compressed to node “23$'$”.

5
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 \( A \subseteq \mathcal{I}(X) \), is identified at a node \( X' \), which is traversed before node \( X \) or is a descendant node of node \( X \). With pruning strategy 1, \( X' \) might be compressed to a node \( X'' \) \((X'' \subset X' \& \mathcal{I}(X'') = \mathcal{I}(X') = A')\), and we can see \( X'' \) 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 upper bounds to be discovered at the descendants of node \( X \) will have been discovered too.

**Lemma 3.6** Suppose pruning strategy 1 is utilized in the enumeration tree. Let \( TT'|_X \) be the conditional transposed table of the current node \( X \). If there exists a row \( r' \notin X \) \& \( TT'|_X \cdot E \), for any ancestor node \( X_1 \), node \( X_2 \), \( r' \notin TT'|_X \cdot X \), that occurs in each tuple of \( TT'|_X \), then all upper bounds to be discovered in the subtree rooted at node \( X \) must have been discovered already.

**Proof:** Let \( X = \{r_1, r_2, ..., r_n\} \), where \( r_1 \neq Ord r_2 \neq Ord \ldots \neq Ord r_n \). For node \( X'' = X \cup \{r'\} \), we have: a. \( \mathcal{I}(X) = I(X''_i) \); b. \( r' \neq Ord r_m \), so \( r' \notin TT'|_X \cdot E \), that occurs in each tuple of \( TT'|_X \), \( \cdot E \). If node \( X'' \) or node \( X_C \) is traversed before node \( X \), the conclusion is done. It is true. (1) If \( r'' \neq Ord r_1 \), node \( X'' \) or node \( X_C \) falls in the subtree rooted at node \( \{r''\} \), which is traversed before node \( X \). (2) If \( r_1 \neq Ord r_2 \neq Ord \ldots \neq Ord r_1 \neq Ord r_2 \neq Ord \ldots \neq Ord r_m \neq Ord \). The ordered rows ids from \( X'' \), node \( X'' \) and node \( X_C \), with upper bounds to be discovered at node \( X'' \), also before node \( X \).\[ \square \]

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

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

### 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 bound of the measures for the subtree rooted at current node \( X \). If the estimated upper bound at \( X \) is less than the user specified threshold, we stops searching down node \( X \).

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 4). At Step 2, we will perform pruning using a loose support and confidence upper bounds that can be calculated before scanning \( TT'|_X \). At Step 4 we calculate a tight support and confidence upper bounds and the chi square upper bound after scanning \( TT'|_X \).

For clarity, we list the notations to be used in the lemmas in this subsection.

**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 \( U_{s2} \) (after scanning \( TT'|_X \)). If the estimated upper bound is less than minimum support \( minsup \), the subtree can be pruned.

If \( r_m \) has consequent \( C \):

\[ U_{s1} = \gamma'.sup + 1 + \max(|TT'|_X \cdot E_{p} \cap \{t\}, t \in TT'|_X); \]

\[ U_{s2} = \gamma'.sup + 1 + |TT'|_X \cdot E_{p}; \]

If \( r_m \) has no consequent \( C \):

\[ U_{s1} = U_{s2} = \gamma'.sup; \]

**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:** If \( r_m \) has no consequent \( C \), the enumeration candidates of nodes down node \( X \) have no consequent \( C \) too because of the ORD order (definition 3.2). The support can not increase down node \( X \), so the support of upper bounds discovered at the subtree rooted at node \( X \) is less than \( \gamma'.sup \). If \( r_m \) has consequent \( C \), for node \( X \) and its descendant nodes, the maximum support of upper bounds discovered at node \( X \) plus 1 (for \( r_m \)) is less than the maximum number of enumeration candidates with consequent \( C \) \((|TT'|_X \cdot E_{p})\) at node \( X \) plus 1 (for \( r_m \)), or more strictly, from the maximum number of enumeration candidates with consequent \( C \) within a tuple of \( TT'|_X \cdot E_{p} \cap \{t\}, t \in TT'|_X \) plus 1 (for \( r_m \)).\[ \square \]

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 its parent node without scanning \( TT'|_X \). If \( r_m \) has consequent \( C \), \( \gamma'.sup + 1 = sup_{p} \), else \( \gamma'.sup = sup_{p} \), where \( sup_{p} \) is the input parameter of current MineIRGs subroutine (Figure 4).

