MINING LOCALIZED CO-EXPRESSED GENE PATTERNS FROM MICROARRAY DATA

By

Ji Liping

(Bachelor of Management, Nanjing University, China)

A DISSERTATION SUBMITTED
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
AT
NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING
JUNE 2006

Table of Contents

Acknowledgements

I would like to express my heartfelt gratitude to my supervisor, Prof. Tan Kian-Lee. Being a novice in the field of research, I feel very much privileged to have worked under him, for his expertise and teachings has taught me invaluable lessons and given me a deeper insight into the world of research. His industrious attitude with the attention to the slightest of details towards research work has greatly inspired me. I am really grateful too for the enduring patience and support that was shown by him to me whenever I encountered difficult obstacles in the course of my research work. His technical and editorial advice contributed a major part to the successful completion of this dissertation. It would have been a much more uphill task without him as my mentor. Lastly, the experience of working as a graduate research student under Prof. Tan has been extremely rewarding. I wish to express thanks for his invaluable advice and encouragement throughout the course of my graduate studies in School Of Computing.

My thanks also go to members of my thesis committee Dr. Anthony K H. Tung and Dr. Sung Wing Kin, who provided valuable feedback and suggestions to my research questions.

Also, I would also like to acknowledge past and current database group members Dr. Cong Gao, Kenneth Mock, Wang Shufan, Dong Xiaoan, Tang Jiajun, Zhou Yongluan, Xu Xin, and Zhang Zonghong. It has really been a great and fulfilling experience working together with them.

I am also very grateful to my undergraduate mentor Yang Jianning, and my friends Wang Guanqun, Baijing, Cao Dongni, Li Yuan, Wang Liping who provided

tremendous mental support to me when I got frustrated at times.

Last, I would like to express my deepest gratitude and love to my parents for their support, encouragement, understanding and love during the many years of my studies.

Life is a journey. It is with all the care and support from my loved ones that has allowed me to scale on to greater heights.

List of Figures

List of Tables

Abstract

With the new advances in DNA microarray technology, expression levels of thousands of genes can be simultaneously measured efficiently during important biological process and across collections of related samples. Analyzing the microarray data to identify localized co-expressed gene patterns are essential in revealing the gene functions, gene regulations, subtypes of cells, and cellular processes of gene regulation networks. Hence, researchers are recently motivated to mine co-expressed gene patterns from microarray data.

This thesis studies both the static and dynamic aspects of localized co-expressed gene patterns and categories the patterns into three types: co-attribute patterns, co-tendency patterns and time-lagged patterns. Designing new algorithms to identify the three types of localized co-expressed gene patterns is the research problem of this thesis.

We present in this thesis a series of new algorithms to mine localized co-expressed gene patterns. First, we extend the 2D frequent closed patterns (FCPs) mining algorithms from sparse data context to dense context, and propose two new algorithms *B-Miner* and *C-Miner* to mine 2D co-attribute patterns (FCPs). We also study the parallel schemes of the two algorithms, which is, to our knowledge, the first parallel frequent closed pattern mining schemes in the literature. Second, we extend the traditional 2D FCPs mining algorithms to the 3D context. We introduce the notion of frequent closed cube (FCC) and formally define it. Based on this notion, we mine 3D co-attribute patterns (FCCs), which settles the new challenges coming up with the spurning of 3D microarray data. We propose two novel algorithms *Representative Slice Mining (RSM)* and *CubeMiner* to mine FCCs from 3D datasets. We also show how *RSM* and *CubeMiner* can be easily extended to exploit parallelism. Third, we propose a *quick hierarchical biclustering* algorithm *(QHB)* to mine co-tendency patterns (biclusters) from 2D microarray data efficiently. *QHB* ensures that the final bicluster trends are not only consistent but exhibit similar degrees of fluctuation between consecutive conditions. Moreover, *QHB* provides a hierarchical picture of inter-bicluster relationships, maintains information integrity and offers users a progressive way of knowledge exploration. Finally, we propose an efficient algorithm *q-cluster* to identify time-lagged patterns. The algorithm facilitates localized comparison and processes several genes simultaneously to generate detailed and complete time-lagged information between genes/gene clusters.

We conduct experiments on both synthetic and real microarray datasets. Our experiments show the effectiveness and efficiency of our algorithms in mining the localized co-expressed gene patterns. We believe our research in this thesis delivers valuable information and provides excellent tools for bioinformatics research.

Chapter 1

Introduction

1.1 Motivation: Microarray Technology and Microarray Data Analysis

1.1.1 Microarray Technology

DNA microarray technologies are one of the latest breakthroughs in recent experimental molecular biology, which provide a powerful tool for researchers to quickly, efficiently and accurately measure the expression levels of thousands of genes simultaneously during important biological process and across collections of related samples. The cDNA microarray [47] and oligonucleotide arrays [16] are two main types of microarray experiments. The whole microarray process, as shown in Figure 1.1, contains three basic procedures [55, 1]:

Chip Manufacture: A microarray is a small chip where thousands of DNA molecules (probes) are attached in fixed grids. Each grid cell relates to a DNA sequence.

Target Preparation, Labelling and Hybridization: A target sample and a reference sample are labelled with red and green dyes, respectively, and each is hybridized with the probes on the surface of the chip.

Scanning Process: Chips are scanned by the fluorescent microscope, and with

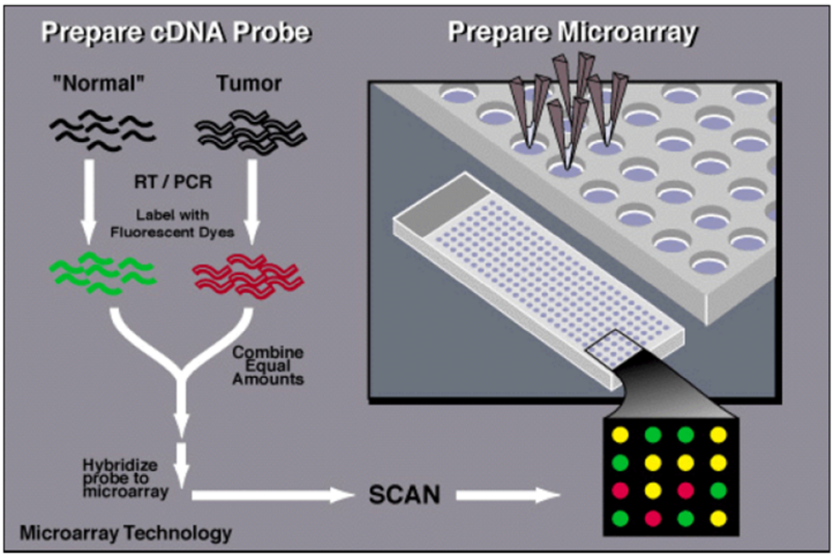


Figure 1.1: Microarray Process

image analysis, the log(green/red) signal intensities of mRNA hybridizing at each site is measured.

Both cDNA microarray and oligonucleotide array experiments measure the expression level for each DNA sequence by the ratio of signal intensity between the experimental sample and the reference sample. Positive values indicate higher expression in the target versus the reference, and vice versa for negative values. Therefore, datasets resulting from both methods share the same biological semantics. In this thesis, we will refer to both the cDNA microarray and the oligonucleotide array as microarray technology and term the measurements collected via both methods as gene expression data.

A microarray experiment typically assesses a large number of DNA sequences (genes, cDNA clones, or expressed sequence tags) under multiple experimental conditions. These experimental conditions may be cellular environments, or a collection of

$$\begin{array}{cc|cccc|}
 & & \multicolumn{4}{c}{\text{Experimental Condition } c_j} \\
\text{Matrix O} & & c_1 & c_2 & \cdots & c_m \\ \cline{3-6}
 & g_1 & O_{11} & O_{12} & \cdots & O_{1m} \\
 & g_2 & O_{21} & O_{22} & \cdots & O_{2m} \\
\text{Gene } g_i & \vdots & \vdots & & & \vdots \\
 & g_n & O_{n1} & O_{n2} & \cdots & O_{nm} \\ \cline{3-6}
\end{array}$$

Figure 1.2: Gene Expression Matrix

different tissue samples (e.g., normal versus cancerous tissues), or a time series during a biological process (e.g., the yeast cell cycle). In this thesis, we will uniformly term the "DNA sequence" as "gene" and refer to all kinds of "cellular environments", "tissue samples", and "time series" as "experimental conditions". The gene expression dataset resulting from a microarray experiment where the expression levels of genes are measured under single category of experimental conditions can be represented by a real-valued gene expression matrix $O = \{O_{ij} | 0 \leq i \leq n, 0 \leq j \leq m\}$, where the rows $G = \{g_1, g_2, \ldots, g_n\}$ form the expression patterns of genes, the columns $C = \{c_1, c_2, \ldots, c_m\}$ represent the expression profiles of experimental conditions, and each cell O_{ij} is the measured expression level of gene i under experimental condition j. Figure 1.2 illustrates such a matrix.

Furthermore, the gene expression dataset resulting from a microarray experiment where the expression levels of genes are measured under multiple categories of experimental conditions can be represented by a real-valued gene expression cube $O = \{O_{ij\ldots k} | 0 \leq i \leq n, 0 \leq j \leq m, \ldots, 0 \leq k \leq l\}$, where one dimension of the cube $G = \{g_1, g_2, \ldots, g_n\}$ forms the expression patterns of genes, the other dimensions

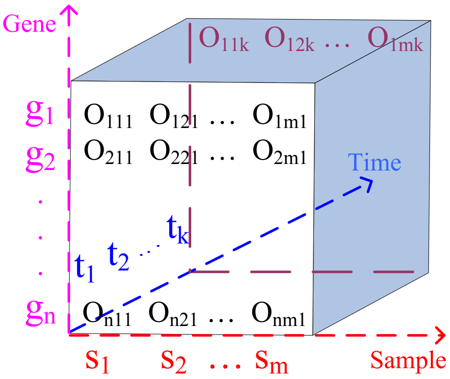


Figure 1.3: Gene Expression Cube

$C_j = \{c_{j1}, c_{j2}, \ldots, c_{jm}\}, \ldots, C_k = \{c_{k1}, c_{k2}, \ldots, c_{kl}\}$ represent the expression profiles of other experimental conditions respectively, and each cell $O_{ij\ldots k}$ is the measured expression level of gene i under several experimental conditions from j to k simultaneously. Figure 1.3 illustrates an example of the 3D *gene-sample-time* data cube where the expression levels of n genes are measured simultaneously under m tissue samples over a series of k time points.

1.1.2 Microarray Data Analysis

The gene expression data produced by the DNA microarray technologies are known as microarray data. Analysis on the huge amount of valuable microarray data has become one of the major bottlenecks in the utilization of the microarray technologies. As various researches on mapping and sequencing genomes are reaching successful completion, the researchers are recently focusing more on functional genomics. Initial experiments suggest that genes of similar functions yield similar expression patterns in microarray hybridization experiments [1]. The genes with similar expression patterns are called co-expressed genes, while the similar gene patterns are called co-expressed gene patterns. Co-expressed gene patterns are essential in revealing the gene functions, gene regulations, subtypes of cells, and cellular processes of gene regulatory networks.

- First, co-expressed genes may demonstrate a significant enrichment for function analysis of the genes. The functions of some poorly characterized or novel genes may be better understood by testing them together with the genes with known functions.

- Second, co-expressed genes with strong expression pattern correlations may indicate co-regulation and help uncover the regulatory elements and the mechanism of the transcriptional regulatory networks.

- Third, elucidating different co-expressed gene patterns may help reveal sub-cell types which are hard to identify by traditional morphology-based approaches [32].

- Finally, in the co-expressed gene patterns, genes are related to specific experimental conditions (cellular environments/samples/time periods) and the related experimental conditions are grouped together as well. This helps to elucidate the underlying knowledge in the co-effects of experimental conditions on the co-expressed genes.

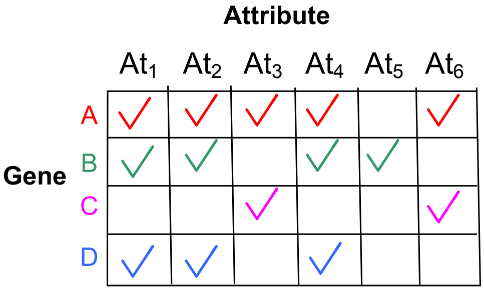
Hence, identifying the co-expressed gene patterns hidden in microarray data offers a great opportunity for an enhanced understanding of functional genomics. Biological studies show that many co-expressed patterns are common to a group of genes only under specific experimental conditions. In cellular processes, subsets of genes are usually co-expressed only under certain experimental conditions, but behave almost independently under other conditions. Hence, identifying co-expressed gene patterns under the whole experimental conditions may not be useful to practical biological application. On the contrary, discovering *localized* co-expressed gene patterns is the key to uncovering many genetic pathways that are not apparent otherwise. Therefore, researchers are motivated to extract a subset of genes that co-express under a subset of experimental conditions.

1.2 Research Problem: Mining Localized Co-expressed Gene Patterns

Data mining, which is a process of analyzing data in a supervised/unsupervised manner to discover useful and interesting information hidden within the data, has become one of the main techniques in the microarray data analysis. In this thesis, our research problem is to mine localized co-expressed gene patterns from microarray data. In the following, we give the definition of localized co-expressed gene patterns, categorize them into three types, and detail each type respectively.

Definition 1.1: Localized Co-expressed Gene Patterns A localized co-expressed gene pattern is made up of a subset of genes and a subset of experimental conditions (biological attributes, samples, time series and etc.) such that the subset of genes either (a) share the same subset of biological attributes; or (b) have the same expressing status under the same subset of experimental conditions; or (c) have the similar changing tendency when experimental conditions change consecutively; or (d) have the similar changing tendency after a certain time lag.

Based on the way how genes co-regulate, we categorize the localized co-expressed



Attribute

Gene	At_1	At_2	At_3	At_4	At_5	At_6
A	✓	✓	✓	✓		✓
B	✓	✓		✓	✓	
C			✓			✓
D	✓	✓		✓		

Figure 1.4: Example: Co-attribute Pattern

gene patterns into three types: co-attribute patterns, co-tendency patterns, and time-lagged patterns.

1.2.1 Co-attribute Pattern

The co-attribute pattern emphasizes the *static* co-regulations among genes. It contains genes that either share the same biological attributes (case(a)), or have the same expressing status (expressed/depressed) under specific experimental conditions (cellular environments/samples/time periods) (case(b)). Given the table in Figure 1.4 for example, let the rows represent genes A, B, C, D; let the columns represent six attributes from At_1 to At_6; and let cells containing "$\surd$" indicate that the relative genes have certain attributes, then genes A, B, D and attributes At_1, At_2, At_4 form a co-attribute pattern. That is, the genes A, B, D share the same attributes of At_1, At_2, At_4, which makes them a co-attribute pattern. Since any subset of A, B, D and At_1, At_2, At_4 can also form co-attribute patterns but contains no new information, in this thesis, we only focus on the "maximal" patterns. The co-attribute pattern is "maximal" if it contains the maximal subsets of biological attributes or experimental conditions that frequently occur in maximal subsets of genes.

