Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/080241

MCR-Miner: Maximal Confident Association Rules Miner Algorithm for Up/Down-Expressed Genes

Wael Zakaria[1,*], *Yasser Kotb*[1,2,*] *and Fayed Ghaleb*[1,*]

[1] Department of Mathematics-Computer Science, Faculty of Science, Ain Shams University, Abbassia, Cairo, Egypt
[2] Information Systems Department, College of Computer and Information Sciences, Al-Imam Muhammad ibn Saud Islamic University, Riyadh, KSA

Received: 13 Apr. 2013, Revised: 14 Aug. 2013, Accepted: 16 Aug. 2013
Published online: 1 Mar. 2014

Abstract: DNA microarrays allow simultaneous measurements of expression levels for a large number of genes across a number of different experimental conditions (samples). The algorithms for mining association rules are used to reveal biologically relevant associations between different genes under different experimental samples. This paper presents a new column-enumeration based method algorithm (abbreviated by MCR-Miner) for *mining maximal high confidence association rules for up/down-expressed genes*. MCR-Miner algorithm uses an efficient *maximal association rules tree* data structure (abbreviated by MAR-Tree). MAR-tree enumerates (lists) all genes with their binary representations, the binary representation of a gene saves the status (normal, up, and down-expressed) of a gene in all experiments. The binary representation has many advantages, scan the dataset only once, the measurements of confidences for association rules are made in one step, and it makes MCR-Miner algorithm easily finds all maximal high confidence association rules. In the experimental results on a real microarray datasets, MCR-Miner algorithm attained very promising results and outperformed other counterparts.

Keywords: Data mining, DNA microarray, mining association rules, closed itemsets, maximal high confidence association rules

1 Introduction

Gene expression is the process of transcribing DNA sequences into mRNA sequences, which are later translated into amino acid sequences called *proteins*. The number of copies of the produced RNA is called the *gene expression level*. Each normal gene has a rate of expression level *e*, *up-expressed gene* is the gene with expression level $>$ e, *down-expressed gene* is the gene with expression level $<$ e. The regulation of gene expression level is essential for proper cell function. Microarray technologies provide the opportunity to measure the expression level of tens of thousands of genes in cells simultaneously. Usually, the expression level is correlated with the corresponding protein made under different conditions (samples) [1,2,3].

The microarray dataset can be seen as an M$\times$N matrix G of expression values; where the rows represent genes g_1, g_2, ..., g_m and the columns represent different experimental conditions (samples) s_1, s_2, ..., s_n. Each element G[i,j] represents the expression level of the gene g_i in the sample s_j (see Table 1). The matrix usually contains a huge data, therefore, data mining techniques are used to extract useful knowledge from such matrices [3,4].

Mining association rules is currently a vital data mining technique for many applications [4,5,6]. Mining association rules technique is applied to microarray dataset to extract interesting relationships among sets of genes [2,4,13]. Let g_1 and g_2 be up-expressed genes and $\overline{g_3}$ be down-expressed gene (see section 4), then the association rule $g_1 \rightarrow g_2,\overline{g_3}$ (with support 80% and confidence 90%) unmasks a relation among the genes g_1, g_2, and g_3. this relation asserts that all of the genes g_1, g_2, and g_3 appear in 80% of the microarray samples and if g_1 is up-expressed then g_2 is up-expressed and $\overline{g_3}$ is down-expressed with probability 90%.

In order to mine association rules in microarray dataset, the data is pre-processed by applying the logarithms procedure to ensure that the data is suitable for analysis. The logarithms procedure transforms DNA microarray data from the raw color intensities into log

* Corresponding author e-mail: Wael_Abdallah@sci.asu.edu.eg, yasser_kotb@sci.asu.edu.eg, fmghaleb@yahoo.com

color intensities; where [1]. Then, based on a predefined threshold, the transformed dataset is discretized into ternary valued matrix, such that each gene value is mapped into 1, 0, or -1 for up-expressed, non-expressed, or down-expressed gene respectively as shown in Table 2.

Table 1: Microarray Dataset

	s1	s2	s3	s4	s5
a	0.039	0.597	0.235	0.267	0.343
b	0.633	0.04	-0.01	0.323	0.252
c	-0.15	0.266	0.41	0.3	0.35
d	-0.32	-0.14	-0.25	-0.45	-0.24
e	0.28	0.34	0.466	0.23	0.23
f	0.26	0.42	-0.1	-0.3	0.485

Table 2: Discretized Microarray.
The expression gene is converted into 1, 0, or -1 if it is $\geq$, =, or $\leq$ -0.2 respectively.

	s1	s2	s3	s4	s5
a	0	1	1	1	1
b	1	0	0	1	1
c	0	1	1	1	1
d	-1	0	-1	-1	-1
e	1	1	1	1	1
f	1	1	0	-1	1

This paper presents a new column(gene)-enumeration based method algorithm. The proposed algorithm is called *MCR-Miner* which overcomes both the computational time and memory explosion problems of column-enumeration used in many algorithms for mining microarray datasets [4]. MCR-Miner scans the microarray dataset only once to obtain a list of all genes in which, each gene *g* is associated with a ternary representation; where each element in the representation shows whether the gene is up-expressed, non-expressed, or down-expressed at the corresponding sample. Therefore, every gene is split into two nodes, one for up-expressed gene and the second for the down expressed gene; where each node saves the binary representation of a gene (up or down) (see subsection 4.1). These nodes are saved in MAR-tree, the structure of MAR-tree is the back bone of the MCR-Miner algorithm. MCR-Miner using MAR-tree easily finds with high speed all maximal high confidence association rules. The experimental results show that the MCR-Miner algorithm is faster than the row-enumeration based methods MAXCONF [15] and RERII [16]. Since, RERII and MAXCONF are better than other column-enumeration based method like CHARM [17], As a result, MCR-Miner algorithm is also faster than the column-enumeration based method CHARM.

The rest of the paper is organized as follows. Section 2 introduces the mining association rules problem. Section 3 presents related works. Section 4 explains the proposed MCR-Miner algorithm for extracting all maximal high confidence association rules for up/down-expressed genes. Section 5 shows the experimental results of MCR-Miner. Section 6 concludes the paper.

2 Mining Association Rules

Mining association rules technique extracts interesting relationships among sets of items (genes) in a large dataset. One of the most famous applications of this technique is *market basket analysis* [5,6] where the objective is to find the relationships between the purchased items under different transactions. Also, mining association rules is applied on *microarray datasets* in order to find the relationships between genes under different samples. In this section, using the transactions (samples) dataset from Table 3, some notations are introduced [5,6,18].