**Pruning Using Confidence Upper Bound**

Similarly, we estimate two 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 can be pruned.

Given \( U_{c1} \) and \( U_{c2} \), the two support upper bounds of subtree rooted at node \( X \), then

\[ U_{c1} = U_{c1}/(U_{c1} + |\mathcal{R}(\gamma'.A \cup \neg C)|); \]

\[ U_{c2} = U_{c2}/(U_{c2} + |\mathcal{R}(\gamma'.A \cup \neg C)|) \] (\( r_m \) has consequent \( C \));
Lemma 3.8 \( U_{c2} = U_{s2}/(U_{s2} + |R(\gamma', A \cup -C)|) + 1 \) \((r_m\) has no consequent \(C\)).

Proof: For a rule \( \gamma'' \) discovered in subtree rooted at node \(X\), its confidence is computed as \( |R(\gamma'', A \cup C)|/|R(\gamma'', A \cup -C)| \) (figure 4). This expression can be simplified as \( x/(x+y) \), where \( x = |R(\gamma''. A \cup C)| \) and \( y = |R(\gamma''. A \cup -C)| \). This value is maximized with the largest \(x \) (\(U_{s1}\) and \(U_{s2}\)) and smallest \(y\). Suppose rule \( \gamma \) is discovered at node \(X\). For any upper bound rule \( \gamma'' \) discovered down node \(X, \gamma'' \subseteq \gamma.A\) because of pruning strategy 1, so we can see \( |R(\gamma'', A \cup -C)| \geq |R(\gamma.A \cup -C)| \). Thus \( y \) is minimized with the smallest \(|R(\gamma'. A \cup -C)| \) or loosely at \(|R(\gamma'. A \cup -C)| + 1 \) if \(r_m\) has no consequent \(C\) and \(|R(\gamma'. A \cup -C)| \) if \(r_m\) has consequent \(C\).

Note that if \(r_m\) has consequent \(C\) then \(|R(\gamma'. A \cup -C)| = sup_m, else\(|R(\gamma'. A \cup -C)| + 1 = sup_m, where sup_m is the input parameter of current subroutine MineIRGs (figure 4).

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 expectation. Let \( x' \) be a rule in the form of \( A \rightarrow C \) of dataset \(D\), \( n\) be the number of rows of \(D\), and \( m\) be the number of instance with consequent \( C\) of \(D\). The four observed values for chi square value computation are listed in the following table. For example \( O_{A\leftarrow C}\) represents the number of rows that contain \(A\) but no \(C\). \( x = O_A\) 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 \(chi(x, y)\).

<table>
<thead>
<tr>
<th></th>
<th>( O_{\leftarrow C} )</th>
<th>( O_{\rightarrow C} )</th>
<th>( O_A )</th>
<th>( O_{A\leftarrow C} )</th>
<th>( O_{\rightarrow C} )</th>
<th>( O_{\rightarrow A} )</th>
<th>( O_{AC} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td>( O_{AC} = y )</td>
<td>( O_{A\leftarrow C} )</td>
<td>( O_A = x )</td>
<td>( O_{A\leftarrow C} )</td>
<td>( O_{\rightarrow C} )</td>
<td>( O_{\rightarrow A} = n - m )</td>
<td>( O_{AC} )</td>
</tr>
<tr>
<td>( \neg A )</td>
<td>( O_{AC} = y )</td>
<td>( O_{A\leftarrow C} )</td>
<td>( O_A = x )</td>
<td>( O_{A\leftarrow C} )</td>
<td>( O_{\rightarrow C} )</td>
<td>( O_{\rightarrow A} = n - m )</td>
<td>( O_{AC} )</td>
</tr>
<tr>
<td>Total</td>
<td>( O_{AC} = m )</td>
<td>( O_{A\leftarrow C} = n - m )</td>
<td>( O_A )</td>
<td>( O_{A\leftarrow C} )</td>
<td>( O_{\rightarrow C} )</td>
<td>( O_{\rightarrow A} = n - m )</td>
<td>( O_{AC} )</td>
</tr>
</tbody>
</table>

The following lemma gives an estimation of upper bound of chi-squared value for rules down the node \(X\).