Frequent closed pattern (FCP) mining technique [41] has been widely applied to mine the “maximal” co-attribute patterns. The resulting FCPs are the “maximal” co-attribute patterns[1]. Several efficient FCP mining algorithms have been proposed in the literature. Some notable schemes include CLOSET [42], CLOSET+ [22], CHARM [60], CARPENTER [39], REPT [12] and D-miner [7]. While these FCP mining algorithms have been shown to perform well in their respective context, it turns out that they have limitations in three aspects: (a) they are not particularly effective for dense biological datasets; (b) they are all limited to 2D dataset analysis; (c) there are no *parallel* closed frequent pattern mining algorithms in the literature. These limitations motivate us to design novel methods to mine FCPs from *dense* datasets effectively, extend existing 2D frequent closed pattern analysis to 3D context, and parallelize the FCP mining process as well.

1.2.2 Co-tendency Pattern

The co-tendency pattern emphasizes the *dynamic* co-regulations among genes. It contains genes that have the similar changing tendency when experimental conditions change consecutively (case(c)). That is, the subset of genes’ expression levels rise and fall coherently under a subset of consecutive experimental conditions. Figure 1.5 shows an example of co-tendency pattern[2]. With the change of time, the expression levels of genes YBR101C and YFL006W have the similar changing tendency, and they exhibit a fluctuation of the similar shape.

Biclustering technique [11] has been well studied in the literature to mine co-tendency patterns. Biclustering simultaneously clusters both genes and experimental

[1] In the thesis, “FCPs” is termed as the counterpart of “ maximal co-attribute patterns”.

[2] data downloaded from http://arep.med.harvard.edu/biclustering/yeast.matrix

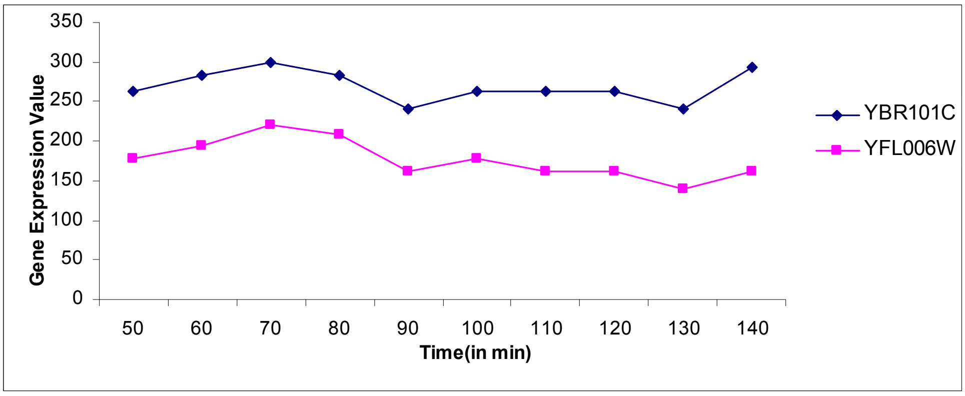


Figure 1.5: Example: Co-tendency Pattern

conditions, which captures the coherence of a subset of genes under a subset of experimental conditions. The resulting biclusters are co-tendency patterns[3]. Some notable biclustering algorithms include bicluster model [11], δ-cluster model [58], pClusters [56], and DBF [63]. While these algorithms can generate co-tendency patterns, they are limited in several ways: (a) they are not adequate to capture the trend consistency of biclusters; (b) they miss out some interesting patterns; (c) they are inefficient due to the hill-climbing paradigm; (d) they cannot provide a graphical representation of the inter-bicluster relationships. To address these limitations, in this thesis, we design an effective and efficient biclustering algorithm that could deliver the inter-bicluster relationships favored by the biologists.

1.2.3 Time-Lagged Pattern

The time-lagged pattern emphasizes the *delayed dynamic* co-regulations among genes. It contains genes that have the similar changing tendency after a certain time lag (case(d)). That is, some genes' expression levels exhibit a fluctuation of the *delayed*

[3] In the thesis, "biclusters" is termed as the counterpart of "co-tendency patterns".

similar shape to the other genes'. Figure 1.6 shows an example of time-lagged pattern[4]. With the change of time, the expression levels of gene YDR224C have a similar but *delayed* changing tendency with gene YGL207W, and they exhibit a fluctuation of the *delayed* similar shape. From the time-lagged pattern, we could infer that the expression of gene YGL207W may have an "activation" effect on the expression of gene YDR224C.

While the FCP mining and biclustering techniques are employed to mine co-attribute patterns and co-tendency patterns respectively, they cannot identify patterns with time-lagged gene co-regulations. Existing work on time-lagged analysis largely analyzes two genes at a time over all conditions and ranks the gene pairs based on the score generated using a certain criterion, such as the Cross-Correlation Function [33] and the Needleman-Wunsch alignment algorithm [34]. The gene pairs with higher scores are regarded as the interesting and promising pairs. Such an approach is clearly computationally inefficient: given n genes, we would need $\binom{n}{2}$ comparisons. More importantly, these techniques may miss out some interesting time-lagged patterns. Since the score is generated based on the analysis of the whole sequence, it is not sensitive to the cases that a small but interesting part of the genes are co-regulated while there is no distinct relationship between the remaining part. As a result, some interesting gene pairs may not always be ranked higher than uninteresting ones. A higher scoring threshold will lose out some interesting patterns while a lower one will bring about tremendous amount of redundant pairs. In addition, there is a lack of detailed information on co-regulated gene pairs, such as the exact lagged-time, the

[4] data downloaded from http://genome-www.stanford.edu/cellcycle/data/rawdata

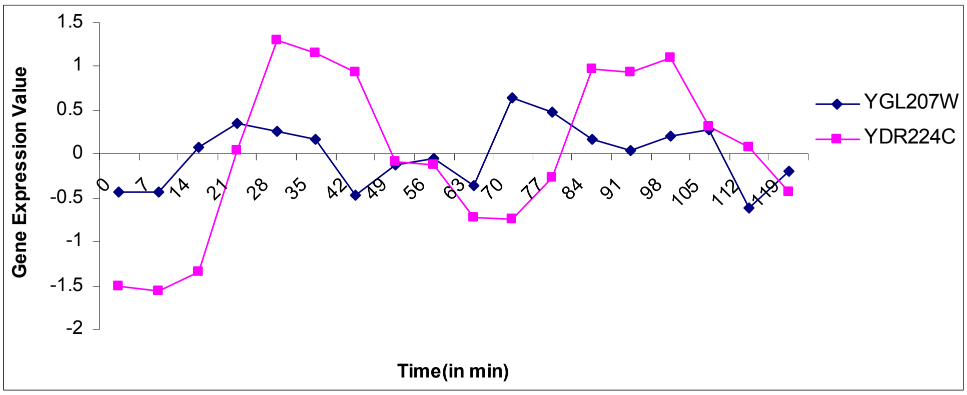


Figure 1.6: Example: Time-Lagged Pattern

starting and ending time points, and the number of the co-regulated patterns between two genes. Moreover, they mostly deliver co-regulations between genes, but seldom draw relationships between gene clusters. As such, we would like to explore new time-lagged clustering algorithm to identify localized time-lagged co-regulations between genes and/or gene clusters efficiently.

1.3 The Contributions

To solve the research problems discussed, we propose several new algorithms in this thesis to mine the three types of localized co-expressed gene patterns from microarray data.

1.3.1 2D FCP from Dense Datasets: *C-Miner* and *B-Miner*

We extend the 2D frequent closed pattern (FCP) mining algorithms from sparse data context to dense context. We introduce a framework that progressively returns FCPs to users. The framework has the following three distinguishing features.

First, the original mining space is recursively partitioned into sub-spaces such that (a) each subspace can be mined independently, and (b) the union of the FCPs obtained from all subspaces is a superset of the answer.

Second, as each subspace is mined independently, redundant FCPs (those that may also be produced in other subspaces) and false drops (those that are FCPs in the subspace but are not FCPs in the original space) are pruned away.

Third, because the subspaces can be mined independently, answers can be progressively returned to users as each subspace is mined. Moreover, the framework facilitates parallel mining efficiently without incurring significant communication overhead. Based on the framework, we propose two schemes: *C-Miner* and *B-Miner*. We have implemented *C-Miner* and *B-Miner*, and our performance study on synthetic datasets and real dense datasets shows their effectiveness over existing schemes. We also report experimental results on parallel versions of these two methods.

1.3.2 3D FCP: *RSM* and *CubeMiner*

We extend the traditional 2D FCP mining algorithms to the 3D context to deal with the new challenges coming up with the spurning of 3D microarray data. Our contributions are as follows.

First, we introduce the concept of frequent closed cube (FCC), which generalizes the notion of 2D frequent closed pattern to 3D context.

Second, we propose two approaches to mine FCCs from 3D dataset. The first approach is a three-phase framework, called *Representative Slice Mining* algorithm *(RSM)* that exploits 2D FCP mining algorithms to mine FCCs. The basic idea is to transform a 3D dataset into a set of 2D datasets, mine the 2D datasets using an existing 2D FCP mining algorithm, and then prune away any frequent cubes that are not closed. The second method is a novel scheme, called *CubeMiner*, that operates directly on the 3D dataset to mine FCCs.

Third, we also show how *RSM* and *CubeMiner* can be easily extended to exploit parallelism.

Finally, we have implemented *RSM* and *CubeMiner*, and conducted experiments on both real and synthetic datasets. The experimental results show that the *RSM*-based scheme is efficient when one of the dimensions is small, while *CubeMiner* is superior otherwise. To our knowledge, there has been no prior work that mine FCCs.

1.3.3 Bicluster: *Quick Hierarchical Biclustering*

To overcome the limitations of traditional biclustering algorithms, we propose a *quick hierarchical biclustering* algorithm *(QHB)* to efficiently mine biclusters with both consistent trends and trends with similar degrees of fluctuations. Compared with previous biclustering models, we have made five main contributions.

First, we define a new bicluster quality measurement called *Mean Fluctuating Degree* (MFD) to reflect the trend consistency of biclusters. Since a similarity score is not enough to ensure trend consistency, we use our MFD only as a supplementary control agent. Instead, the trend consistency is mainly controlled and embedded in the partitioning strategy of *QHB*, which ensures the high quality of consistent trends within each bicluster.

Second, instead of improving on only part of the "seeds", *QHB* takes the entire dataset into consideration. During the hierarchical partitioning process, all valuable information of a parent node is kept into the child nodes without any loss.

Third, *QHB* adopts a partition based refinement that can simultaneously process several rows/columns. This is much more efficient than existing techniques.

Fourth, *QHB* provides a very clear hierarchical inter-bicluster relationships. Such graphical representation of the relationships among biclusters provides more valuable knowledge to the biologists.

Finally, the hierarchical partitioning strategy of *QHB* facilitates a progressive refinement of results. Biclusters are refined from generality to details progressively. This is very helpful in biological application. Instead of waiting long hours for all detailed results, biologists now would be provided with a general picture of the whole results from the upper levels of the hierarchical tree in a very short response time. Then biologists could freely choose their focus, rolling up to generalize it or rolling down to detail it, progressively. This would help biologists quickly focus on their most interested patterns for further exploration.

1.3.4 Time-Lagged Pattern: *q-cluster*

To overcome the limitation of existing time-lagged gene co-regulation analysis algorithms, we propose an efficient algorithm *q-cluster* to identify time-lagged co-regulated gene clusters. The algorithm facilitates localized comparison and processes several genes simultaneously to generate detailed and complete time-lagged information between genes/genes clusters. Compared with previous works, we have made three main contributions.

First, *q-cluster* takes localized co-regulation into consideration, which is more detailed and valuable than traditional global analysis. In addition, it delivers a more detailed information on co-regulated gene patterns, such as the exact lag time, the starting and ending time points and the number of co-regulated patterns between genes.

Second, *q-cluster* processes several genes simultaneously, which is much more efficient than previous algorithms that analyze only two genes each time.

Third, *q-cluster* not only delivers time-lagged co-regulations between genes (as traditional global methods), but also delivers time-lagged co-regulations between gene clusters.

1.4 Thesis Outline

The remainder of this thesis is organized as follows. In Chapter 2, we will review the previous mining techniques for localized co-expressed gene pattern identification. Chapter 3 presents the two new algorithms *C-Miner*, *B-Miner* and their parallel versions for efficient mining of frequent closed patterns (2D co-attribute gene patterns) in 2D dense context. Chapter 4 proposes the notion of *frequent closed cube* and introduces two novel algorithms *RSM*, *CubeMiner* and their parallel versions for frequent closed cubes (3D co-attribute gene patterns) mining in 3D context. In Chapter 5, we propose a quick hierarchical biclustering algorithm *QHB* for efficient biclusters (co-tendency gene patterns) mining. In Chapter 6, we propose a new efficient algorithm *q-cluster* to identify time-lagged co-regulated gene clusters (time-lagged gene patterns). Finally, chapter 7 concludes this thesis and discusses some future research work.

In Chapter 3, The 2D frequent closed pattern mining algorithms from dense datasets take the material from paper [26]; the 3D frequent closed cube mining algorithms in Chapter 4 adopt some material from paper [27]; Chapter 5 uses the algorithm in paper [23] to mine biclusters from 2D datasets; and the *q-cluster* algorithm for mining 2D time-lagged patterns take some material appearing in papers [24, 25].

Chapter 2

Literature Reviews

In this chapter, we will review some existing mining techniques for co-attribute patterns, co-tendency patterns and time-lagged patterns respectively. We also review the data preprocessing techniques which can improve the quality of the data, thereby helping to improve the accuracy and efficiency of the subsequent mining process.

2.1 Co-attribute Patterns: Frequent Closed Pattern Mining

Frequent pattern mining is an unsupervised mining technique that identifies all subsets of items or attributes frequently occurring in many database records or transactions. Frequent pattern mining is a fundamental step to several essential data mining tasks, including association rule analysis [3], sequential patterns [4], episodes [37], partial periodicity [20], and etc. As such, many efficient frequent pattern mining algorithms have been proposed in the literature [3, 36, 50, 61]. However, frequent pattern (FP) mining is a time-consuming process to generate too many patterns (a large number of which are "redundant" in the sense that they do not shed additional insights) for users to digest. To reduce the number of frequent patterns, frequent closed pattern (FCP) mining [41] was proposed to identify all maximal subsets of items or attributes that frequently occur in maximal subsets of database records or transactions. While the number of FCPs are much smaller than the FPs, FCPs carry the same information as the FPs.