Table 3: Microarray Transactions(Samples) Dataset.
The gene $\overline{d}$ at sample 1 means that the gene d is down-expressed at this sample (where its value=-1 in table 2)

tiD (sample is)	Transactions (samples)
1	b,$\overline{d}$,e,f
2	a,c,e,f
3	a,c,$\overline{d}$,e
4	a,b,c,$\overline{d}$,e,$\overline{f}$
5	a,b,c,$\overline{d}$,e,f

Definition 1 ***(Association Rules).*** *Let* $I=\{i_1,i_2,...,i_n\}$ *be a set of* n *items (genes). A subset* $T\subseteq I$ *is called a* transaction (sample). *The transactions dataset* D *is a set of transactions; where each transaction has a unique id called* tid. *In other words,* $D=\{<tid,T>; T\subset I, tid\in\{1, 2,..., k\}; k=|D|\}$. *An association rule is a pair of itemsets(X,Y) where* $X,Y\subseteq I$ *and* $X\cap Y=\phi$, *and is denoted by* $X\rightarrow Y$. *The itemsets (set of items) X and Y are called antecedent and consequent of the rule* $X\rightarrow Y$, *respectively.* ***Convention****:* $T\in D$ *denotes that* $\exists tid$ *such that* $<tid,T>\in D$.

Definition 2 ***(Measurements of Association Rules).*** *An association rule* $X\rightarrow Y$ *has two measurements: support and confidence. They are defined, with respect to a transactions (samples) dataset D, as follows:*

$$supp_D(X \to Y) = \frac{supp_D(X \cup Y)}{|D|}$$
$$conf_D(X \to Y) = \frac{supp_D(X \cup Y)}{supp_D(X)}$$
$$where\ supp_D(X) = |T; X \subseteq T, T \in D|$$

Definition 3 ***(Frequent Itemset).*** *The itemset X is called* frequent *if supp(X)≥minsup; where minsup is a user defined threshold.*

Definition 4 ***(Strong/Confident Association Rules).*** *The rule X→Y is called* strong or confident *if supp(X→Y)≥minsup and conf(X→Y)≥minconf; where minconf is another user defined threshold.*

The process of mining confident association rules is performed in two steps [6, 18]:

1. Generate all frequent n-itemsets (set of n items).
2. Using all frequent n-itemsets, generate all strong/confident association rules X→Y, where X and Y are frequent n-itemsets.

The dataset such "*market basket analysis*" has the property that the number of items in the dataset is less than the number of transactions. This kind of dataset called *sparse*, i.e., the longest frequent itemsets is relatively short. However, there are many real-life datasets such as microarray datasets, that the number of items (genes) is greater than the number of transactions (samples). This kind of dataset called *dense*, i.e., they contain very long frequent itemsets (genesets). Therefore, generating all frequent itemsets in such dense datasets requires large memory. Hence, recent algorithms prevent this problem by expanding only frequent closed itemsets [15, 16, 19, 21].

Definition 5 ***(Frequent Closed Itemset).*** *The frequent itemset X is called a frequent closed itemset if ∄ a frequent itemset Y such that X⊆Y and supp(X)=supp(Y).*

For example, if AB and ABC are two frequent itemsets with supp(AB)= supp(ABC), then AB is called *non-closed* itemset.

With respect to microarray datasets, the set of all mined confident association rules from frequent closed itemsets might still be very large. Therefore, some algorithms mine only the maximal confident association rules from microarray datasets.

Definition 6 ***(Maximal Confident Association Rules).*** *A confident rule r_1 is called maximal confident association rule, if ∄ other confident rule r_2 such that*

1. *antecedent(r_1)=antecedent(r_2), and*
2. *consequent(r_1)⊂consequent(r_2).*

For example, if the rules A→BCD and A→BC are confident, then A→BC is called *non-maximal confident association rule*.

3 Related Works

The most algorithms of frequent pattern mining based on one the following two methods [4]:

1. *Column(item)-enumeration based method*: This method uses breadth-first search to enumerate each 1-itemset. Repeatedly, join (k-1)-itemsets with itself to get a k-itemsets; k=2, 3, ..., L; where L is the longest-frequent itemsets. **Apriori algorithm** [7] is the first mining association rules algorithm that pioneered the use of support-based pruning to control the exponential growth of candidate itemsets. It uses pruning principle that is state of *"If there is any itemset which is infrequent, its superset should be infrequent"*. However, Apriori algorithm pass over the original dataset L times; L is the longest frequent itemsets. Also, the generation of candidates itemsets takes exponential time. **Eclat** [11] and **Quick-Apriori** [10] algorithms overcome the problems of traversing the dataset L times by using the bottom-up search procedure that generates the frequent itemsets by intersecting the tids-lists (transaction TIDs) of all distinct pairs of itemsets. This procedure is repeated until all frequent itemsets have been enumerated. Apriori, Eclat, and Quick-Apriori show good performance with sparse datasets such as marketbasket data, but these algorithms face difficulties when applying to dense datasets such as microarrays. this difficulties according to the number of items (genes) is more greater than transactions (samples). In these algorithms, in order to produce all frequent itemsets of length L, they produce all 2^L of its subsets. This exponential complexity restricts these algorithms to discover only short patterns.
   MaxEclat [9] and **Max-Miner** [8] optimize Apriori by exploiting additional pattern constraints by mining only the longest of the maximal frequent itemsets. Max-Miner algorithm outperforms than MaxEclat; where Max-Miner attempts to look ahead through the search in order to quickly identify long frequent itemsets. By pruning all the non-maximal frequent itemsets in early steps. However, it still traverses the dataset more than once. **CHARM** [17] and **CLOSET** [21] optimize Apriori algorithm by mining only closed frequent itemsets (see Detention 5); the set of closed frequent itemsets is a lot smaller than the set of all frequent itemsets. CLOSET with compressed FP-tree structure is efficient and scalable than CHARM. However, using Max-Miner or CLOSET algorithm with dense datasets microarrays still poses great challenges.
2. *Row(transaction)-enumeration based method*: This method uses a depth-first search to enumerate each transaction; each transaction is assigned to a support of value 1. A successive intersecting processes of each transaction with the other transactions in the dataset, resulting in a transaction with smaller number of

intersected items. This process continues recursively until no smaller itemsets can be formed. The row-enumeration based method **CARPENTER** algorithm [12] is used to mining frequent closed itemsets. CARPENTER algorithm outperforms than column-enumeration based method CLOSET and CHARM. **RERII** [16] algorithm is similar to CARPENTER but it optimizes its process by utilize three support pruning methods, these pruning methods reduce the used spaces and remove the redundant frequent closed itemsets. In microarray datasets, the RERII algorithm is faster than CARPENTER.