Lemma 3.9 Suppose rule \( \gamma \) 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: 
\[
\text{max}\{\chi(x(\gamma) - y(\gamma) + m, m, m), \chi(y(\gamma) + n - m, y(\gamma))\},
\]
\[
\chi(x(\gamma), y(\gamma)).
\]

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

1. \( x' \leq x \leq n (\text{if } |R(A)| \leq |R(A')|)\)
2. \( y' \leq y \leq m (\text{if } |R(A' \cup C)| \leq |R(A' \cup C)|)\)
3. \( y' \leq x' (\text{if } |R(A' \cup C)| \leq |R(A')|)\)
4. \( n - m \geq x' - y' \geq x - y (\text{if } |R(A' \cup -C)| \geq |R(A' \cup -C)|)\)

Figure 5. The Possible Chi-square Variables

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 5). Since the chi square function \(\chi(x, y)\) is a convex function [15], which is maximized at one of its vertices, and \(\chi(n, m) = 0\) (please refer to [15]), we only need to consider the remaining three vertices.

3.3 Implementation

In the implementation of FARMER, we use memory pointers to point at the relevant tuples in the in-memory transposed table to simulate the conditional transposed table. Other algorithms which adopt such an approach include BUC [5] and H-mine [19]. Interested readers can refer to [5, 19] for details.

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 only small number of rows (usually from 100 to 1000).

Following is the running example. Suppose the current node is node "1p" (Figure 6), and \(\text{minsup} = 1\). The in-memory transposed table is shown on the right hand side of the figure. Memory pointers are organized into conditional pointer lists.

Since the "1p"-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 the tuple ‘a’ is contained in row \(r_1\) at position 1. We can extend the "1p"-conditional transposed table \(TT'_{\langle 1 \rangle}\) 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 "2p"-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 "2p"-conditional pointer list. It will be inserted subsequently as we will see later.

A further scan through the "1p"-conditional pointer list will allow us to generate the "12", "13", "14" and "15" conditional pointer lists. Figure 7 shows the state of memory pointers when we are processing node \{1, 2\}. Again, note that feature \(a\) is not in the "13"-conditional pointer list although the corresponding tuple does contain the set of rows \{r_1, r_3\}. This is because we will first process the combination \{1, 2\} and move entries in "12"-conditional pointer list to the other pointer lists (including
combinations "13", "14" and "15" here) later. As a result, the existing "13"-conditional pointer list only indicates tuples that have \{1, 3\} BUT NOT \{1, 2\}.

Finally, we show the state of conditional pointer lists after node \{1\} and all its descendants have been processed (Figure 8). Since all enumerations involving row \( r_2 \) have been either processed or pruned off, the entries in the "12"-conditional pointer list are moved into the remaining conditional pointer lists. For the "23"-conditional pointer list, two new entries are appended: \(< a, 2 >\) and \(< l, 2 >\). This indicates that both tuples representing ‘a’ and ‘l’ contain \( r_2 \) at the second position. The entries in the "23"-conditional pointer list will be moved to the other conditional pointer lists after node \{2\} and it 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, we scan from the position of each pointer to the first row of each tuple, instead of scanning the transposed table from the position of each pointer to the end of each tuple. In the case, 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 bound

After discovering the upper bounds of interesting rule groups, users might be interested to have a closer look at the members of the rule group. To do so, we derive the algorithm, MineLB which will find the lower bounds of a rule group. Since a rule group has an unique upper bound and the consequent of a rule group is fixed, the remaining problem is to generate 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 as we can generate the lower bounds of a closed set (definition 3.4).

**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 \( \mathcal{R}(A) = \mathcal{R}(A') \).

**Definition 3.4 Lower Bound of a Closed Set**

Suppose \( A \) is a closed set of dataset \( D \). \( A_1, A_1 \subseteq A \), is a lower bound of closed set \( A \), iff \( \mathcal{R}(A_1) = \mathcal{R}(A) \) and there is no \( A' \subset A_1 \) such that \( \mathcal{R}(A') = \mathcal{R}(A) \).