Several efficient FCP mining algorithms have been proposed in the literature. A-close [41] uses a breadth-first search to find FCPs. CLOSET [42] and CLOSET+ [22] adopt a depth-first, feature enumeration strategy. CLOSET uses a frequent pattern tree for a compressed representation of the dataset. CLOSET+, an enhanced version of CLOSET, uses a hybrid tree-projection method to build conditional projected table in two different ways according to the density of the dataset. Both MAFIA [9] and CHARM [60] use a vertical representation of the datasets. MAFIA adopts a compressed vertical bitmap structure while CHARM enumerates closed itemsets using a dual *itemset-tidset* search tree and adopts the *Diffset* technique to reduce the size of the intermediate *tidsets*. Since these methods adopt a feature enumeration strategy, they cannot efficiently handle datasets with a large number of features (columns) and a small number of rows (which are common in microarray datasets).

A recently proposed FCP mining algorithm, CARPENTER [39], is designed to deal with the special “large columns small rows” characteristic of biological datasets. CARPENTER combines the depth-first, row enumeration strategy with some efficient search pruning techniques, which results in a scheme that outperforms traditional closed pattern mining algorithms on biological data. Another algorithm, COBBLER [40], has also been proposed to mine biological datasets. COBBLER is designed to dynamically switch between feature enumeration and row enumeration depending on the data characteristic in the process of mining. However, the decision to switch the enumeration strategies at runtime is not very precise and is costly. Yet another algorithm is REPT [12]. REPT traverses the row enumeration tree using a projected transposed table. The projected transposed table is represented by a prefix tree, which is similar to the FP-tree [42]. However, unlike the FP-tree whose nodes represent items, nodes in a prefix tree are rows. Experimental results showed that REPT is more efficient than CLOSET+ and CARPENTER [12]. Unfortunately, all these three algorithms do not work well when the dataset is dense.

In [7], a novel algorithm, D-miner, was proposed to identify closed sets of attributes (or items) for dense and highly-correlated boolean contexts. As we will explore D-Miner in this thesis, we describe the algorithm of D-Miner in details here. D-miner mines FCPs (T, G) from data matrix A under constraints. It builds the sets T and G and uses monotonic support threshold constraints simultaneously on the object set O and item set P to reduce the search space. D-Miner uses H to denote a set of cell groups which are partitions of the false values (i.e., "0") of the boolean matrix. An element $(a, b) \in H$ is called a "cutter" if $\forall t \in a$, and $\forall g \in b$, $A_{t,g} = 0$. H contains as many elements as rows in the matrix. Each element is composed of the attributes valued by 0 in this line. Given the matrix A in Table 2.1 for example, the cutter set H contains three elements: $(t_1, g_1 g_2)$, (t_2, g_2), and $(t_3, g_1 g_2)$.

Table 2.1: An Example Dataset (Matrix A).

O/P	g_1	g_2	g_3
t_1	0	0	1
t_2	1	0	1
t_3	0	0	1

D-Miner starts with the whole dataset $A(O, P)$ and then splits it recursively using the cutters of H until all cutters in H are used and consequently all cells in each resulting submatrix have the value 1. A cutter $(a, b) \in H$ is used to cut a submatrix (X, Y) if $a \cap X \neq \emptyset$ and $b \cap Y \neq \emptyset$. When a submatrix (X, Y) is split by a cutter $(a, b) \in H$, then $(X \backslash a, Y)$ (the left son) and $(X, Y \backslash b)$ (the right son) are generated. Recursive splitting leads to all FCPs, but also some non-maximal unclosed frequent patterns. Figure 2.1 shows the splitting tree generated from the 2D matrix A in Table 2.1. From Figure 2.1, we can see that the resulting submatrix (t_3, g_3) and (t_2t_3, g_3) are non-maximal unclosed frequent patterns as they have a superset $(t_1t_2t_3, g_3)$.

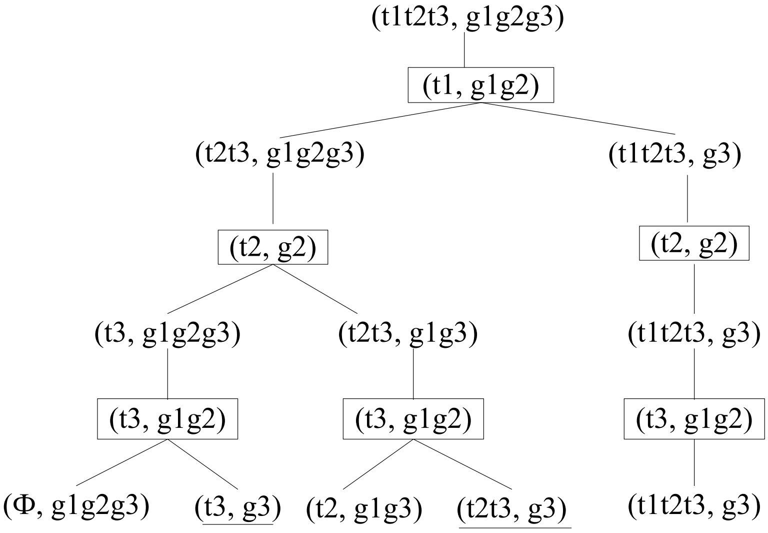


Figure 2.1: D-Miner Splitting Tree.

To remove the unclosed patterns from the results, D-Miner employs a close checking property as follows:

Property 2.1: Let (X, Y) be a leaf of the tree and $H_L(X, Y)$ be the set of cutters associated to the left branches of the path from the root to (X, Y). Then (X, Y) is a FCP if it contains at least one item of each element of $H_L(X, Y)$. It means that when trying to build a right son (X, Y), we must check that $\forall (a, b) \in H_L(X, Y)$, $b \cap Y \neq \emptyset$.

According to **Property 2.1**, (t_3, g_3) and (t_2t_3, g_3) in Figure 2.1 are pruned off in that they contain neither g_1 nor g_2.

D-miner's effectiveness comes from the fact that it focuses on the missing items/ attributes of an attribute/item, which are actually the sparse "0" portion of the dataset. However, the efficiency of D-miner highly depends on the number of cutters, which is relevant to the minimum number of the dataset's rows/columns containing "0". As a result, when the dataset has relatively large number of rows and columns, D-miner loses its advantages.

Although the above algorithms may have good applications in their specific domains, it turns out that they have limitations in three aspects.

First, they are not suited for applications that involve datasets with very high density where nearly 50% or more of the cells contain ones (as we shall see, all the real microarray datasets that we used in the performance study are dense) - they are either very inefficient (i.e., take hours or even days to produce patterns even with high minimum support threshold), or may even fail (i.e., run out of memory). In addition, these methods are non-progressive, i.e., the users are swarmed with all the answer patterns (after a very long wait) at a single time when the algorithm completes. These limitations motivate us to mine FCPs from *dense* datasets efficiently and *progressively.*

Second, they are all limited to 2D dataset analysis, for example, the *gene-time*, *gene-sample* biological datasets in microarray dataset analysis. With recent advances in microarray technology, the expression levels of a set of genes under a set of samples can be measured simultaneously over a series of time points, which results in 3D *gene-sample-time* microarray data [32]. This trend motivates us to extend existing 2D frequent closed pattern analysis to 3D context. In [46], a scheme is proposed to discover calendric association rules. Although time intervals are taken as a third dimension, they are pre-defined by users as calendric information. Hence, no thorough enumeration on the third dimension is employed and no "close" constraint is put on any dimension. In [45], sequential pattern mining is studied in multi-dimensional context. However, it is still 2D frequent pattern mining along with multi-dimensional projected database. The third or even the fourth dimensions do not fully enumerate on different entries as what the two base dimensions do, and different entries on the third/fourth dimension are only employed to divide the data records into different projected groups. Moreover, no "close" relationships between the third/fourth dimension and the two base dimensions are delivered. Thus, these works cannot be extended to mine FCCs. More recently, [32] and [64] proposed clustering algorithms to analyze clusters on 3D microarray data, however, such algorithms cannot be employed to mine 3D frequent closed patterns.

Third, there are no *parallel* closed frequent pattern mining algorithms in the literature. As data mining is computationally expensive, there has also been a number of attempts to design parallel and distributed mining algorithms. As noted in the survey paper on parallel association mining [62], most of the previous parallel pattern mining algorithms are extensions of their sequential counterparts. For example, Count Distribution is based on Apriori, ParEclat on Eclat, and APM on DIC. However, most of these incur significant communication overhead. Several recently proposed parallel frequent pattern mining algorithms [15, 52], avoid such communication cost with either new data structures or new partition methods. In [15], an algorithm called Inverted Matrix is proposed that exploits replication across parallel nodes, and a relatively small independent tree is built to summarize co-occurrences, which ensures minimum inter-processor communication. In [52], a parallel projection approach for partitioning the transaction data is proposed to mine frequent patterns without communication information. However, all these parallel mining algorithms are limited to frequent pattern mining, to our knowledge, no parallel algorithms for "closed" frequent pattern mining have been reported in the literature.

To overcome these limitations, we propose new algorithms that *progressively* and *efficiently* return FCPs to users in Chapter 3. In Chapter 4, we introduce the concept of frequent closed cube (FCC) that generalizes the notion of 2D frequent closed pattern to 3D context, and propose novel algorithms to mine FCCs from 3D datasets. Moreover, we study the parallel versions of these new algorithms.

2.2 Co-tendency Patterns: Biclustering

While frequent closed pattern mining algorithms are effective in static co-attribute pattern identification, they cannot mine co-tendency patterns with dynamic changes. Instead, clustering is a widely used technique in identifying co-tendency patterns from microarray data.

Clustering analysis is another unsupervised mining technique that partitions a set of objects into clusters such that objects in the same cluster are similar than objects in other clusters. Clustering algorithms are usually classified into two categories: *global* clustering and *subspace* clustering. Many conventional clustering algorithms [14, 49, 18, 28] on gene expression data analysis are classified into *global* clustering as the sample space is globally shared by all resulting clusters. Recently, interactive clustering frameworks [29, 31] are proposed to adopt the domain knowledge in the mining process for higher biological accuracy. Moreover, joint mining algorithms on both gene expression data and protein interaction data are also proposed in [44, 43] to further enhance the accuracy. However, in cellular processes, subsets of genes are usually co-expressed only under certain experimental conditions, but behave almost independently under other conditions. Hence, the *global* clustering results are limited by the existence of a number of samples where the activity of genes is uncorrelated. For this reason, *subspace* clustering was first proposed in [2] to find subsets of objects that appear together under subsets of features. The *subspace* clustering algorithm on microarray data analysis was first introduced by [11] as "biclustering" to simultaneously cluster both genes and experimental conditions, which captures the coherence of a subset of genes under a subset of experimental conditions. As highlighted in [56], discovery of biclusters is essential in revealing the significant connections in gene regulatory networks. Therefore, researchers are motivated to extract a subset of genes whose expression levels rise and fall coherently under a subset of conditions, that is, they exhibit fluctuation of a similar shape when conditions change, which is called "consistent trends".

In [11], the biclustering algorithm begins with the original matrix and iteratively masks out null values and biclusters that have been discovered. The node-deletion and node-addition algorithms are introduced to find submatrices in expression data that have low mean squared residue (*MSR*) score. Let $I \subset X$ and $J \subset Y$ be subsets of genes and conditions. The pair (I, J) specifies the submatrix A_{IJ}. The *MSR* of $A_{I,J}$ is defined as follows:

$H(I,J) = \frac{1}{|I||J|} \sum\limits_{i \in I, j \in J} (d_{ij} - d_{iJ} - d_{Ij} + d_{IJ})^2$ where $d_{iJ} = \frac{1}{|J|} \sum\limits_{j \in J} d_{ij}$, $d_{Ij} = \frac{1}{|I|} \sum\limits_{i \in I} d_{ij}$, $d_{IJ} = \frac{1}{|I||J|} \sum\limits_{i \in I, j \in J} d_{ij}$ are the row and column means and the means in the submatrix A_{IJ}. A submatrix A_{IJ} is called a δ-bicluster if $H(I,J) \leq \delta$ for some $\delta > 0$.

Based on the idea of bicluster model, δ-cluster model [58] is proposed to further accelerate the biclustering process. The δ-cluster model incorporates null values and a move-based algorithm (FLOC) is proposed. FLOC starts at choosing initial biclusters called "seeds" randomly from the original matrix and then proceeds with iterative gene/condition deletion and addition, aiming at achieving the best potential *MSR* score reduction.

Another work [56] also addresses such issue by proposing a depth-first algorithm to mine pClusters. This method clusters dataset row-wise as well as column-wise to find pClusters that satisfy a user specified minimum *pScore*. Given $x, y \in I$, and $a, b \in J$, the *pScore* of a 2×2 matrix is defined as:

$$pScore\left(\begin{bmatrix} d_{xa} & d_{xb} \\ d_{ya} & d_{yb} \end{bmatrix}\right) = |(d_{xa} - d_{xb}) - (d_{ya} - d_{yb})|.$$

Pair (I, J) forms a δ-pCluster if for any 2×2 submatrix X in (I, J), $pScore(X) \leq \delta$ for some $\delta > 0$.

Beside these data mining algorithms, Getz G. et al. devised a coupled two-way iterative clustering algorithm to identify biclusters [19]. The notion of a plaid model is introduced in [35]. It describes the input matrix as a linear function of variables corresponding to its biclusters and an iterative maximization process of estimating a model is presented. Amir Ben-Dor et al. defined a bicluster as a group of genes whose expression levels induce some linear order across a subset of the conditions, i.e., an order preserving sub-matrix [6]. They also proposed a greedy heuristic search procedure to detect such biclusters. Segal E. et al. described many probabilistic models to find a collection of disjoint biclusters which are generated in a supervised manner [48]. Moreover, the idea of bipartite graph is applied in [53] to discover statistically significant biclusters. A two-way interrelated clustering algorithm is proposed in [54] to dynamically manipulate the relationship between the gene clusters and sample groups while conducting an iterative clustering through both of them. Furthermore, [30] applies a pattern-based clustering model which is a generalization of several previous models.