MAXCONF [15] algorithm is closely related to RERII in which the generation of nodes is similar. But it depends only on confidence pruning (i.e. free support pruning) to produce the rules with high confidence and low support. In this algorithm, the rules with only one gene on the LHS are created. (i.e., create all rules on the form X $\rightarrow$ Y, where $|X|$=1). MAXCONF exploits two confidence pruning methods in order to prune the search space and eliminating the non-maximal rules in early steps as in Max-Miner. MAXCONF algorithm is better than RERII. These row enumeration based method algorithms are faster than the column enumeration based method algorithms when applying on dense datasets such as microarray datasets. Note that, recent paper [4] noted that "a comparative analysis using several known datasets revealed that without using any support threshold MAXCONF provide excellent results".

This paper presents a new algorithm based on the column(gene)-enumeration based method. The proposed algorithm is called **MCR-Miner** which overcomes both the computational time and memory explosion problems of the relative column-enumeration based method algorithms such as Apriori [7] and MAX-Miner [8]. MCR-Miner is used for mining all maximal high confidence association rules for up/down-expressed genes like the row-enumeration based method MAXCONF algorithm [15]. The experimental results show that MCR-Miner is faster than MAXCONF algorithm. As consequents it is faster than the mentioned algorithms such as RERII, CARPENTER, CHARM, MAX-Miner, and Apriori.

4 MCR-Miner Algorithm

This section introduces the *(MCR-Miner)* algorithm based on the column (gene) enumeration method and only confidence pruning in order to mine maximal high confidence association rules for up/down-expressed genes in microarray dataset. The mined rules have the form LHS$\rightarrow$RHS (conf$\geq$minconf); |LHS|=1. The samples dataset in Table 2 is used as running example to illustrate the steps of MCR-Miner algorithm; minconf is set to be 50%. The following four subsections show the steps of MCR-Miner algorithm:

4.1 Discretization

The normalized microarray dataset is usually represented as a series of continuous numbers. Discretization is the process of transformation from continuous data into discrete data. There are many discretization techniques [22]. In this paper, the threshold method is used in order to discretize data; each gene expression is converted into one of the three discrete values 1, 0, or -1 for up-expressed, non-expressed, or down-expressed gene respectively. Therefore, in order to mine association rules, microarray matrix G is converted into matrix G' (as shown in Table 2) depending on the particular threshold cut value *c* [22]:

$$G'[i,j] = \begin{cases} 1 & \text{G[i,j]}\geq \text{c,(g}_i \text{ is up-expressed at sample j)} \\ -1 & \text{G[i,j]}\leq \text{-c,(g}_i \text{ is down-expressed at sample j)} \\ 0 & \text{Otherwise, (g}_i \text{ is non-expressed at sample j)} \end{cases}$$

After discretization, each gene (Table 2) can be represented by ternary representation (see Definition 7).

Definition 7 ***(Ternary representation of a gene).*** *A ternary representation of a gene* g*; (*TR_g*) =* (a_1 a_2 ... a_n) *with* n *is the number of samples; where*

$$a_j = \begin{cases} 1 & g \text{ is up-expressed at sample } j \\ -1 & g \text{ is down-expressed at sample } j \\ 0 & g \text{ is non-expressed at sample } j \end{cases}$$

For example, Table 2 shows the discretized microarray dataset ; where the threshold cut value c=0.2. For example, the gene f with ternary representation TR_f=110(-1)1 means that the gene f is up-expressed in samples 1, 2, and 5, it non-expressed in sample 3, and it down-expressed in sample 4. In the discretized microarray dataset, each gene g contains 1 and -1 splits into two genes:

1. **Up-expressed gene (*g*)**: A ternary representation of gene g is converted into binary representation (Definition 8) in which the zeros are set instead of negative ones see Table 4.
2. **Down-expressed gene ($\overline{g}$)**: A ternary representation of gene g is converted into binary representation (Definition 9) in which the zeros are set instead of positive ones see Table 4.

For example, Table 4 shows the up-expressed and down-expressed genes dataset in which the gene f is split into two genes f and $\overline{f}$ with binary representation are 11001 and 00010 respectively.

Note that, the gene g with only positive ones or only negative ones is converted into g or $\overline{g}$ respectively.

Definition 8 ***(Binary representation of a up-expressed gene).*** *A binary representation of a up-expressed gene* g*; (*BR_g*) = (*a_1 a_2 ... a_n*) with* n *is the number of samples; where*

$$a_j = \begin{cases} 1 & g \text{ is up-expressed at sample } j \\ 0 & g \text{ is (non/down)-expressed at sample } j \end{cases}$$

Definition 9 ***(Binary representation of a down-expressed gene).*** *A binary representation of a down-expressed gene* $\overline{g}$*; (*$BR_{\overline{g}}$*) = (*a_1 a_2 ... a_n*) with* n *is the number of samples; where*

$$a_j = \begin{cases} 1 & g \text{ is down-expressed at sample } j \\ 0 & g \text{ is (non/up)-expressed at sample } j \end{cases}$$

Table 4: Up/Down-Expressed Genes Dataset

	s1	s2	s3	s4	s5
a	0	1	1	1	1
b	1	0	0	1	1
c	0	1	1	1	1
$\overline{d}$	1	0	1	1	1
e	1	1	1	1	1
f	1	1	0	0	1
$\overline{f}$	0	0	0	1	0

4.2 MAR-Tree Structure

MCR-Miner algorithm uses *maximal association rules* tree data structure to enumerate (list) all genes by constructing a tree that have the following three levels only:

– **Level 1**: contains the root of the tree that refers to all genes as children nodes at level 2.
– **Level 2**: contains set of nodes, each node N consists of the following five fields:
  – **Antecedents_genes_set(ant_set)**: list of all alphabetically sorted genes with the same binary representation (see Definition 8 and 9). For example, in table 4, the gene *a* and gene *c* have the same binary representation, then they are combined in a single node.
  – **Binary_Representation_of_ant_set(**BR_2**)**: the binary representation of a node N is equal to the binary representation of any gene belong to ant_set i.e., $N.BR_2 = BR_g$; g (up/down) is any gene ∈ ant_set.
  – **Support_of_ant_set(**$supp_2$**)**: the number of ones in $N.BR_2$ (i.e., the numbers of samples in which the genes of ant_set are expressed[1]).
  – **Consequents_Set(**$conseq\_set_2$**)**: list of all alphabetically sorted genes which expressed in the all samples whenever the genes of N.ant_set are expressed. This means that: $N.conseq\_set_2 = \bigcup_{S \in AS(N) \wedge S.BR_2 > N.BR_2} (S.ant\_set)$[2].
  – **Children_Set(children)**: contains set of children nodes of node N. These nodes are created at level 3.
– **Level 3**: contains a set of children nodes, each child node C of a parent N contains the following five fields:
  – **Consequents_genes_Set(**$conseq\_set_3$**)**: list of all alphabetically sorted genes of S.ant_set for which genes of N.ant_set are expressed in the all samples whenever genes of (N.ant_set ∪ S.ant_set) are expressed; S ∈ AS(N) and $supp(N.BR_2 \wedge S.BR_2) / N.BR_2 \geq$ minconf. This means that: $C.conseq\_set_3 = \bigcup_{S \in AS(N) \wedge conf \geq minconf} (S.ant\_set)$ $conf=supp(N.BR_2 \wedge S.BR_2) / N.BR_2$.
  – **Binary_Representation_of_ant_set union conseq_set**$_3$ **(**BR_3**)**: $C.BR_3 = N.BR_2 \wedge S.BR_2$; S ∈ AS(N).
  – **Support_of_ant_set union conseq_set**$_3$**(**$supp_3$**)**: the number of ones in $C.BR_3$ (i.e., the numbers of samples in which the genes in (N.ant_set ∪ $C.conseq\_set_3$) are expressed).
  – **Generate non-maximal rule (GNMR)**: this field is set to be true, if the child C will generate non-maximal rule (Definition 10).
  – **Participate (part)**: contains all the indices of the children nodes at level 3 which participate to generate the child C (Definition 11). For example, if the two children C_a and C_b are combined to form new child C_k. Therefore, C_k.part={a,b}, C_a.GNMR=true, and C_b.GNMR =true.