MineLB is an incremental algorithm that is initialized with one closed set \( A \), which is the antecedent of an upper bound \( A \rightarrow C \) of 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 upper bound \( A' \rightarrow C \). In this way MineLB keeps track of the latest lower bounds of \( A \). MineLB is based on the following lemmas.

**Lemma 3.10** Let \( A \) be the closed set which 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 \( A \Gamma_1 = \{ l_i | l_i \in A \Gamma \land l_i \subset A' \} \). \( A \Gamma_2 = \{ l_i | l_i \in A \Gamma \land l_i \notin A \Gamma_1 \} \). Then the newly generated lower bounds of \( A \) must be in the form of \( l_1 \cup \{ i \} \), where \( l_1 \in A \Gamma_1 \) and \( i \in A \land i \notin \Gamma \).

**proof:**

Suppose \( l \) is a newly generated lower bound of \( A \).

(1) \( l \supset l_1 \). Since \( \mathcal{R}(l_1) = \mathcal{R}(A) \) (Definition 2.2) before \( A' \) is added, there must exist a \( l_1 \in A \Gamma \) such that \( l_1 \subset l \subset A \). If \( l_1 \in A \Gamma_2 \), \( l \) can not be a new lower bound, since \( l_1 \in A \Gamma_2 \) is still a lower bound of \( A \) after \( A' \) is added. So \( l \supset l_1 \). \( l_1 \in A \Gamma_1 \).

(2) 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 \( l_1 \in A \Gamma_1 \) and \( i \in A \land i \notin A' \). 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 of \( i \in A' \). So, \( l' = l_1 \cup \{ i \} \) is a 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 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
Subroutine: MineLB (Table 9. upper bound rule; γ).

1. \( A = γ. A; A. Γ = \{i | i \in A\}; \ Σ = \emptyset; \)
2. for each row \( r_{ad} \) in \( D \) that \( r_{ad} \notin R(A) \) do
   if \( (I(r_{ad}) \cap A) \subset A \) then add \( (I(r_{ad}) \cap A) \) to \( Σ \);
3. for each closed set \( A' \) in \( Σ \) do
   \( A. Γ = A. Γ_2 = \emptyset; \)
   for each lower bound \( l_i \in A. Γ \) do
     if \( l_i \subset A' \) then add \( l_i \) to \( A. Γ_1 \); else add \( l_i \) to \( A. Γ_2 \); \( \text{CandiSet} = \emptyset; \)
   for each candidate \( c_i \in \text{CandiSet} \) do
     if \( c_i \) does not cover any \( l_i \in A. Γ \)
     and \( c_i \) does not cover any other \( c_j \in \text{CandiSet} \)
     then add \( c_i \) to \( A. Γ \). 
   \}
4. Output \( A. Γ \).

Figure 9. MineLB

the correctness of MineLB because of the Lemma 3.11

Lemma 3.11 If closed set \( A_1 \subset A \) is already added and the collection of \( A \)'s lower bounds \( A. Γ \) is already updated, \( A. Γ \) will not change after adding closed set \( A_2, A_2 \subset A_1 \).

proof: After \( A_1 \subset A \) is added, \( A. Γ \) is updated so that no \( l_i \in A. Γ \) can satisfy \( l_i \subset A_1 \). So no \( l_i \in A. Γ \) can satisfy \( l_i \subset A_2, A_2 \subset A_1 \). Since \( A_2 \) will not cover any \( l_i \in A. Γ \), \( A. Γ \) will not change, according to Lemma 3.10.

Example 7 Find Lower Bound

Given an upper bound rule with antecedent \( A = abcd \) and two rows, \( r_1 : abef \) and \( r_2 : cdeg \), the lower bounds \( A. Γ \) of \( A \) can be determined as follows.

1. Let the set of lower bounds \( A. Γ = \{a, b, c, d, e\} \).
2. add "abcdef" (= \( I(r_1) \cap A \)): We get \( A. Γ_1 = \{a, b, c\} \) and \( A. Γ_2 = \{d, e\} \). Since all the candidate lower bounds, "ad", "ae", "bd", "be", "ce" cover a lower bound from \( A. Γ_2 \), no new lower bounds are generated. So \( A. Γ = \{d, e\} \).
3. add "cde" (= \( I(r_2) \cap A \)): We get \( A. Γ_1 = \{d, e\} \) and \( A. Γ_2 = \emptyset \). The candidate lower bounds are "ad", "be", "ce" and "bd", "ae". Because none of them is covered by another candidate and \( A. Γ_2 = \emptyset \), \( A. Γ = \{ad, bd, ae, be\} \).