More recently, a deterministic biclustering algorithm DBF is proposed [63] to further improve the biclustering quality and efficiency. DBF is a two-phase algorithm. In phase 1, a set of good-quality biclusters (with low mean squared residue) are generated by the frequent closed pattern mining algorithm CHARM [60]. By modelling the changing tendency between two consecutive experimental conditions as an item, and genes as transactions, a frequent itemset with the supporting genes essentially forms a bicluster. All resulting biclusters are sorted based on the ratio of its mean squared residue over its volume. Only biclusters with low $\frac{MSR}{Volumn}$ are retained as "good seeds" for further refinement. In phase 2, the "good seeds" are iteratively refined by a node addition heuristics. In each iteration, each bicluster is repeatedly tested with columns and rows not included in it to determine if they can be included. The concept of *gain* [58] is applied in the testing. Given a mean squared residue threshold δ, the *gain* of inserting a column/row x into a bicluster c is defined as [63]:

$Gain(x, c) = \frac{r_c - r'_c}{\frac{r^2}{r_c}} + \frac{v'_c - v_c}{v_c}$ where r_c, r'_c are the mean squared residues of bicluster c and bicluster c', obtained by performing the insertion respectively, and v_c and v'_c are the volumes of c and c' respectively.

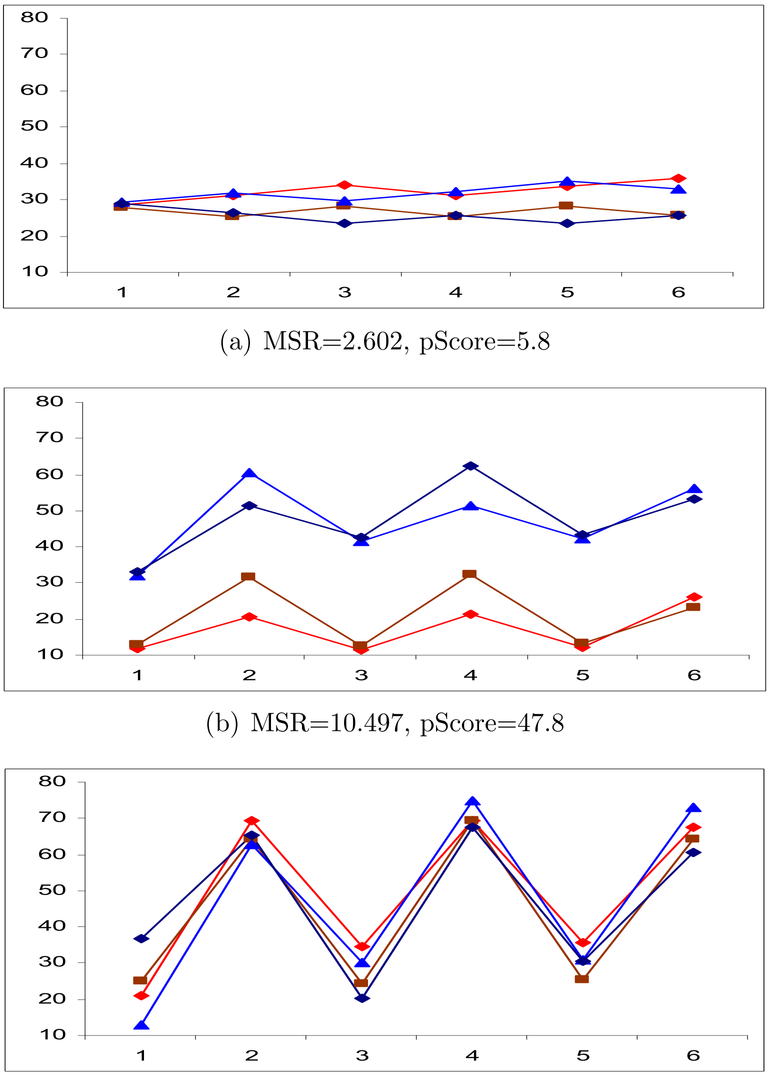
At each iteration, each bicluster is repeatedly extended by an additional gene or condition that has the most gain while keeping the *MSR* below the predetermined threshold δ. A minimum row variance threshold is set to remove biclusters with trivial changes in trends.

The above methods on mining biclusters with consistent trends still have some limitations.

First, the similarity measures of existing methods are inadequate to ensure the consistent trends of biclusters. Existing methods use either *MSR* or *pScore* as the similarity measure for biclustering process. Big volume biclusters with low *MSR* score or *pScore* are defined as "good" biclusters, which are supposed to be generated by the algorithms. Strategies that are based on the *MSR* or *pScore* increase the trend consistency to some extent by pruning off bad patterns with inconsistent trends. However, neither *MSR* nor *pScore* itself is enough to ensure trend consistency of the whole bicluster. Patterns with higher *MSR* score or *pScore* could have more consistent trends than those with lower *MSR* score or *pScore*. Figure 2.2 shows an example of three patterns (a), (b) and (c) with both *MSR* score and *pScore* in increasing order. However, we see clearly that trends in pattern (c) are more consistent than those in pattern (b) and pattern (a).

Hence, we conclude that no single value is enough to control the trend consistency of the whole pattern. Therefore, previous algorithms that take either *MSR* or *pScore* as the main control agent inevitably bring in biclusters with inconsistent trends.

Second, existing methods have two aspects of information loss during mining process. On one hand, it is due to the score (*MSR* or *pScore*) oriented row/column removing process. Since the score of the whole pattern cannot reflect all localized trend consistency, some good genes/conditions would inevitably be removed from the pattern. A tight threshold on the similarity measure would prune off more potential valuable information while a loose one would result in bad patterns. On the



(a) MSR=2.602, pScore=5.8

(b) MSR=10.497, pScore=47.8

(c) MSR=14.641, pScore=94.9

Figure 2.2: Trend Consistency.

other hand, previous algorithms usually work on “part” of the whole dataset. They generate biclusters by improving on either randomly selected “seeds” or good ranked “seeds”. This might miss out a lot of interesting patterns and result in loss of relevant information.

Third, existing methods are not efficient. The seed improvement process follows the hill-climbing paradigm and can involve significant amount of computation. The process often involves the iterative testing of whether the addition/deletion of more rows or columns to/from the biclusters could enhance the similarity score. This testing requires a fair bit of calculation. Moreover, the testing is random and rows/columns are tested one by one. This would result in a long processing time before any acceptable result is returned to users.

Finally, very few inter-bicluster relationships are delivered by previous framework (e.g., which biclusters are closer to each other, which biclusters are remote from each other, and which bicluster is superset/subset of another bicluster). A biclustering algorithm that (bi)clusters a gene expression dataset and provides a graphical representation of the inter-bicluster relationships would be more favored by the biologists. To the best of our knowledge, no previous work has established a clear relationship between biclusters.

Taking into consideration the above limitations of existing works, we propose a new quick hierarchical biclustering algorithm in Chapter 5.

2.3 Time-Lagged Patterns: Time-Lagged Clustering

There are a number of previous approaches for identifying time-lagged gene coregulations. The first is the *Cross-Correlation Method* [33]. Compared with the traditional *Pearson Correlation Coefficient Method*, this method takes into account the time lag issue. However, it is only useful in determining whether two variables have strong global (i.e., similarity is measured over all conditions), but not local time-lagged similarity (i.e., similarity is measured for a subset of conditions). The second method is the *Edge Detection Method* [10]. This method sums up the number of edges of two gene expression curves where the edges have the same direction within a reasonable time lag to generate a score. Edges that are further apart are assigned lower score than those that are nearer. As a result, the gene pairs with higher scores are regarded as the promising pairs with activation relationship. Although this method considers more localized similarities, its current form can only determine potential activation relationships. In these two methods, the regulation direction of gene pairs is not considered. Besides these two methods, *Bayesian Networks* [5] have also been applied; however, the high computational cost renders it impractical. Another approach is the *Dominant Spectral Component Method* [59]. Based on the autoregressive modelling technique, this method decomposes the time series expression sequences into spectral components, and the correlation between two sequences is formulated as a sum of scaled sub-correlations. Although this method looks into the temporal aspect of time series microarray data, it measures only gene-to-gene relationship rather than relationships among multiple gene clusters.

Recently, [34] proposed the *Event Method* to deal with some of the above-mentioned limitations. The algorithm marks the directional changes as an event *Rising (R)*, *Constant (C)*, or *Falling (F)* by calculating the slope of the expression value at each time interval, resulting in a string of events. Then a global sequence alignment algorithm, *Needleman-Wunsch* algorithm, is employed to match the corresponding events of two genes, based on which a numerical score is generated as an indicator of the existing likelihood of regulatory relationship between those two genes. The alignment is run in both directions to decide the regulator and the target gene. As for the inhibition relationship, the event string is first re-encoded by changing each R to F, and vice versa, while C remains unchanged. Then the alignment process is performed again as above. This manner of processing can be regarded as "two genes one relationship per alignment", which means that each alignment can only decide one relationship (activation/inhibition) between two genes. Although this method delivers more information and is relatively efficient, its scoring system to identify promising time-lagged gene pairs is still questionable for the following reasons: first, it cannot tell whether a relative low score is due to a "mismatch" or a "match with a large time lag"; second, the score cannot tell whether two sequences have frequent "short matches" or infrequent "long sequential matches"; third, it is not sensitive to genes whose event sequences are similar for only a small part of time period but different from each other as a whole. Hence, some interesting time-lagged patterns are not always scored high and may be missed out. As for the result, *Event Method* generates only gene pairs without detailed information such as the exact lagged-time, and starting and ending time of the co-regulation. Moreover, it tests all combinations of two genes, which is not very efficient to some extent, and finally, it only provides results between two genes, but not co-regulated relationships between gene clusters.

The limitations of existing work on time-lagged pattern identification motivate us to find a new time-lagged clustering algorithm to identify localized time-lagged co-regulations between genes and/or gene clusters efficiently in Chapter 6.

2.4 Data Preprocessing

Data preprocessing is important for microarray data analysis in that the gene expression data tend to be incomplete, noisy, and inconsistent. Data preprocessing includes data cleaning, data integration, data transformation and data reduction [21]. We mainly review data transformation and data reduction techniques as they are closely related to our work.

2.4.1 Data Transformation

Data transformation is a process to convert data into an appropriate form for mining. Normalization is the main transformation technique that scales the values of an attribute so that they fall within a small specified range, such as 0.0 to 1.0. Normalization helps prevent attributes with initially large ranges from outweighing attitudes with initially smaller ranges. There are many normalization methods in the literature. We review three notable methods [21]: min-max normalization, z-score normalization, and normalization by decimal scaling.

Min-max normalization performs a linear transformation on the original data. Let min_A and max_A be the minimum and maximum values of the attribute A. Min-max normalization maps a value v of A to v' in the range $[new\_min_A, new\_max_A]$ by computing $v' = \frac{v - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A$.

Min-max normalization preserves the relationships among the original data values.

It will encounter an "out of bounds" error if a future input case for normalization falls outside the original data range for A.

Z-score normalization (or zero-mean normalization) performs the attribute value normalization based on the mean and standard deviation of the attribute values. A value v of attribute A is normalized to v' by computing $v' = \frac{v-\bar{A}}{\sigma_A}$, where $\bar{A}$ and σ_A are the mean and standard deviation, respectively, of attribute A.

Z-score normalization is useful when the actual minimum and maximum of attribute A are unknown, or when there are outliers that dominate the min-max normalization.

Normalization by decimal scaling normalizes by moving the decimal point of values of attribute A. The number of decimal points moved depends on the maximum absolute value of A. A value v of A is normalized to v' by computing $v' = \frac{v}{10^j}$, where j is the smallest integer such that $Max(|v'|) < 1$.

In this thesis, since this is not the focus of our research, we simply adopt the ideas of Z-score normalization and normalization by decimal scaling in the data transformation.

2.4.2 Data Reduction

Data reduction techniques can be applied to obtain a reduced representation of the data, yet closely maintain the integrity of the original data. The use of concept hierarchies for data discretization is an alternative form of data reduction. Discretization techniques can be used to reduce the number of values for a given continuous attribute, by dividing the range of the attribute into intervals. Interval labels can then be used to replace actual data values. The numeric attributes of microarray data are usually reduced into concept hierarchies before the mining tasks. The concept hierarchies for numeric attributes can be constructed automatically based on data distribution analysis. There are five main methods for numeric concept hierarchy generation [21]: binning, histogram analysis, cluster analysis, entropy-based discretization, and data segmentation by “natural partitioning”.

Binning: The attribute values can be discretized by distributing the values into bins, and replacing each bin value by the bin mean or median.

Histogram analysis: The histogram for an attribute A partitions the data distribution of A into disjoint subsets, or buckets. The buckets could be determined by the partitioning rules such as Equiwidth, Equidepth, V-Optimal and MaxDiff. In an Equiwidth histogram, the width of each bucket range is uniform; in an Equidepth histogram, each bucket contains roughly the same number of contiguous data samples; in a V-optimal histogram, the histogram variance is a weighted sum of the original values that each bucket represents, where bucket weight is equal to the number of values in the bucket; In a MaxDiff histogram, a bucket boundary is established between each pair for pairs having the $\beta - 1$ largest differences, where β is user-specified.

Cluster analysis: A clustering algorithm can also partition data into groups such that each cluster forms a node of a concept hierarchy.

Entropy-based analysis: An information-based measure called *entropy* can be used to recursively partition the values of a numeric attribute A. Unlike other methods, entropy-based discretization uses class information.

Segmentation by natural partitioning: The numerical ranges are partitioned into relatively uniform, easy-to-read intervals that appear intuitive or “natural”.

The ideas of binning, equiwidth partitioning, and natural partitioning are applied in the algorithms of this thesis.

2.5 Summary

This chapter reviews the existing algorithms for mining the co-attribute patterns, co-tendency patterns and time-lagged patterns. The advantages and limitations of some notable algorithms are analyzed, which serve as a background for our new methods development in the following chapters. Some notable data preprocessing methods that could improve the efficiency and ease of the mining process are also reviewed.

Chapter 3

Mining 2D Frequent Closed Patterns from Dense Datasets

3.1 Overview

In this chapter, we address the problem of mining frequent closed patterns (FCPs) from dense 2D datasets. As we shall see, the real microarray datasets we used are dense where approximately 50% of the cells are ”1”s while the rest are ”0”s. Existing techniques (as noted in Chapter 2: Literature Review) cannot handle such datasets.

We present a framework that allows us to mine FCPs from *dense* datasets efficiently and *progressively*. The framework comprises two phases. In the first phase, the mining space is partitioned into a number of smaller subspaces such that (a) each subspace can be mined independently, and (b) the union of the FCPs from all subspaces is a superset of the FCPs obtained from the original space. In the second phase, each subspace is mined independently to return the FCPs. The crucial task in this phase is to prune away redundant FCPs (those that may also be produced in other subspaces) and false drops (those that are FCPs in the subspace but are not FCPs in the original space). Such a framework has two key advantages. First, it facilitates progressiveness - as each subspace can be independently mined, we can return its answers to the users without having to wait for all subspaces to be completely processed. This means users enjoy short initial response time, and are no longer overwhelmed by all the answers at the same time. Second, the schemes are amiable to parallelism with little or no synchronization (and hence negligible communication overhead) - the subspaces can be mined independently and concurrently across a number of parallel sites. This is critical as, to our knowledge, there is no reported work in the literature on parallel FCP mining.