Definition 10 ***(Generate non-maximal rule).*** *A child node* C_i *at level 3 is set true to the field generate non-maximal rule (GNMR) if* ∃ *a node* C_k *at level 3 such that* $C_k.BR_3 < C_i.BR_3$ *or* $(supp(C_i.BR_3 \wedge C_k.BR_3) / parent(C_i).supp_2) \geq minconf$.

Definition 11 ***(partcipate).*** *A child node* C_i *at level 3 is called participate to form a node* C_k *at level 3, if one of the following two cases holds:*

1. *If* $C_k.BR_3 < C_i.BR_3$*, then the child node* C_i *participates to form* C_k*. In this case, add index i to* C_k*.part. In addition,* C_i*.GNMR=true.*
2. *If* ∃ *child node* C_p *at level 3;* $C_p.BR_3 \wedge C_i.BR_3 = C_k.BR_3$ *and* $supp(C_i.BR_3 \wedge C_p.BR_3) / parent(C_i).supp \geq minconf$*, then the children nodes* C_i *and* C_p *participate to form a child node* C_k*. In this case, add the two indices* {*i*,*p*} *to* C_k*.part. In addition,* C_i*.GNMR=true and* C_p*.GNMR=true.*

[1] expressed means up-expressed or down-expressed

[2] AS(N) is the all siblings of node N and S.BR > N.BR means that $S.BR_2 \wedge N.BR_2 = N.BR_2$

4.3 MCR-Miner Algorithm

To generate all maximal high confidence association rules for up/down-expressed genes, MCR-Miner algorithm (see Algorithm 1) works as follows:

1. MCR-Miner algorithm scans (Line 2) the up/down-expressed genes dataset (Table 4), then saves each gene and its binary representation BR into a single node *n* at level 2 in a tree. At line 3, the nodes with the same binary representation are combined into single node (see Fig. 1). At line 4, the procedure compare (Algorithm 2) is invoked for generating all children nodes of all nodes at level 2 in a tree (see the next subsection).
   The conseq_set$_2$ and children fields are initialized with empty.
2. Procedure compare (Algorithm 2) traverses each node n_i at level 2 (Figure 1); i=1, 2, ..., #(children of roots)-1 (Lines 2-3), for each node n_i the following three steps should be followed:
   2.1 **Create all children nodes and conseq_set_2 of n_i**
   Let d= $n_i.BR_2 \wedge n_k.BR_2$; k=i+1, i+2, ..., #(children of roots) (Lines 4-6 Algorithm 2), the following steps should be followed:
   ► **Update node n_i**. In order to add genes to n_i.conseq_set$_2$ or new child to node n_i.children, one of the following cases holds (Lines 7-16 Algorithm 2):
   - d=$n_i.BR_2$ (i.e., $n_i.BR_2 < n_k.BR_2$) which means that the genes of n_k.ant_set are expressed in the all samples whenever the genes of n_i.ant_set are expressed. In this case, add n_k.ant_set to n_i.conseq_set$_2$.
   - d $\neq$ $n_i.BR_2$ which means that there exist samples in which the genes of n_k.ant_set and the genes of n_i.ant_set are expressed simultaneously. If (supp(d)/n_i.supp$_2$) $\geq$ minconf, then add new child c to n_i; c=newNodeAtLevel3 (n_k.ant_set, d, supp (d), false, ϕ) see function newNodeAtLevel3 at Algorithm 3.

   ► **Similarly, update node n_k**. In order to add genes to n_k.conseq_set$_2$ or new child to node n_k.children, one of the following cases holds (Lines 17-26 Algorithm 2)
   - d=$n_k.BR_2$, then add n_i.ant_set to n_k.conseq_set$_2$.
   - d $\neq$ $n_k.BR_2$, if (supp(d)/n_k.supp$_2$) $\geq$ minconf, then add new child c to n_k; c = newNodeAtLevel3 (n_i.ant_set,d, supp (d), false, ϕ)(Algorithm 2).

   Lines 28-35 Algorithm 2 will be discussed in the next subsections.

   Example Fig. 2 shows the output tree of step 2 of MCR-Miner algorithm. In the figure, The genes of node n_1.ant_set=*ac* are up-expressed in all samples whenever the gene of node n_6.ant_set=$\overline{f}$ is down-expressed (i.e., $n_6.BR_2$ =00010 $< n_1.BR_2$ = 01111), then n_1.ant_set will be added to n_6.conseq_set$_2$, i.e., n_6.conseq_set$_2$={ac}. Similarly, the genes of nodes n_2.ant_set, n_3.ant_set, and n_4.ant_set are (up/down)-expressed in all samples whenever the gene of node n_6.ant_set is down-expressed, then n_6.conseq_set$_2$={ac} $\cup$ {b} $\cup$ {$\overline{d}$} $\cup$ {e} = abc$\overline{d}$e[3]. The node n_1 and n_3; $n_1.BR_2 \wedge n_3.BR_2$=01111 $\wedge$ 10111=00111, and supp(00111)/n_1.supp$_2$ = 3/4 = 75% $\geq$ minconf=50%, then new child node C_p of node n_1 is created; C_p.conseq_set$_3$ = n_3.ant_set = {$\overline{d}$}, $C_p.BR_3$=00111, C_p.supp$_3$=3, C_p.GNMR = false, and C_p.part = ϕ. Also, supp(00111)/ n_3.supp$_2$ = 3/4 = 75% $\geq$ minconf=50%, then new child node C_p of node n_3 is created; C_p.conseq_set$_3$ = n_1.ant_set = {ac}, $C_p.BR_3$=00111, C_p.supp$_p$=3, C_p.GNMR=false, and C_p.part=ϕ.