4 Performance Studies

This section provides a performance evaluation of FARMER. All our experiments were performed on a PC with a Pentium IV 2.4 Ghz CPU, 1GB RAM and a 80GB harddisk. Algorithms were coded in Standard C.

Algorithms: We compare FARMER with the interesting rule mining algorithm in [3] which is based on the Dens Miner [4]. The algorithm in [3] 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 as most existing rule mining algorithms. We also compare FARMER with the closed set discovery algorithms CHARM [27] and CLOSET+ [25], 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 upper bound and lower bounds of the interesting rule groups.

Compared with CHARM, FARMER does extra work in: 1) computing the lower bounds of Interesting Rule Groups 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 an IRG.

Datasets: We choose 5 real-life microarray datasets to analyze the performance of FARMER. The 5 datasets are clinical data on lung cancer (LC)\(^3\), breast cancer (BC)\(^6\), prostate cancer (PC)\(^7\), ALL-AML leukemia (ALL)\(^8\), and ovarian cancer (OC)\(^9\). In such datasets, the rows represent clinical samples while the columns represent the activity levels of genes/protein in the samples. There are two categories of samples in these datasets.

Table 1. Microarray Datasets

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

To discretize the datasets, we use two methods, one is the entropy-minimized\(^10\) and the other is the equal depth partition with 10 buckets. On the datasets generated from entropy-minimized discretization, the ColumnE algorithm needs several days to run even at very high minimum support and confidence thresholds and CHARM always stops within several hours after running out of memory (CLOSET+ has the same problem).

The reason is that an extremely large number of long closed sets are generated from such datasets which are relatively dense. Although FARMER is still efficient on such datasets, it is hard to get results of ColumnE and CHARM. Therefore, we use the datasets generated from the equal depth partition to report experiment results.

\(^3\)http://www.chestsurg.org
\(^6\)http://www.rri.com/publications/default.htm
\(^7\)http://www-genome.wi.mit.edu/mpg/prostate
\(^8\)http://www-genome.wi.mit.edu/cgi-bin/cancer
\(^9\)http://clinicalproteomics.steem.com/
\(^10\)the code is available at http://www.sgi.com/tech/mlc/
4.1 Varying Minimum Support

The first set of experiments (Figure 10) shows the effect of different minimum support settings as $\text{minsup}$ is varied. The graphs of the figure plot runtime for several algorithms at various settings of minimum support. Note that the y-axes in Figure 10 are in logarithmic scale. We set both $\text{minconf}$ and $\text{minchi}$ as ZERO, which disables pruning with the minimum confidence constraint and minimum chi-square constraint of FARMER.

For CHARM, $\text{minsup}$ 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 of the two figures on datasets BC and LC. CHARM also runs out of memory at $\text{minsup} = 18$ at Figure 10(e).

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, are orders of magnitude greater than the search space of FARMER, dependent on the possible number of row combinations.
combinations, on microarray datasets.

As shown in Figure 10(f), the number of Interesting Rule Groups discovered at low minsup is much larger than that at high minsup. Besides the row enumeration, the number of IRGs also affects the efficiency of FARMER. First, FARMER discovers only those interesting groups by comparison (see algorithm section, step 7). The more IRGs discovered, the more time is consumed for the comparison. The time can take 20% of the runtime of FARMER at low minsup. 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 can take up to 20% at low minsup. ColumnE also does the comparison to get interesting rules while all runtime of CHARM is used to discover closed sets.

Readers may wonder why do we choose different highest minimum support for different datasets. Our principle here is to make the runtime of FARMER about 10 seconds. We can see ColumnE and CHARM might catch up with FARMER if we further increase the highest minsup. However, the absolute time difference must be less than 10 seconds and is not interesting to compare. This is negligible compared to the difference in running time at low minsup. To make figures clear, we do not show them any more.