Based on this framework, we propose two algorithms, *C-Miner* and *B-Miner*, for efficient and progressive mining of FCPs. The two schemes differ in two ways. First, the partition methods are different: *C-Miner* partitions the mining space based on Compact Rows Enumeration while *B-Miner* partitions the space based on Base Rows Projection. Second, because the partitioning methods are different, different pruning strategies are used in the second phase.

We have implemented *C-Miner* and *B-Miner*, and experimented with synthetic datasets and three real microarray datasets. Our results show that our *C-Miner* and *B-Miner* are superior to Closet+, REPT and D-Miner on dense datasets. We report results on parallel versions of our proposed schemes. We also show that the FCPs obtained from our methods are of biological significance.

The rest of this chapter is organized as follows. In the next section, we present some preliminaries. Section 3.3 presents the proposed *C-Miner* and *B-Miner* algorithms. In Section 3.4, we report experimental results obtained from comparing *C-Miner* and *B-Miner* against some existing schemes. Finally, we conclude in Section 3.5.

3.2 Preliminaries

We shall first define some notations that we will be using throughout this chapter, and then give the problem description.

Let $R = \{r_1, r_2, \ldots, r_n\}$ denote a set of rows, and $C = \{c_1, c_2, \ldots, c_m\}$ denote a set of columns. In the dataset, each row r_i contains a set of columns, and each column c_j is contained in a set of rows. In this chapter, we represent a dataset by a binary matrix $O = n \times m$, where cell $O_{i,j}$ corresponds to the relationship between row i and column j; a value true (i.e., "1") denotes the "containing/contained" relationship; and a false value otherwise. Table 3.1 shows an example. In the table, r_3 contains c_2 and c_6, denoted as $C(r_3) = \{c_2, c_6\}$; and c_7 is contained in r_5 and r_6, denoted as $R(c_7) = \{r_5, r_6\}$.

Table 3.1: A Sample Dataset (Matrix O).

R/C	c_1	c_2	c_3	c_4	c_5	c_6	c_7
r_1	1	0	0	0	1	1	0
r_2	1	1	0	0	0	1	0
r_3	0	1	0	0	0	1	0
r_4	1	1	1	1	1	1	0
r_5	1	1	0	1	0	0	1
r_6	1	0	1	1	0	0	1

Definition 3.1 Column Support Set $R(C')$: *Given a set of columns $C' \subseteq C$, the maximal set of rows that contain C' is defined as the Column Support Set $R(C') \subseteq R$.*

For example, in Table 3.1, let $C' = \{c_1, c_4\}$, then $R(C') = \{r_4, r_5, r_6\}$ since r_4, r_5 and r_6 contain c_1 and c_4, and no other rows contain both two columns.

Definition 3.2 Row Support Set $C(R')$: *Given a set of rows $R' \subseteq R$, the maximal set of columns that contain R' is defined as the Row Support Set $C(R') \subseteq C$.*

For example, in Table 3.1, let $R' = \{r_1, r_2\}$, then $C(R') = \{c_1, c_6\}$ since c_1 and c_6 are contained in r_1 and r_2, and no other columns are contained in both two rows.

Definition 3.3 Support $|R(C')|$: *Given a set of columns C', the number of rows in the dataset that contain C' is defined as the Support of C', denoted as $|R(C')|$.*

Definition 3.4 Closed Patterns (CP): *A set of columns $C' \subseteq C$ is called a closed pattern if there exists no C'' such that $C' \subseteq C''$ and $|R(C'')| = |R(C')|$.*

Definition 3.5 Frequent Closed Patterns (FCP): *A set of columns $C' \subseteq C$ is called a frequent closed pattern if (1) $|R(C')|$, the support of C', is higher than a minimum support threshold; and (2) C' is a closed pattern.*

For example, given that $minsup = 1$, the column set $\{c_1, c_5, c_6\}$ will be a frequent closed pattern in Table 3.1 since it occurs two times which is more than the *minsup* threshold. However, $\{c_2, c_3\}$ is not a frequent closed pattern in that it has a superset $\{c_1, c_2, c_3\}$ and $|R(\{c_1, c_2, c_3\})| = |R(\{c_2, c_3\})|$.

Definition 3.6 Data Density: *The Data Density (denoted as Density) is defined as the percentage of cells containing value "1" in the (boolean) dataset.*

Definition 3.7 Pattern Length: *Given a frequent closed pattern, the number of columns contained in the pattern is defined as the Pattern Length, denoted as Len.* For example, given the frequent closed pattern $\{c_1, c_5, c_6\}$, the pattern length $Len = 3$.

Problem Definition (FCP Mining): Given a dataset O, our problem is to discover all FCPs with respect to a user support threshold *minsup* and a user pattern length threshold *minlen*.

Before leaving this section, we would like to point out that we will often need to refer to the column support set of a FCP. As such, for convenience, we will also refer to the submatrix $R(FCP) \times FCP$ as FCP.

3.3 Progressive FCP Mining

In this section, we first present the basic framework for progressive FCP mining. We then present the two schemes, *C-Miner* and *B-Miner*, that are based on the framework. Finally, we show how the framework can be easily adapted for parallel FCP mining.

3.3.1 A Framework for Progressive FCP Mining

Let O be the original dataset (matrix) to be mined. We shall refer to O as a space; in this case, the original mining space. Let $MineFCP(M)$ denote the set of frequent closed patterns (FCPs) mined from space M. The basic idea of the framework, as illustrated in Figure 3.1, comprises two phases - subspace generation phase, and subspace mining phase.

In the first phase, the subspace generation phase, the original mining space O is recursively[1] split into submatrices/subspaces $S_1, S_2, \ldots, S_t$, $t \geq 1$, such that

$$MineFCP(O) \subseteq \cup_{i=1}^{t} MineFCP(S_i) \tag{3.3.1}$$

In other words, the original space is split such that the union of the FCPs mined from all the subspaces may be a superset of the actual answer. This property allows us to mine the various subspaces independently and concurrently. In this way, answers obtained from a subspace can be returned immediately to the users and hence realizing progressiveness. Moreover, since a subspace is smaller than the original data space, it can be mined more efficiently. As already mentioned, the ability to mine the subspaces independently facilitates parallelism.

[1] In our current implementation, we adopt only one level of splitting.

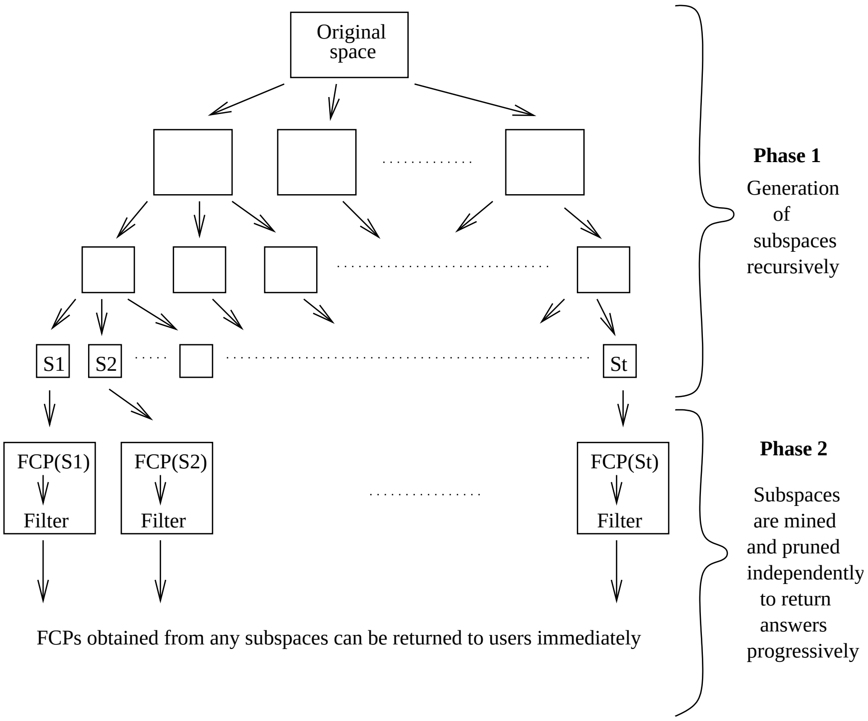

Original space
S1
S2
St
FCP(S1)
Filter
FCP(S2)
Filter
FCP(St)
Filter
FCPs obtained from any subspaces can be returned to users immediately
Phase 1
Generation of subspaces recursively
Phase 2
Subspaces are mined and pruned independently to return answers progressively

Figure 3.1: The progressive framework.

In the second phase, the subspace mining phase, each subspace is mined for FCPs independently. However, as noted in Equation 3.3.1, the FCPs mined from a subspace may contain patterns that are not answers. There are two scenarios in which this can happen: (a) the FCPs mined from a subspace may contain false drops - a pattern is an FCP of the subspace, but not globally (i.e., not in the original space), and (b) the FCPs mined from a subspace is redundant, i.e., the FCP may be mined from multiple subspaces. In other words,

$$MineFCP(S_i) \cap MineFCP(S_j) \neq \emptyset$$

As such, a pruning mechanism must be deployed to remove such non-global and redundant FCPs so that only answers are returned. As shown in Figure 3.1, as soon as answers are generated from a subspace, they can be returned to users.

In the next two subsections, we shall present two algorithms, *C-Miner* and *B-Miner*, that are based on this framework. The two schemes differ in how the original space is partitioned, and hence the pruning strategies.

3.3.2 Algorithm *C-Miner*

In this section, we shall present *C-Miner* which is based on Compact Rows Enumeration.

Partitioning the Mining Space

The partitioning phase of *C-Miner* comprises four steps. In the first step, similar rows in the original dataset O, which is an $n \times m$ binary matrix, are grouped together by clustering. Any clustering algorithm can be employed here. In our experimental study, we have used a well-known gene expression clustering software CLUTO[2]. The number of clusters k is a user specified parameter. In CLUTO, the desired k-way clustering solution is computed by performing a sequence of $k-1$ repeated bisections. In this approach, the matrix is first clustered into two groups, then one of these groups is selected and bisected further. This process continues until the desired number of clusters is found. During each step, the cluster is bisected so that the resulting 2-way clustering solution optimizes the clustering criterion function that $maximizes \sum_{i=1}^{k} \sqrt{\sum_{v,u \in R_i} sim(v,u)}$, where R_i is the set of rows assigned to the ith cluster, v and u represent two rows, and $sim(v,u)$ is the similarity between two rows. The cluster to be selected for further partitioning is controlled by the rule that its bisection will optimize the value of the overall clustering criterion function the most.

In the second step, rows within the same cluster are combined to form a new

[2] http://www-users.cs.umn.edu/~karypis/cluto/

compact row, called cluster-row. Let $G = \{r_1, r_2, \ldots, r_q\}$ be the set of rows of a particular cluster D. Then the cluster can be represented as $D = q \times m$ matrix. The cluster-row of D, denoted $L = \{l_1, l_2, \ldots, l_m\}$, is formed according to the rule that $l_j = \sum_{i=1}^{q} \bigvee d_{i,j}$, where $j = 1, 2, \ldots, m$. That is, the cell value of the cluster-row is 0 only when all of its make-up values are 0; otherwise, the cell value is 1. By the above processing, O is transformed into a compact matrix $O' = l \times m$, where l is the number of clusters and $l \leq n$. Given matrix O in Table 3.1 for example, let us suppose that its rows "r_1, r_2, r_3" and "r_5, r_6" are grouped into clusters L_1 and L_3 respectively, then its compact matrix O' is shown in Table 3.2.

Table 3.2: Compact Matrix O'.

	c_1	c_2	c_3	c_4	c_5	c_6	c_7
$l_1(r_1, r_2, r_3)$	1	1	0	0	1	1	0
$l_2(r_4)$	1	1	1	1	1	1	0
$l_3(r_5, r_6)$	1	1	1	1	0	0	1

In the third step, *C-Miner* applies a compact row enumeration strategy on the compact matrix O' to divide the space O into subspaces. In previous works, rows (or columns) to enumerate have equal weight during processing. In *C-Miner*, the weight of each cluster-row is the number of its make-up rows (i.e., number of rows of the corresponding cluster). Hence, during the cluster-row enumeration, the *Support* of a subspace is given by the sum of the weights of the corresponding cluster-rows. We refer to the subspace generated in this step (on O') as the Compact Subspace (CS). While any existing row enumeration algorithm can be employed, they have to be adapted to handle weighted enumeration. Since the process of row enumeration is equal to the process of recursively removing from the matrix either a row or all cells

Table 3.3: Cutters.

$C(X,Y)$
$C(l_1, c_3c_4c_7)$
$C(l_2, c_7)$
$C(l_3, c_5c_6)$

valued “0” of the row, we adopt a depth-first tree splitting strategy (similar to D-Miner [7]) for compact row enumeration, which works efficiently on dense data.

Table 3.4: Resulting CSs and Subpaces ($minsup = 3, minlen = 2$).

Cluster-Row Set	Original Row Set	Original Column Set	Support	Pattern Length
l_1, l_2	r_1, r_2, r_3, r_4	c_1, c_2, c_5, c_6	4	4
l_1, l_2, l_3	$r_1, r_2, r_3, r_4, r_5, r_6$	c_1, c_2	6	2
l_2, l_3	r_4, r_5, r_6	c_1, c_2, c_3, c_4	3	4

The scheme works as follows. We group all cells with value “0” in each cluster-row together, and define each group as a cutter $C(X, Y)$ where $X \subseteq L$ and $Y \subseteq C$. Thus, the number of cutters is equal to the number of rows containing at least a “0” element. Then in cutter $C(X, Y)$, $\forall l_i \in X$, both $\forall c_j \in Y, O'_{i,j} = 0$ and $\forall c_k \in (C \setminus Y), O'_{i,k} = 1$ are satisfied. Table 3.3 shows the 3 cutters generated from the running example of matrix O' in Table 3.2.

The splitting tree takes the whole compact matrix $O'(L, C)$ as the root and splits it recursively using each cutter until all cutters are used and consequently all cells in each resulting CS have the value “1”. A cutter $C(X, Y)$ is used to cut a node (L', C') if $X \cap L' \neq \emptyset$ and $Y \cap C' \neq \emptyset$. By convention, we define the left son of the node by $(L' \setminus X, C')$ and the right son by $(L', C' \setminus Y)$. The resulting CSs represent a full enumeration of cluster-rows that satisfies the support and pattern length constraints. Only nodes not satisfying the *minsup* and *minlen* are pruned off. Thus, no valuable information for FCP mining is removed during subspace dividing process. The support of a node is calculated by the weight sum of its cluster-rows rather than the number of its cluster-rows.