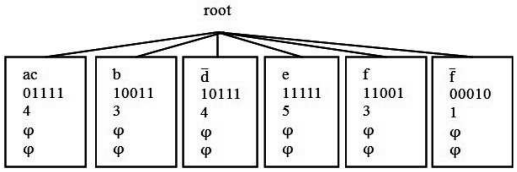

Fig. 1: The Column (gene) Enumeration Tree for Up/Down-Expressed Genes Dataset

Algorithm 1 The MCR-Miner algorithm

```
1: procedure MCR-MINER(Dataset D,float minconf)
2:     root.children=scan up/down-expressed genes dataset D
   and save each genes in a single node;
3:     root.children=combine the nodes which have the same
   binary representation;
4:     compare(root,minconf);
5: end procedure
```

2.2 **Producing children nodes of n_i that produce all maximal high confidence association rules**
In this step, the children nodes of n_i at level 3 need more processes to produce all maximal high confidence association rules. In order to do this, the following steps should be followed:

i. All children nodes of n_i are moved to children_buffer list (i.e., n_i.children will become empty) (Lines 28-29 at Algorithm 2).
ii. Each child node $C_k \in$ children_buffer; k=1, 2, ..., #(children_buffer) will be inserted into n_i.children by calling the procedure Insert (C_k, n_i, minconf) (Lines 30-32 Algorithm 2). The procedure Insert(C_k, n_i, minconf) (Algorithm 4) firstly combines the child node C_k with existed child C_j of n_i if C_k and C_j have the same binary

[3] abc$\overline{d}$e is sorted set of genes a, b, c, $\overline{d}$, and e

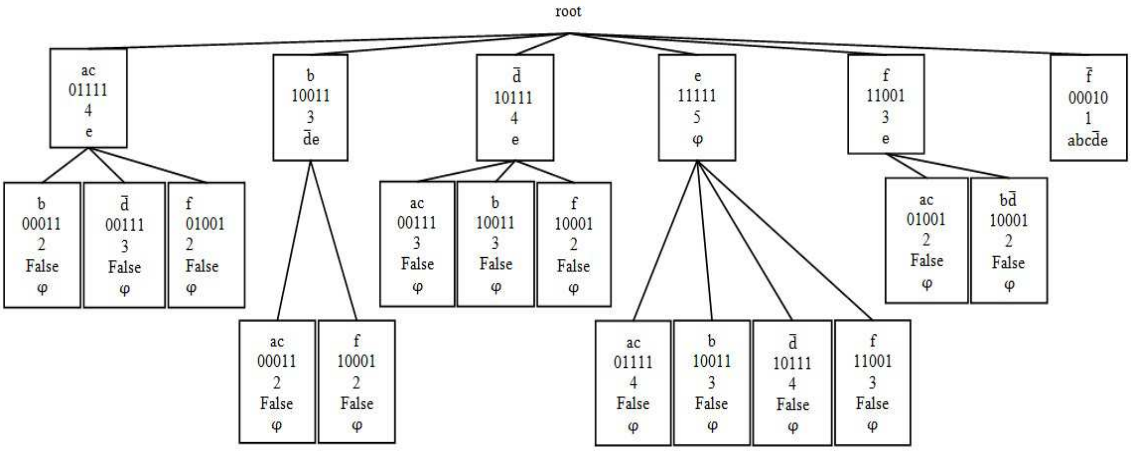

Fig. 2: Tree Contains All conseq_set2 and Children of All Nodes at Level2

Algorithm 2 Procedure Compare

```
1: procedure COMPARE(MAR-Tree root,float minconf)
2:     for i=1:#(children nodes of root)-1 do
3:         n_i= the node at position i
4:         for k=i+1:#(children nodes of root) do
5:             n_k= the node at position k
6:             d=n_i.BR_2 ∧ n_k.BR_2;
7:             if (n_i.BR_2=d) then                ▷ n_i.BR_2 < n_k.BR_2
8:                 add n_k.ant_set to n_i.conseq_set_2 ;
9:             else
10:                conf= supp(d)/n_i.supp;
11:                if (conf ≥ minconf) then
12:                    child=newNodeAtLevel3(n_k.ant_set,d,
13:                    supp(d),false, φ);
14:                    add c to n_i.children;
15:                end if
16:            end if
17:            if (n_k.BR_2=d) then                ▷ n_i.BR_2 > n_k.BR_2
18:                add n_i.ant_set to n_k.conseq_set_2
19:            else
20:                conf=supp(d)/n_k.supp;
21:                if (conf ≥ minconf) then
22:                    child=newNodeAtLevel3 (n_i.ant_set,
23:                    d, supp(d),false, φ);
24:                    add c to n_k.children;
25:                end if
26:            end if
27:        end for
28:        buffer_children=n_i.children
29:        clear n_i.children
30:        for each C_k in buffer_children do
31:            Insert (C_k, n_i, minconf);
32:        end for
33:        remove all C with C.GNMR=true;
34:        GenerateMaximalHighRules(n_i);
35:        clear n_i and their children;
36:    end for
37: end procedure
```

Algorithm 3 Function Create New Node at Level 3

```
1: function NEWNODEATLEVEL3(conseq_set, BR, supp,
   GNMR , part): node at level 3.
2:     construct new node at level 3;
3:     node.conseq_set_3= conseq_set;
4:     node.BR_3= BR;
5:     node.supp_3= supp;
6:     node.GNMR= GNMR;
7:     node.part= part;
8:     return node;
9: end function
```

representation (Lines 2-7). Otherwise, the child node C_k will be inserted at position p (Line 8) then it will be compared with all existed children nodes C_j; $C_j \in n_i$.children and C_j.GNMR=false (Lines 9-10) according to the following cases (Lines 15-37):

A. If genes of C_j.conseq_set$_3$ are expressed in all samples whenever genes of C_k.conseq_set$_3$ are expressed (C_k.BR$_3$ < C_j.BR$_3$,) then the child node C_k will be updated with genes of C_j.conseq_set$_3$, C_j.GNMR=true, and add j to C_k.part [i.e., C_j participates to form C_k].

B. If genes of C_k.conseq_set$_3$ are expressed in all samples whenever genes of C_j.conseq_set$_3$ are expressed (C_j.BR$_3$ < C_k.BR$_3$), then a child node C_j will be updated with genes of C_k.conseq_set$_3$, C_k.GNMR=true , and add the index p to C_j.part; p is the position of C_k [i.e., C_k participates to form C_j].