4.2 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 as ZERO. For all parameter settings in Figure 11, CHARM can not finish because it always runs out of memory within several hours while ColumnE always needs more than 1 day to finish. 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 since it is too slow to compare. We first ignore the lines marked with “minchi = 10” here. For datasets LC, BC, PC and ALL, we set minsup = 1, which means that minimum support pruning is nearly disabled. For dataset OC, it is even slow for FARMER to run at minsup = 1 so that we set minsup = 12.

Figure 11 shows that the run time of FARMER decreases with the increasing value of minconf on all 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 run time of FARMER when minconf increases from 85% to 99% on datasets LC, BC, PC and ALL. The reason behind this is that there are few upper bound rules whose confidences fall between 85% and 99%. We observe that nearly all IRGs discovered at 85% on datasets LC, BC, PC and ALL have a 100% confidence. As a result, FARMER does no additional pruning when minconf increases from 85% to 99% on the four datasets. On dataset OC, there are much more IRGs newly discovered at minconf = 85% than at minconf = 99% although it also has many 100% confidence rules. This make the confidence pruning effective.

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

4.3 Varying Minimum Chi Square Value

The last set of experiments were preformed 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 draw the runtime vs various minconf in Figure 11 due to the space limitation, where minconf is set the same as in section 4.2.

The pruning exploited by constraint minchi = 10 is shown to be very effective on datasets BC, PC and ALL. At some cases, the saving can be an order of magnitude larger. The pruning effect is not so obvious on datasets LC and OC. By checking the return IRGs, we found that discovered IRGs from LC and OC usually have higher chi square value. If we impose a tighter constraint by increasing minchi, the pruning based on minchi will be more effective on these two datasets. 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 pruning based on minsup, minconf and minchi are effective. In general, the execution time required by 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.

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 [24, 8, 21, 7].

Association rule can relate gene expressions to their cellular environments or categories, thus available for building accurate classifiers on microarray datasets as in [9, 12]. 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 [6], 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 mining. All existing association 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 identify the interesting (or optimal) rules with some interestingness measures [3]. The interesting rule discussed in [3] is quite similar to our interesting rule group. However, [3] randomly discovers one rule for each rule group while FARMER discovers the upper bound and lower bounds for each rule group.

The second class of rule mining algorithms aim to find all association rule satisfying user specified constraints by identifying all frequent itemsets at the key step, such as [1, 14, 11, 19]. Recently the concept of closed itemset [18] is proposed to reduce redundant itemsets and rules [26]. Several efficient mining algorithms [18, 2, 27] are proposed to mine frequent closed
items sets. On the other hand, there is some work \cite{23, 16} that investigate incorporating item constraints to reduce the number of frequent itemsets. Other work \cite{4, 15} leverages the item constraint as rule consequent and utilize minimum thresholds of confidence, support and other statistical constraints. The intention is quite similar to ours but our pruning strategies are different and specially designed for our unconventional way of row enumeration.

The third class of algorithms aim at mining predictive rules. However neither the prediction effectiveness nor the completeness of the rules generated is guaranteed. One example is the decision tree induction algorithm \cite{10}. Alternatively, some work \cite{13, 9} 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 can be applied using our interesting rule groups.

In a short paper in \cite{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 find IRGs that satisfy interestingness constraints like $minconf$ and $minchi$. The effectiveness of pruning with such constraints is evident in our experiments.

6 Conclusion

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 this bold approach yields good payoff as FARMER outperforms existing closed pattern discovery algorithms like CHARM and ColumnE by a large order of magnitude on microarray datasets.

In the future, we will look at the following two open questions: First, we plan to utilize discovered IRGs to construct classifier and compare it to existing works that build classifiers using single rules. Second, we will extend FARMER for datasets with both long columns and large number of rows by searching by column-wise first and switching to row-wise enumeration in later levels.

References

\begin{itemize}
  \item [11] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In \textit{Proc. 2000 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD’00)}.
  \item [16] R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In \textit{Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD’98)}.
  \item [20] J. Pei, J. Han, and R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In \textit{Proc. 2000 ACM-SIGMOD Int. Workshop Data Mining and Knowledge Discovery (DMKD’00)}.
\end{itemize}