Let *minsup* = 3 and *minlen* = 2. Figure 3.2 shows the splitting tree of our running example and the CSs generated are shown in Table 3.4(made up of column 1,3,4,5).

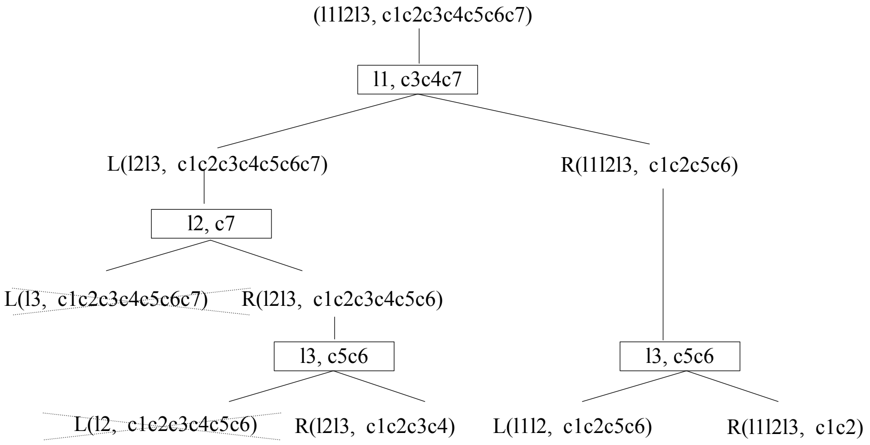


Figure 3.2: Splitting tree using cutters.

We note that the ordering in which cutters are applied affects performance. As a heuristic, cutters with more 0s are applied first as it will result in a shorter tree (and hence more efficient processing).

Finally, in step four, for each CS, its cluster-rows are "decompressed" back into their original rows. The decompression introduces new cells that may contain 0s in the corresponding dataset. Now, each of these datasets forms a subspace from which we can mine the actual FCPs (of the original dataset). Considering the CSs in Table 3.4, we have, after decompression, the resulting subspaces as shown in Table 3.4 (columns 2-5).

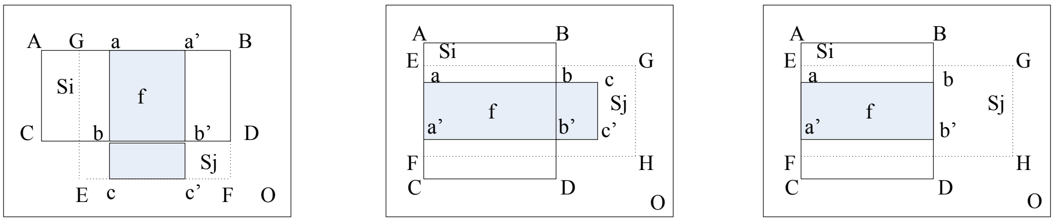
Lemma 1. Let O be the original mining space. Let the subspaces generated by Phase 1 of *C-Miner* from O be $S_1, S_2, \ldots, S_t$, $t \geq 1$. Then $MineFCP(O) \subseteq \cup_{i=1}^{t} MineFCP(S_i)$.

Proof: To prove Lemma 1 holds, we need to show that every FCP that can be determined from O can be mined from one of the subspaces. Consider an arbitrary FCP from O, say $A = \{r_1, \ldots, r_u\} \times \{c_1, \ldots, c_v\}$ where $r_i \in R$ and $c_j \in C$. Clearly, $\forall i, j,\ O(r_i, c_j) = 1$. Let clusters $C_1, \ldots, C_q$ be the clusters containing rows $r_1, \ldots, r_u$, where $q \leq u$. Let $l_1, \ldots, l_q$ be the corresponding cluster-rows of these clusters respectively. Then we get $O'(l_i, c_j) = 1$ (deduced from the row combination rule). Hence, by the cluster-row enumeration of O', $A' = \{l_1, \ldots, l_q\} \times \{c_1, \ldots, c_v, c_{v+1}, \ldots, c_m\}$ will be generated. Given that we take a full enumeration of cluster-rows using the splitting tree, and only prune off unsatisfactory compact subspaces, the resulting compact subspaces are complete. Then A' will be decompressed into the subspace $A'' = \{r_1, r_1', r_1'', \ldots, r_u, r_u'\} \times \{c_1, \ldots, c_v, c_{v+1}, \ldots, c_m\}$ where r_1, r_1', r_1'' are rows in Cluster 1. Clearly, the subspace A'' is actually the superset of A. Since $A = \{r_1, \ldots, r_u\} \times \{c_1, \ldots, c_v\}$ is a FCP, it will be mined out in phase 2 of C-Miner from $A'' = \{r_1, r_1', r_1'', \ldots, r_u, r_u'\} \times \{c_1, \ldots, c_v, c_{v+1}, \ldots, c_m\}$. Thus, Lemma 1 holds. □

Mining Subspaces to Generate FCPs

To produce the actual FCPs, each subspace is mined independently. We used D-Miner [7] in this phase as it is quick at picking out 0s in very dense dataset with very few 0s.

From Lemma 1, we note that it is possible for a FCP f extracted from a subspace S_i (i.e., $f \in MineFCP(S_i)$), to be a *false drop* (i.e., $f \notin MineFCP(O)$) or f may also be extracted from another subspace S_j (i.e., $f \in MineFCP(S_j), i \neq j$). There are three cases in which false drops and redundancy may occur (see Figure 3.3).



(1) row set unclosed (2) column set unclosed (3) redundancy

Figure 3.3: False drops and redundancy.

- Case (1): *The supporting row set of $f \in MineFCP(S_i)$ is not globally closed.* This case occurs when there exists a row $r_x \in R$, which is outside subspace S_i but contains C_f (column set of f). Then, there must exist $f' \in MineFCP(S_j)$ such that $f \subset f'$. For example, $f = aa'bb'$ mined from subspace $S_i = ABCD$ is not globally row-set closed in that $f' = aa'cc'$ from subspace $S_j = GBEF$ is f's superset. Hence, we conclude that, given $f = (R_f \times C_f) \in MineFCP(S_i)$, if there exists a row $r_x \in R$ and $r_x \notin R_i$ (row set of S_i) such that $\forall c_y \in C_f, O_{x,y} = 1$, then f should be pruned off.

- Case (2): *The column set of $f = (R_f \times C_f) \in MineFCP(S_i)$ is not globally closed.* Let $S_i = \{l_{i1}, l_{i2}, \ldots, l_{iu}\} \times C_i$ where C_i is the column set and l_{ix} is the cluster-row contributing to S_i. Let $L_{i1}, L_{i2}, \ldots, L_{iu}$ be the corresponding row sets (in the original dataset) from which each cluster-row is generated. Suppose $\exists l_{ix} \in \{l_{i1}, l_{i2}, \ldots, l_{iu}\}$ such that $R_f \cap L_{ix} = \emptyset$, that is, some cluster-row does not contribute to f. Then there must exist another subspace without such contributing cluster-row $S_j = (\{l_{i1}, l_{i2}, \ldots, l_{iu}\} \setminus l_{ix}) \times C_j$ where $C_i \subset C_j$. It follows that $\exists f' = (R'_f \times C'_f) \in MineFCP(S_j)$ such that $R_f = R'_f$ and $C_f \subseteq C'_f$. We defer the equality case "$C_f = C'_f$" to Case (3) below. Here, if

$C_f \subset C'_f$, f should be pruned off in that it is not globally closed in column set. For example, $f = aa'bb'$ from $S_i = ABCD$ is not globally column-set closed if there exists $f' = aa'cc'$ from subspace $S_j = EFGH$.

- Case (3): *$f \in MineFCP(S_i)$ is redundant in that $f \in MineFCP(S_j)$.* Following the prerequisites of Case (2) above, if $C_f = C'_f$, then $f = f'$. Hence, f is redundant and can be pruned off from S_i. For example, $f = aa'bb'$ can be mined out in both $S_i = ABCD$ and $S_j = EFGH$.

Based on above observations, we can ensure that our final results contain all and only the right answers. Before we prove this result, we give the definition of Compact Row Set first.

Definition 3.8 Compact Row Set: *Given a compact subspace $S_i = \{l_{i1}, l_{i2}, \ldots, l_{iu}\} \times C_i$ where C_i is the column set and l_{ix} is the cluster-row contributing to S_i, we define $L_{i1}, L_{i2}, \ldots, L_{iu}$ as the corresponding Compact Row Set (in the original dataset) from which each cluster-row is generated.*

Given the cluster-row l_1 in Table 3.2 for example, the corresponding Compact Row Set is $L_1 = \{r_1, r_2, r_3\}$.

Now, for each FCP f generated from subspace S_i, we drop redundant FCPs or false drops based on the pruning rules given in Lemma 2: (i) condition (a) implies that some Compact Row Set of S_i does not contribute to f. The pruning by condition (a) removes any f that is either not globally closed in column set or redundant; (ii) condition (b) implies that some other rows outside S_i contain f's column set. The pruning by condition (b) drops any f that is not globally closed in row set.

Lemma 2. Let O be the original space. Let $S_1, \ldots, S_t$ be the subspaces generated in phase 1 of *C-Miner*. Let $S_i = R_i \times C_i$ and let $f = (R_f \times C_f) \in \mathit{MineFCP}(S_i)$. Then f can be pruned if (a) $\exists L_{ix} \subset R_i$ such that $R_f \cap L_{ix} = \emptyset$; or (b) $\exists r_y \in (R \setminus R_i)$ such that $\forall c_z \in C_f, O_{y,z} = 1$.

Proof: Let $f \in \mathit{MineFCP}(S_i)$. If (a) holds, $R_f \subseteq (R_i \setminus L_{ix})$. Since there exists another subspace $S_j = (R_i \setminus L_{ix}) \times C_j$ where $C_i \subset C_j$, $f \subset S_j$. Hence, $\exists f' = R_f \times C'_f = \mathit{MineFCP}(S_j)$. If $C_f = C'_f$, $f = f'$, then f can be pruned off as redundancy; if $C_f \subset C'_f$, f can be pruned off due to unclosed column set. If (b) holds, then $\exists f' = R(C_f) \times C_f$ where $r_y \in R(C_f)$. Since $f = R_f \times C_f \in \mathit{MineFCP}(S_i)$, $R_f \subset R(C_f)$, hence, f can be pruned off due to unclosed row set. □

Lemma 3. Let O be the original space. Let $S_1, \ldots, S_t$ be the subspaces generated in phase 1 of *C-Miner*. Let $P_1, \ldots, P_t$ be the set of FCPs that are pruned from the corresponding subspaces in phase 2. Then, $\mathit{MineFCP}(S_i) - P_i \subset \mathit{MineFCP}(O)$.

Proof: Suppose $\exists f \in \mathit{MineFCP}(S_i) - P_i$ and $f \notin \mathit{MineFCP}(O)$. Since $f \notin \mathit{MineFCP}(O)$, there must exist a subspace S_j such that there exists $f' = R'_f \times C'_f \in \mathit{MineFCP}(S_j)$ and $f' \in \mathit{MineFCP}(O)$ such that either (1) $R_f = R'_f$ and $C_f \subset C'_f$ or (2) $R_f \subset R'_f$ and $C_f = C'_f$. Since f is not pruned off, it means both conditions (a) and (b) (in Lemma 2) are violated. Violation of condition (a) indicates that $\forall L_{ix} \subset R_i, R_f \cap L_{ix} \neq \emptyset$, and $\forall L_{jx} \subset R_j, R'_f \cap L_{jx} \neq \emptyset$. Violation of condition (b) indicates that $R_f = R(C_f)$ and $R'_f = R(C'_f)$. Suppose (1) $R_f = R'_f$ and $C_f \subset C'_f$ is satisfied. From $R_f = R'_f$, we know that $S_i = S_j$ (violation of condition (a)). Since $C_f \subset C'_f$, f' instead of f will be mined out from S_i, which contradicts the supposition that $\exists f \in \mathit{MineFCP}(S_i) - P_i$. Suppose (2) $R_f \subset R'_f$ and $C_f = C'_f$ is satisfied. From $C_f = C'_f$, we know that $R_f = R(C_f) = R(C'_f) = R'_f$ (violation of condition (b)), which contradicts $R_f \subset R'_f$. Hence, the supposition is violated. Thus, we conclude that $MineFCP(S_i) - P_i \subset MineFCP(O)$. □

Theorem 1. Let O be the original space. Let $S_1, \ldots, S_t$ be the subspaces generated in phase 1 of *C-Miner*. Let $P_1, \ldots, P_t$ be the set of FCPs that are pruned from the corresponding subspaces in phase 2. Then $MineFCP(O) = \cup_{i=1}^{t}(MineFCP(S_i) - P_i)$.

Proof: The proof follows from Lemma 1- 3. □

In our running example, after phase 2, the resulting FCPs are shown in Table 3.5.

Table 3.5: FCP($minsup = 3, minlen = 2$).

support set	FCP	support	pattern length
r_1, r_2, r_4	c_1, c_6	3	2
r_2, r_3, r_4	c_2, c_6	3	2
r_2, r_4, r_5	c_1, c_2	3	2
r_4, r_5, r_6	c_1, c_4	3	2

3.3.3 Algorithm *B-Miner*

We shall now examine algorithm *B-Miner*. *B-Miner* is based on Base Rows Projection.

Partitioning the Mining Space

B-Miner partitions the space $O = R \times C$ in two steps: row set partition and column set partition. In the first step, the row set R is partitioned into several row groups, defined as *Base Row Groups (BRGs)*. The number of rows in each BRG is the same, which is a user specified parameter, defined as *Group Length (GL)*. Given $GL = k$, the row set $R = \{r_1, r_2, \ldots, r_n\}$ is partitioned into q BRGs: $\{r_1, r_2, \ldots, r_k\}$, $\{r_{k+1}, r_{k+2}, \ldots, r_{2k}\}$, ..., $\{r_{q\times k+1}, r_{q\times k+2}, \ldots, r_{q\times k}\}$, where $q = \lfloor \frac{n}{k} \rfloor + 1$. Given a

$BRG_l = \{r_{(l-1)\times k+1}, r_{(l-1)\times k+2}, \ldots, r_{l\times k}\}$, $\{r_1, r_2, \ldots, r_{(l-1)\times k}\}$ is defined as BRG_l's *Former Row Set* FRS_l; and $\{r_{l\times k+1}, \ldots, r_n\}$ is defined as BRG_l's *Latter Row Set* LRS_l.

In the second step, by projection on each BRGs, column set $C = \{c_1, c_2, \ldots, c_m\}$ is partitioned into q column groups, defined as *Base Column Groups (BCGs)*. For the lth Base Row Group $BRG_l = \{r_{(l-1)\times k+1}, r_{(l-1)\times k+2}, \ldots, r_{l\times k}\}$, the Base Column Group $BCG_l = \{c'_1, c'_2, \ldots, c'_m\}$ where $\{c'_1, c'_2, \ldots, c'_m\} \subseteq C$ and $\forall c'_j \in \{c'_1, c'_2, \ldots, c'_m\}$, $\sum_{i'=(l-1)\times k+1}^{l\times k} \bigvee O_{i',j'} = 1$.