C. If supp(d)/n_i.supp$_2$ ≥ minconf then add a new child node C_{kj} into n_i.children; C_{kj}.conseq_set$_3$ = C_j.conseq_set$_3$ ∪ C_k.conseq_set$_3$,C_{kj}.BR$_3$=d, C_{kj}.supp$_3$ = supp(d), C_{kj}.GNMR = false, C_{kj}.part=C_{kj}.part ∪ {p,j}. **Note that**, C_k and C_j participate to form C_{kj}, then C_k.GNMR = true,

and C_j.GNMR=true. **Recursively**, call the procedure insert to insert the new child C_{kj} to n_i.

D. Otherwise (C_k cannot form new node from C_j), then C_k is compared using these four steps with all children nodes that participated to form C_j.

Algorithm 4 Procedure Insert

```
1: procedure INSERT(nodeAtLevel3 C_k,nodeAtLevel2 n_i,float minconf)
2:     for each node C_j in n_i.children do
3:         if C_k.BR_3 =C_j.BR_3 then
4:             add C_k.conseq_set_3 to C_j.conseq_set_3;
5:             return;
6:         end if
7:     end for
8:     add C_k at end position p of n_i.children
9:     for each node C_j in n_i.children do
10:        if C_j.GNMR= false then
11:            d= C_k.BR_3 ∧ C_j.BR_3;
12:            if supp(d)=0 then
13:                continue;
14:            end if
15:            if d= C_k.BR_3 then            ▷ C_k.BR_3 < C_j.BR_3
16:                add C_j.conseq_set_3 to C_k.conseq_set_3;
17:                add j to C_k.part;
18:                C_j.GNMR=true;             ▷ n_j form node C_k
19:                C_k.GNMR=false;
20:            else if d=C_j.BR_3 then        ▷ n_j.BR_3 < n_k.BR_3
21:                add C_k.conseq_set_3 to C_j.conseq_set_3;
22:                add index p to C_j.part;
23:                C_k.GNMR=true;
24:                C_j.GNMR= false;
25:            else if supp(d)/n_i.supp2 ≥ minconf then
26:                node C_kj=
27:                newNodeAtLevel3 (C_j.conseq_set_3 ∪
28:                C_k.conseq_set_3,d,supp(d),false,{j,p});
29:                C_k.GNMR=true; C_j.GNMR=true;
30:                Insert(C_kj,n_i,minconf).   ▷ recursive call
31:            else if ∼ empty(C_j.part) then
32:                ▷ the node cannot produce new node from C_j
33:                for each index q in C_j.part do
34:                    compare C_q and C_k using four
35:                    cases of this procedure;
36:                end for
37:            end if
38:        end if
39:    end for
40: end procedure
```

Example, Fig. 3 shows the final tree after creating all children nodes that will produce maximal high confidence rule for up/down-expressed genes. In this figure, the processes of inserting the children nodes to $n_{\overline{d}}$ ($n_{\overline{d}}$.ant_set=$\overline{d}$) should be as follows:

◇ The child node C_{ac}; C_{ac}.GNMR=false will be inserted without any comparison.

◇ The child node C_b will be compared with C_{ac}; supp ($C_b.BR_3 \wedge C_{ac}.BR_3$)/$n_{\overline{d}}$.supp$_2$ ≥ minconf. Then a new child C_{abc} is created; C_{abc}.conseq_set$_3$=C_{ac}.conseq_set$_3$ ∪ C_b.conseq_set$_3$, $C_{abc}.BR_3 = C_b.BR_3 \wedge C_{ac}.BR_3$, C_b.supp=supp($C_b.BR_3 \wedge C_{ac}.BR_3$), C_{abc}.GNMR = false, and C_{abc}.part = {1, 2}; 1 and 2 are the indices of nodes C_{ac} and C_b respectively. Moreover, C_{ac}.GNMR=true and C_b.GNMR=true.

◇ The child node C_f will be compared with all children nodes C of $n_{\overline{d}}$; C.GNMR=false. We find that C_f is compared only with child C_{abc}, but C_f did not form any result with C_{abc}, then C_f will be compared with all children nodes that participated to form C_{abc}. I.e., C_f will be compared with C_b and C_{ac}; $C_f.BR_3 < C_b.BR_3$, then C_f.coneq_set$_3$=bf, C_f.part={2}; 2 is the index of node C_b. But C_f and C_{ac} not produce any child node.

2.3 **Generate all maximal high confidence association rules for up/dow-expressed genes of node n_i** Finally, after all children are created for node n_i (see Fig. 3). All children nodes $C_k \in n_i$.children with C_i.GNMR=true will be pruned, because these nodes will produce non-maximal rules (line 33 Algorithm 2) then the procedure GenerateMaximalHighRules (Line 34 Algorithm 2) is invoked. The procedure GenerateMaximalHighRules (Algorithm 5) checks each child node $C_k \in n_i$.children; C_k.GNMR=false in order for producing all maximal high confidence association rules of node n_i. All extracted maximal high confidence association rules from our algorithm shown in Fig 4. After creating all maximal high confidence association rules from n_i, n_i with its children will be pruned (Line 35 Algorithm 2)

Algorithm 5 Procedure to Mine All Maximal High Confidence Association Rules

```
1: procedure GENERATEMAXIMALHIGHRULES(nodeAtLevel2 N_i)
2:     for each C_j in N_i.children do
3:         for each gene g ∈ N_i.ant_set do
4:             Form a rule on the following form
5:             g → N_i.ant_set-{g} ∪ N_i.conseq_set_2 ∪
6:             C_j.consq_set_3;
7:         end for
8:     end for
9: end procedure
```

Advantage of MCR-Miner algorithm

1. It mines all maximal high confidence association rules.
2. Binary representation saves the memory and speeds up the intersection processes. In addition, binary representation makes MCR-Miner algorithm scans the

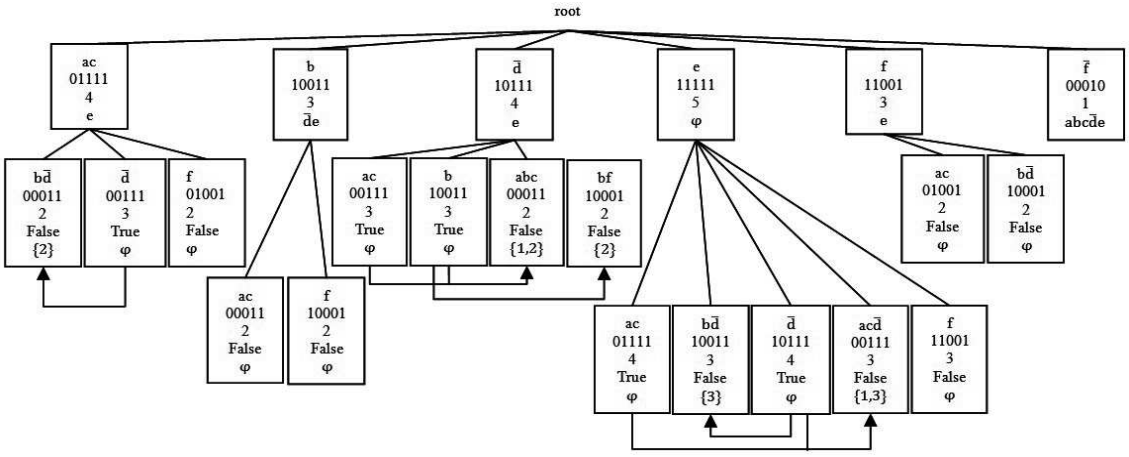

Fig. 3: Tree Contains All Nodes Which Mine Maximal High Confidence Association Rules

a $\rightarrow$ bc$\bar{d}e$:50%	$\bar{d} \rightarrow$ abce :50%
a $\rightarrow$ cef :50%	$\bar{d} \rightarrow$ bef :50%
c $\rightarrow$ ab$\bar{d}e$:50%	e $\rightarrow$ b$\bar{d}$:60%
c $\rightarrow$ aef:50%	e $\rightarrow$ f :60%
b $\rightarrow$ ac$\bar{d}$e :66.66%	e $\rightarrow$ ac$\bar{d}$:60%
b $\rightarrow$ $\bar{d}$ef :66.66%	$\bar{f} \rightarrow$ acb$\bar{d}$e :100%

Fig. 4: Mined Maximal High Confidence Association Rules.

dataset only once and the measurements of confidence easier.

3. The tree has 3 levels only; this saves both used space and time.
4. It overcomes both the computational time and memory explosion problems of column-enumeration based method algorithms. Also, it is better than row-enumeration based method algorithm like MAXCONF [15]. As consequent, MCR-Miner is faster than RERII [22], CHARM [23], CARPENTER [12], Max-Miner [8], and Apriori [7].

5 Experimental Results

We present our experimental results that ran on PC with Intel(R) core 2 Duo 3.20 GHz, 8.00 GB of RAM, Windows 7 64 bit system using Java compiler JDK jdk-7u3-windows-x64 and netbeans-6.7.1-ml-windows IDE. MCR-Miner algorithm is compared with the more related algorithm called MAXCONF [15] for mining maximal high confidence association rules for up-expressed genes only and also for up/down-expressed genes. The two algorithms are tested over the microarray dataset "Hughes et al 2000" of 300 samples and 6316 genes [23] and "Spellman et al. 1999" with 77 samples and 6178 genes [24]. Fig. 5 a. shows the running time in seconds of the two algorithms on the "Hughese et al" dataset; where each algorithm shows is represented with two curves. The first one (MCR-Miner1 or MAXCONF1) for mining rules for up-expressed genes only, the second (MCR-Miner2 or MAXCONF2) for mining rules for up/down-expressed genes. Fig. 5 b. shows the number of generated rules in both algorithms for up-expressed genes and for up/down-expressed genes. It is clear that, the two algorithms (MCR-Miner1 and MAXCONF1) or (MCR-Miner2 and MAXCONF2) produce the same number of the maximal high confidence association rules. Similarly, Fig. 6 a. shows the running time in seconds (Note that the y-axes of these graphs are in logarithmic (10) scale) of the two algorithms on the "Spellman et al" dataset. Fig. 6 b. shows the number of generated rules in both algorithms. It is clear that, the two algorithms produce the same number of the maximal high confidence association rules. The comparative study shows that, MCR-Miner algorithm is faster than MAXCONF algorithm.

6 Conclusion

In this paper, we have proposed and implement a new algorithm called MCR-Miner based on column (gene)-enumeration method. MCR-Miner algorithm used an efficient MAR-tree data structure with three levels only. The MAR-tree is used to efficiently save the gene with its binary representation. Using the binary representation for each gene makes the intersection processes easier and faster than the intersection processes between samples in MAXCONF algorithm. Moreover, the binary representation reduces the used memory; where all association rules are fitted in the available memory. The experimental results on the real microarray

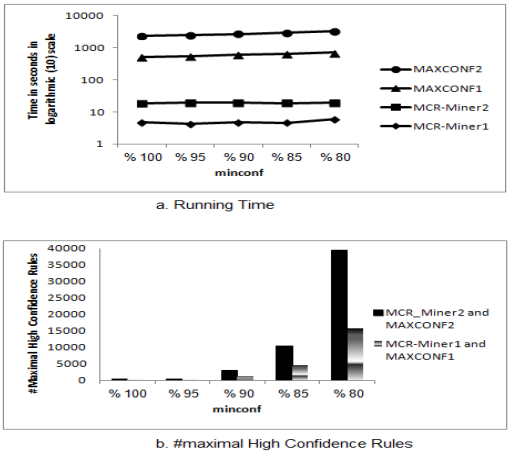

Fig. 5: MCR-Miner and MAXCONF Algorithms are Applied on Hughes Dataset (MCR-Miner1 and MAXCONF1 for up-expressed genes) and (MCR-Miner2 and MAXCONF2 for Up/Down-Expressed Genes).

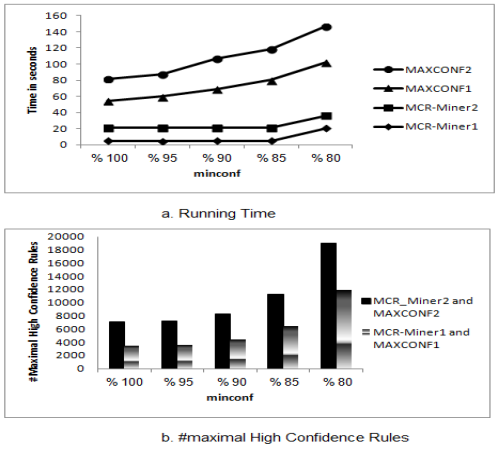

Fig. 6: MCR-Miner and MAXCONF Algorithms are Applied on Hughes Dataset (MCR-Miner1 and MAXCONF1 for up-expressed genes) and (MCR-Miner2 and MAXCONF2 for Up/Down-Expressed Genes).

datasets showed that our algorithm is more efficient and scalable than MAXCONF algorithm. Our proposed algorithm has been applied to extract the two kinds of maximal high association rules; the first one for up-expressed genes, and the second one for up/down-expressed genes.