Each subspace is made up of three elements: BRG, LRS, and BCG. Hence, the ith subspace $S_i = (BRG_i \cup LRS_i) \times BCG_i$, which is also equivalent to $S_i = LRS_{i-1} \times BCG_i$. Given matrix O in Table 3.1 for example, with $GL = 2$, there are three subspaces generated: $S_1 = \{r_1, r_2, r_3, r_4, r_5, r_6\} \times \{c_1, c_2, c_5, c_6\}$, $S_2 = \{r_3, r_4, r_5, r_6\} \times \{c_1, c_2, c_3, c_4, c_5, c_6\}$, $S_3 = \{r_5, r_6\} \times \{c_1, c_2, c_3, c_4, c_7\}$.

FCPs will not be generated in the subspace that has fewer rows than *minsup*. Hence, the number of subspaces $q = \lfloor \frac{(n-minsup)}{k} \rfloor + 1$ rather than $\lfloor \frac{n}{k} \rfloor + 1$. It is safe to ignore those latter subspaces without enough rows. Column sets with enough row support have already been covered by the former subspaces. For the above example, if we set $minsup = 3$, only the first two subspaces (S_1 and S_2) will be mined. The last subspace S_3 with only 2 rows is safe to be dropped off.

Lemma 4. Let O be the original mining space. Let the subspaces generated by Phase 1 of *B-Miner* from O be $S_1, S_2, \ldots, S_t$, $t \geq 1$. Then $MineFCP(O) \subseteq \cup_{i=1}^{t} MineFCP(S_i)$.

Proof: To prove Lemma 4 holds, we need to show that every FCP that can be determined from O can be mined from one of the subspaces. Consider an arbitrary FCP from O, say $A = \{r_1, \ldots, r_u\} \times \{c_1, \ldots, c_v\}$ where r_1 is the first row. There must exist a subspace S_i that $r_1 \in BRG_i$. Hence, $\{r_1, \ldots, r_u\} \subseteq (BRG_i \cup LRS_i)$ that $r_f \in BRG_i$. According to the projection rules, r_f's column support set $r_f(C') \subseteq BCG_i$. Since $\{c_1, \ldots, c_v\} \subseteq r_f(C')$, then $\{c_1, \ldots, c_v\} \subseteq BCG_i$. Moreover, $S_i = (BRG_i \cup LRS_i) \times BCG_i$, then $A \subseteq S_i$. Hence, $A \subseteq MineFCP(S_i)$. Thus, Lemma 4 holds. □

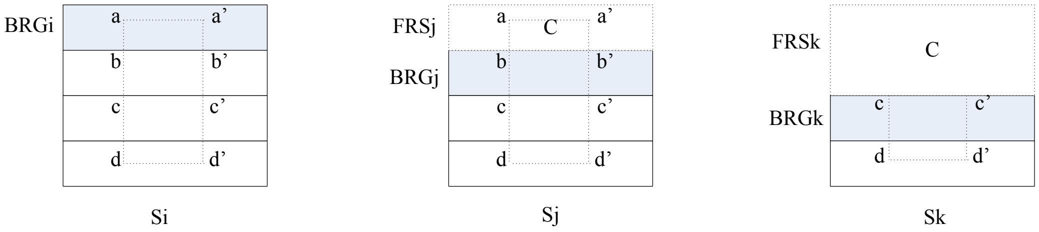


Figure 3.4: Subspace pruning.

Mining Subspaces to Generate FCPs

Like *C-Miner*, any FCP mining algorithm can be applied on each subspace to mine the FCPs. We also adopt D-Miner here. However, due to the way the space is partitioned, some local FCPs are either globally unclosed or redundant (appearing in several different subspaces).

Figure 3.4 shows several examples. Consider three consecutive subspaces S_i, S_j, and S_k. It is clear that a pattern mined from previous subspaces may appear again in the latter subspaces. For example, pattern *bb′dd′* from S_i may also appear in S_j, and *cc′dd′* from S_i may also appear in both S_j and S_k. Such are cases of redundancy. Moreover, pattern from the latter subspaces may be unclosed if its column set is contained in its FRS. Given *bb′dd′* from S_j for example, since its column set also exists in FRS_j, it is unclosed in that pattern *aa′dd′* from the former subspace S_i is its superset and is globally closed.

To thoroughly remove the globally unclosed and/or redundant FCPs, we develop two pruning strategies (see Lemma 5). The first condition implies that the FCP does not contain any row of its subspace's BRG. The pruning condition thus ensures no redundancy, that is, FCPs from a certain subspace will not appear again in latter subspaces. For example, in Figure 3.4, FCPs from S_i without supporting row in BRG_i such as $bb'dd'$, $cc'dd'$ will be pruned off, while FCPs such as $aa'dd'$, $aa'cc'$ will be retained. The second condition implies that there is a row in its subspace's FRS that contains the FCP's full column set. The pruning condition thus ensures no globally unclosed FCPs, that is, FCPs from a certain subspace will not have any superset in its former subspaces. For example, in Figure 3.4, FCPs from S_j but with supporting row in FRS_j such as $bb'dd'$ will be pruned off in that it has a superset $aa'dd'$ in former subspace S_i.

Lemma 5. Let O be the original space. Let $S_1, \ldots, S_t$ be the subspaces generated in phase 1 of *B-Miner*. Let $FCP_i = \{r_{i1}, \ldots, r_{iu}\} \times \{c_{i1}, \ldots, c_{iv}\}$ be the pattern mined from subspace S_i. Then FCP_i can be pruned if (a)$\{r_{i1}, \ldots, r_{iu}\} \cap BRG_i = \emptyset$, or (b)$\exists r_x \in FRS_i$, such that $\forall c_{iy} \in \{c_{i1}, \ldots, c_{iv}\}, O_{x,iy} = 1$.

Proof: First, we prove that the FCP_i can be pruned off if either (a) or (b) hold. As for (a), $FCP_i \subseteq S_i$, hence $\{r_{i1}, \ldots, r_{iu}\} \subseteq (BRG_i \cup LRS_i)$. Since $\{r_{i1}, \ldots, r_{iu}\} \cap BRG_i = \emptyset$, thus $\{r_{i1}, \ldots, r_{iu}\} \subseteq LRS_i$. Hence, there must exist a latter subspace S_i' that $r_{i1} \in BRG_i'$, where r_{i1} is the first row of $\{r_{i1}, \ldots, r_{iu}\}$. Hence, $\{r_{i1}, \ldots, r_{iu}\} \subseteq (BRG_i' \cup LRS_i')$. Moreover, according to the projection rules, $\{c_{i1}, \ldots, c_{iv}\} \subseteq BCG_i'$. Hence, $FCP_i \subseteq S_i'$. Hence, FCP_i can be pruned off from S_i as either a redundancy or false drop.

As for (b), since $\exists r_x \in FRS_i$, such that $\forall c_{iy} \in \{c_{i1}, \ldots, c_{iv}\}, O_{x,iy} = 1$, $\exists FCP_x = \{r_x, r_{i1}, \ldots, r_{iu}\} \times \{c_{i1}, \ldots c_{iv}\}$, such that $FCP_i \subset FCP_x$. Hence, FCP_i can be pruned off from S_i as a false drop. □

We note that each subspace S_i can be independently mined without any knowledge of other subspaces. As such, all nodes can work in parallel when mining the allocated subspaces.

Lemma 6. Let O be the original space. Let $S_1, \ldots, S_t$ be the subspaces generated in phase 1 of *B-Miner*. Let $P_1, \ldots, P_t$ be the set of FCPs that are pruned from the corresponding subspaces in phase 2. Then, $MineFCP(S_i) - P_i \subset MineFCP(O)$.

Proof: We need to prove that if $\exists FCP_i \in MineFCP(S_i) - P_i$, FCP_i is the global distinct closed frequent pattern. This can be proved in two aspects. First, a subspace is not "global" for its FCP in that it drops off the *Former Row Set.* Since (b) ensures that no rows in its *Former Row Set* contain the FCP's full column set, the FCP is hence global closed. Second, since (a) ensures that a FCP contains at least one row from its *Base Row Group*, and the space partition method ensures that all latter subspaces do not contain such a row, the FCP is hence ensured global distinct, not appearing again in the former/latter subspaces. As a result, the FCPs generated are distinct and globally closed. □

Theorem 2. Let O be the original space. Let $S_1, \ldots, S_t$ be the subspaces generated in phase 1 of *B-Miner*. Let $P_1, \ldots, P_t$ be the set of FCPs that are pruned from the corresponding subspaces in phase 2. Then $MineFCP(O) = \cup_{i=1}^{t}(MineFCP(S_i) - P_i)$.

Proof: The proof follows from Lemma 4- 6. □

3.3.4 Parallel FCP Mining

As noted in the previous subsections, the progressive FCP mining framework can be easily adapted for parallel processing. In this section, we shall present the parallel FCP mining framework.

We use as our context a parallel environment that comprises a network of nodes (i.e., PCs) that are loosely integrated. This would be similar to work like Seti@Home[3] and Folding@Home[4]. Moreover, we assume that only a source node has the original dataset; in other words, we do not assume that the dataset is partitioned across all participating nodes. When the dataset needs to be mined, the source node will look for nodes to parallelize the mining process. Like traditional load-balanced query processing, our framework generates a large number of subspaces (larger than the number of nodes) and then allocates these subspaces to nodes to be mined concurrently and independently. It is essentially a straightforward adaptation of the progressive FCP mining framework, and it operates in three logical phases:

- *Task execution phase.* The task execution phase corresponds to the subspace generation phase of the progressive framework. Thus, the original space is partitioned into subspaces such that mining all the subspaces will lead to a superset of the answers. This phase can be done at the source node (in which case, the source node generates all the subspaces). Alternatively, we can parallelize this phase by exploiting more and more nodes to perform the partitioning: (a) the source node will generate t_1 subspaces; (b) these subspaces are then allocated to t_1 nodes (including the source); each of these t_1 nodes will further partition

[3] http://setiathome.berkeley.edu/

[4] http://folding.stanford.edu/

the allocated subspace into t_2 smaller subspaces which are then further distributed to t_2 nodes; (c) the above process is repeated until sufficient number of tasks/subspaces have been produced. For simplicity, in our experimental study, this task is accomplished by the source node (i.e., we do not parallelize this phase).

- *Task allocation phase.* In the second phase, the source node (which acts as a coordinator) will assign a subspace to each node to mine.
- *Task execution phase.* Finally, in the third phase, which is similar to the subspace mining phase, each node independently mines the allocated subspaces.

We note that the second and third phases operate iteratively: whenever a node completes processing its subspace, it will request the source node for another subspace. In this way, the scheme is also load-balancing.

Now, both *C-Miner* and *B-Miner* can be parallelized under the framework. There is, however, one issue to be addressed: in order for a node to be able to mine a subspace S_i independently, the pruning of false drops or redundant FCPs must be done without incurring any communication overhead between nodes. To ensure this, we need to disseminate the original dataset O to all participating nodes. This cost, fortunately, is inexpensive (in terms of response time) as it can be done concurrently while the data space is being partitioned. Moreover, only one copy per node is necessary even if multiple subspaces are being allocated to a node. In addition, for our real datasets, they are not big.

3.3.5 Time Complexity

The problem of mining maximal frequent itemsets, which is a subset problem of mining frequent closed patterns, has been proved to be NP-hard [57]. For the 2D dataset $O = R \times C$, where $|R| = N$, $|C| = M$, and η is the number of subspaces partitioned in the mining process, the time complexity of *C-Miner* and *B-Miner* is $O(2^{\frac{N}{\eta}} \times M)$, without applying any pruning strategy. By applying *minsup*, *minlen* and closeness constraints, the efficiency of *C-Miner* and *B-Miner* is improved greatly.

3.4 Experimental Results

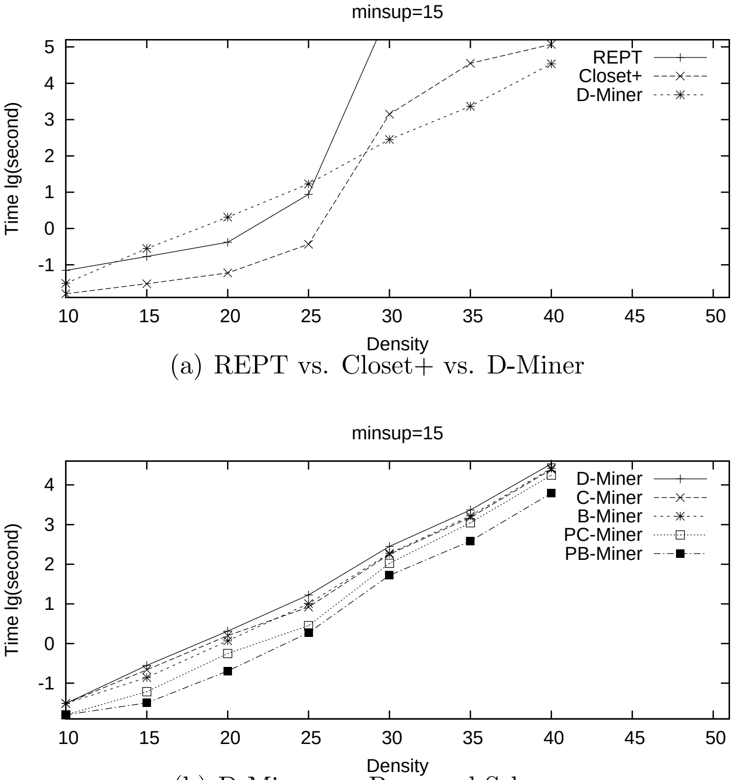
We have implemented *C-Miner* and *B-Miner*, and their parallel versions (denoted as *PC-Miner* and *PB-Miner* respectively) in C. For *C-Miner*, we employ CLUTO to group rows into clusters in its phase one. For both *C-Miner* and *B-Miner*, we adapt D-Miner [7] in phase two by incorporating the respective pruning strategies to mine the subspaces for the exact FCPs. We conducted a performance study to evaluate their efficiency against Closet+[5], REPT [12] and D-Miner. For our experiments, we use two real microarray datasets: the breast cancer dataset (BC) [6] and the prostate cancer dataset (PC) [7]. In such datasets, the rows represent clinical samples while the items represent the activity levels of genes/proteins in the samples. In the BC dataset, there are 78 tissue samples and each sample is described by the activity level of 24481 genes. In the PC dataset, there are 102 tissue samples each described by the activity level of 12600 genes. The BC and PC datasets are discretized by doing a equal-width partition for each column with 20 and 4 buckets respectively, resulting a dataset with

[5] The code is downloaded from http://illimine.cs.uiuc.edu/.