References

[1] DOV STEKEL, Microarray Bioinformatics, Cambridge University Press, (2003).
[2] A. V. Senthil Kumar, Knowledge Discovery Practices and Emerging Applications of Data Mining: Trends and New Domains, Information Science Reference, Hershey, New York, (2011).
[3] M. Wang, X.Q.Shang, and Z.H.Li, Strong association rules mining without using frequent items for microarray analysis, The 3rd Int. Conf. on Bioinformatics and Biomedical Engineering, (iCBBE 2009).
[4] Alves R, Rodriguez-Baena DS, and Aguilar-Ruiz JS, Gene association analysis: a survey of frequent pattern mining from gene expression data, Brief Bioinform, (2010).
[5] J. Han and M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers, Elsevier, (2002).
[6] R. Agrawal and R. Srikant, Fast Algorithms for Mining Association Rules, Proceedings of the 20th Int. Conf. on Very Large Data Bases (VLDB94),475486, Santiago de Chile, Chile, (1994).
[7] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Inkeri Verkamo, Fast Discovery of Association Rules, Advances in Knowledge Discovery and Data Mining, Menlo Park, 307-328 (1996).
[8] R. J. Bayardo, Efficiently Mining Long Patterns From Databases, ACM SIGMOD Conf. Management of Data, June (1998).
[9] Zaki M. J., Parthasarath S., Ogihara M., and Li W., New Algorithms for Fast Discovery of Association Rules. Proc. of the Third Int. Conf. on Knowledge Discovery in Databases and Data Mining, 283-286 (1997).
[10] Wael Z., Yasser K., Elham E., and Fayed G., Mining Association Rule with Reducing Candidates Generation. Proc. of Fourth Int. Conf. on Intelligent Computing and Information Systems (ICICIS2009). ACM, (2009).
[11] M. J. Zaki, Scalable Algorithms for Association Mining, Transactions on Knowledge and Data Engineering, **12**, 372-390 (2000).
[12] F. Pan, G. Cong, A. K. H. Tung, J. Yang, and M. J. Zaki, CARPENTER: Finding closed patterns in long biological datasets, Proc. ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining(KDD), (2003).
[13] G. Piatetsky-Shapiro, P. Tamayo, Microarray data mining: facing the challenges, SIGKDD Explor. Newsl, **5**, 1-5 (2003).
[14] C. Becquet, S. Blachon, B. Jeudy, J.-F. Boulicaut, and O. Gandrillon, Strong association rule mining for large gene expression data analysis: a case study on human SAGE data, Genome Biology, **12**, (2002).
[15] T. McIntosh and S. Chawla: High confidence rule mining for microarray analysis. In: IEEE/ACM TCBB, **4**, 611-623 (2007).
[16] G. Cong, K.-L. Tan, A. Tung, and F. Pan: Mining Frequent Closed Patterns in Microarray Data. In: Proc. Fourth IEEE Intl Conf. Data Mining (ICDM), **4**, 363-366 (2004).
[17] M. J. Zaki and C. Hsiao: CHARM: An efficient algorithm for closed association rule mining. In: Proc. SIAM Intl Conf. on Data Mining (SDM), (2002).
[18] Agrawal, R., Imielinski, T., and Swami, A. N: Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, 207-216.
[19] F. Pan, G. Cong, K. Tung, Yang J., and M. J. Zaki: Carpenter, Finding closed patterns in long biological datasets, Proc.ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD), 637-642 (2004).

[20] G. Cong, X. Xu, and F. Pan, A. Tung, J. Yang, FARMER: Finding Interesting Rule Groups in Microarray Datasets, SIGMOD, (2004).
[21] J. Pei, J. Han, and R. Mao, CLOSET: An efficient algorithm for mining frequent closed itemsets, In Proc. ACMSIGMOD Int. Workshop Data Mining and Knowledge Discovery (DMKD), (2000).
[22] J. Agrwal and J. C. Ramesh: Analysis of Gene Microarray Data using Association Rule Mining, Journal of computing, **4**, (2012).
[23] T. Hughes et al, Functional Discovery via a Compendium of Expression Profiles, Cell, **102**, 109-126 (2000).
[24] P. Spellman, G. Sherlock, M. Zhang, V. Iyer, K. Anders, M. Eisen, P. Brown, D. Botstein, and B. Futcher: Comprehensive Identification of Cell Cycle-Regulated Genes of the Yeast Saccharomyces Cerevisiae by Microarray Hybridization. In: Molecular Biology of the Cell, **9**, 3273-3297 (1998).

Wael Zakaria is assistant lecturer of Computer Science at Department of Mathematics, Faculty of Science, Ain Shams University, Cairo, Egypt. He received the B.Sc and M.Sc degree; the thesis titled with ”On Data mining and web mining” , he start works in PhD in 2010 at Department of Mathematics, Computer Science Division, Faculty of science, Ain Shams University. His research interests are in the areas of Data mining in Bioinformatics and DNA microarray.

Yasser Kotb is an Assistant Professor with College of Computer and Information Sciences, Al-Imam Muhammad ibn Saud Islamic University, KSA (On leave from Faculty of science, Ain Shams University, Egypt). He received the PhD degree from School of Information Science, Japan Advanced institute of Science and Technology (JAIST), Japan (2003). He got the distinguish JSPS postgraduate scholarship for two years at JAIST (2004-2006). He received his BS and M.Sc. degrees in Computer Science from Ain Shams University, Egypt (1991 and 1997, respectively). During this period, He has been working as assistant lecturer in the department of Mathematics/Computer science, Faculty of Science, Ain Shams University. Since 1997, he has been working as an assistant professor in the same department too. His research areas include Bioinformatics, Data and Web Mining, Web Technologies, Web Services, P2P and Formal Methods. Dr. Kotb has served as an organized and on the program committees of numerous international conferences and workshops, and published many papers in refereed journals and conference proceedings. He has taught a wide range of computer and information sciences courses for postgraduate and graduate levels.

Fayed F. M. Ghaleb is Professor Emeritus in Computer Science Division, Department of Mathematics, Faculty of Science, Ain Shams University, Cairo, Egypt. He received his B.Sc. (special) degree in Mathematics from Ain Shams University in1966. Dr F. Ghaleb received the Ph.D. degree in Applied Functional Analysis from Moscow State University, Moscow, Russia in1978. He participated in the establishment of Computer Science Division at the Math. Dept. in Faculty of Science, Ain Shams University in 1983, and he has been supporting the division with all efforts till now (Vice Chair for 6 years). Professor F. Ghaleb also participated in establishing the Egyptian Mathematical Society (ETMS) in 1992 and shared in organizing all its national, international and one-day conferences (more than 170 till now). He is the treasurer of ETMS from its beginning till now. He participated and attended conferences and workshops in Italy, France, and USA. He worked as a full time and part time teaching Professor in many of the Egyptian and Arabic Universities. His current fields of interests include image processing, e-learning, data and web mining, and bioinformatics. He supervised more than 37 M.Sc. and Ph.D. students in Mathematics and Computer Science. He published more than 47 papers in different fields. Recently, Professor F. Ghaleb is the president of Ain Shams University Staff Association.