[6] http://www.rii.com/publications/default.htm

[7] http://www-genome.wi.mit.edu/mpr/prostate

density of 49.76% (i.e., 49.76% of the cells contain one, while the rest contain zero) and 49.18% accordingly. To study the effect of the proposed schemes on other factors (e.g., density, scalability), we also use synthetic datasets generated by the IBM data generator[8]. All the experiments are run on a Pentium 4 PC with 1 GB RAM. We have run a large number of experiments, and shall present only representative results here. The default number of processors for the parallel algorithms is 8.



(a) REPT vs. Closet+ vs. D-Miner

(b) D-Miner vs. Proposed Schemes

Figure 3.5: Variation of Density.

[8] The generator is available at http://www.cs.umbc.edu/c̃giannel/assoc_gen.html.

3.4.1 Varying Dataset Density

In the first set of experiments, we study the effects of dataset density on the execution time. We experiment on seven synthetic datasets generated by the IBM data generator with 50 rows, 500 columns, and density varying from 10% to 40%. We compare the performance of Closet+, REPT, D-Miner, *C-Miner* ($N_{cluster=5}$), and *B-Miner* ($GL = 1$), *PC-Miner* and *PB-Miner*. We set $minsup = 15$ and $minlen = 1$. Figure 3.5 shows the execution time (seconds in logarithm scale) of each algorithm. From Figure 3.5(a), we find that Closet+ and REPT are quicker than D-Miner when the density is below 25%, but become much slower than D-Miner when density is above 30%. That is, although Closet+ and REPT are efficient on sparse datasets, they lose their advantage on dense datasets compared with D-Miner. Thus, since our focus is on dense datasets, we will not discuss them any further. Instead, we shall compare our proposed schemes with D-Miner. Figure 3.5(b) shows that our proposed schemes *C-Miner* and *B-Miner* are much quicker than D-Miner, and *C-Miner* is slightly quicker than *B-Miner* on denser datasets. Moreover, the parallel versions can further reduce the processing time greatly. We also note that *PB-Miner* is more efficient than *PC-Miner*.

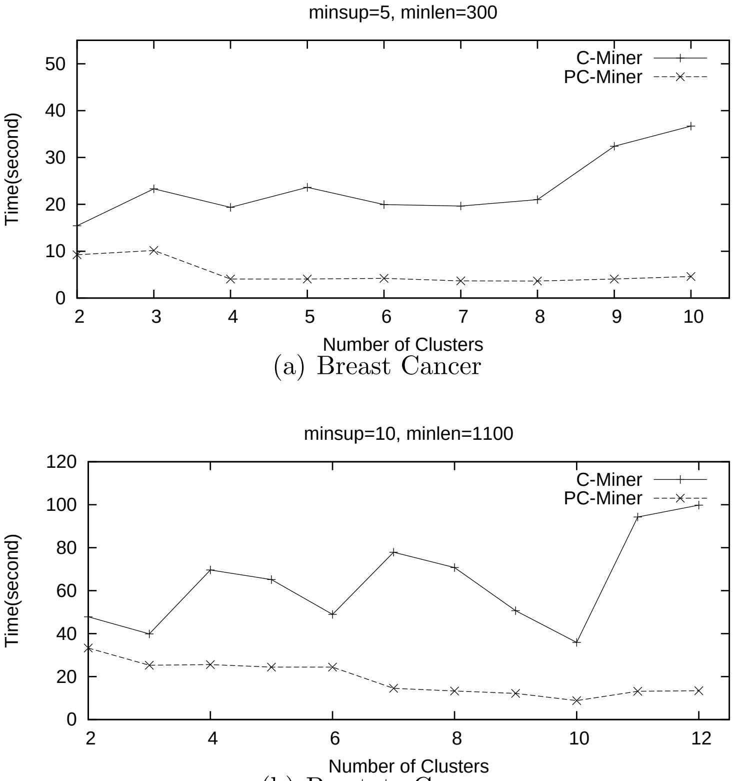
3.4.2 Experiments on Real Microarray Datasets

In the second set of experiments, we compare our proposed schemes against D-Miner on real microarray datasets by variation of *minsup* and *minlen* respectively. For *C-Miner*, *B-Miner* and their parallel versions, one of the key parameters is the number of subspaces which affects the execution time. For *C-Miner* and *PC-Miner*, the number of subspaces is controlled by the number of clusters. And for *B-Miner* and *PB-Miner*, the number of subspaces is controlled by the group length(GL). Hence, we begin by tuning the various algorithms - *C-Miner*, *B-Miner* and their parallel versions - on these two parameters.

Tuning the Proposed Schemes

We vary the number of clusters for *C-Miner* and *PC-Miner*. The results for the two cancer datasets are shown in Figure 3.6, where we set $minsup = 5$ and $minlen = 300$ for BC dataset, and $minsup = 10$ and $minlen = 1100$ for PC dataset respectively. The results show that there is a certain "optimal" cluster number for *C-Miner*. From the results, we find that more clusters lead to better load balancing in parallelism. Having more clusters, and hence more subspaces, may be beneficial as it facilitates load balancing. Having a smaller subspace may result in some "heavy-weight" mining subspace that dominates performance. Thus, in general, the more subspaces there are, the better the running time for parallelism. However, when the number of clusters increases to some point, the overall processing time keeps increasing and the advantage in parallelism is affected as well. When the number of clusters is very large, the number of subspaces becomes large. This means that generating the subspaces (in phase 1) becomes costly, and processing a large number of subspaces (in phase 2) is also costly.

As for the BC and PC datasets, *C-Miner* and *PC-Miner* keeps increasing when the number of clusters is above 9 and 11 respectively. Hence, we suggest choosing the number of clusters below the values. For BC dataset, the "optimal" value is 2 for *C-Miner* and 8 for *PC-Miner*. Users may choose the value according to whether they prefer a centralized or a parallelled scheme. As for the PC dataset, the "optimal" value is 10 for both *C-Miner* and *PC-Miner*. We shall use $N_{cluster} = 2$ for BC dataset and

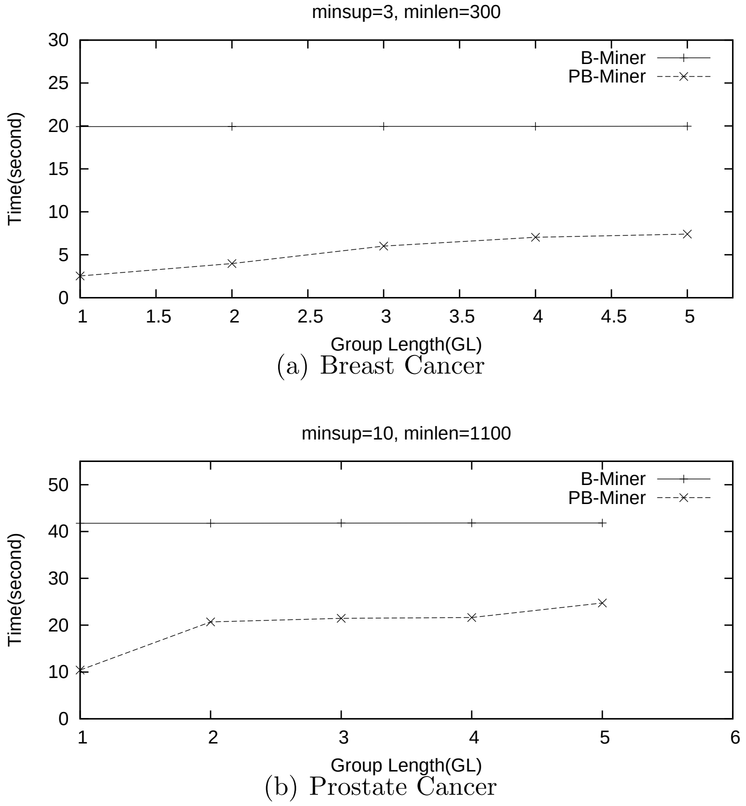


(a) Breast Cancer

(b) Prostate Cancer

Figure 3.6: Vary number of clusters (and subspaces).

$N_{cluster} = 10$ for PC dataset as the default when experimenting with these datasets in all subsequent studies.

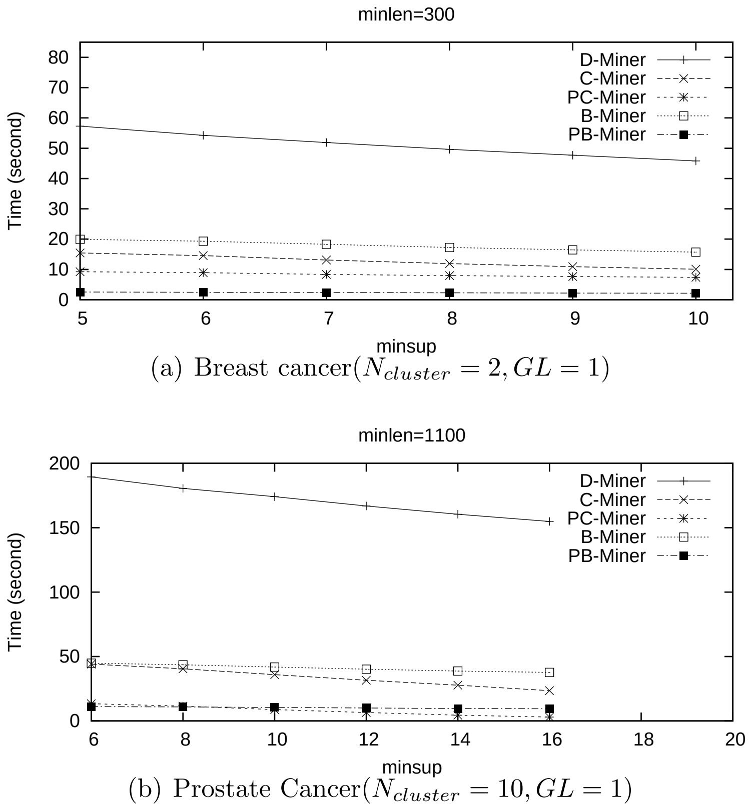


(a) Breast Cancer

(b) Prostate Cancer

Figure 3.7: Vary *Group Length* (*GL*) (and subspaces).

For *B-Miner* and *PB-Miner*, the number of subspaces is determined by the *Group Length* (*GL*). We vary *GL* from 1 to 5. The results for the two datasets are shown in Figure 3.7, where we set $minsup = 5$ and $minlen = 300$ for BC dataset, and $minsup = 10$ and $minlen = 1100$ for PC dataset respectively. As *GL* increases, the number of subspaces decreases - $GL = 1$ indicates largest number of subspaces (i.e., number of subspaces = number of rows) and $GL = number\ of\ rows$ indicates no partitioning (i.e., one single subspace). As shown in Figure 3.7, the execution time of both *B-Miner* and its parallel version *(PB-Miner)*, increases with the increase of GL. GL's effect on *B-Miner* is relatively small. To optimize the *PB-Miner*, we shall use $GL = 1$ as the default for the datasets.



(a) Breast cancer($N_{cluster} = 2, GL = 1$)

(b) Prostate Cancer($N_{cluster} = 10, GL = 1$)

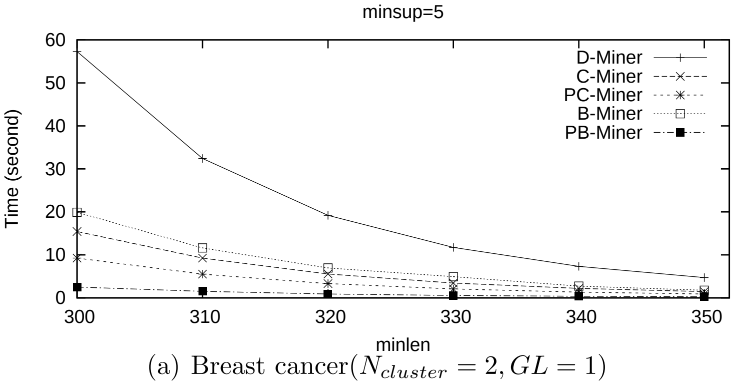
Figure 3.8: Variation of *minsup*.

Varying minsup and minlen

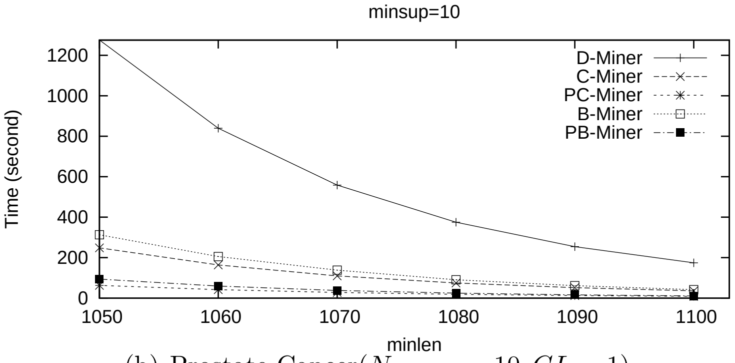
In this set of experiments, we study the effects of *minsup* and *minlen* on the execution time. We experiment on BC and PC datasets and compare the performance of D-Miner, *C-Miner*, *B-Miner*, *PC-Miner* and *PB-Miner*.

First, we set $minlen = 300$ and vary the $minsup$ from 5 to 10 for BC dataset;

and set *minlen* = 1100 and vary the *minsup* from 6 to 16 for PC dataset. The comparative results are presented in Figure 3.8.



(a) Breast cancer($N_{cluster} = 2, GL = 1$)



(b) Prostate Cancer($N_{cluster} = 10, GL = 1$)

Figure 3.9: Variation of *minlen*.

Second, we set *minsup* = 5 and vary the *minlen* from 300 to 350 for BC dataset; and set *minsup* = 10 and vary the *minlen* from 1050 to 1100 for PC dataset. The comparative results are shown in Figure 3.9.

The comparative results presented in Figure 3.8 and Figure 3.9 show clearly that the execution time decreases with increasing *minsup* and *minlen* values. Moreover, as in the previous experiments, *C-Miner*, *B-Miner* and their parallel versions outperformed D-Miner. *C-Miner* outperformed *B-Miner* for both datasets. For BC dataset, the parallel version of *C-Miner(PC-Miner)* is slightly slower than *PB-Miner*. This is due to the reason that the subspaces for *PC-Miner* is much fewer than those for *PB-Miner*, considering $N_{cluster} = 2$. As for PC dataset, since $N_{cluster} = 10$, *PC-Miner* has more subspaces such that the load is well balanced and hence it outperforms *PB-Miner*.

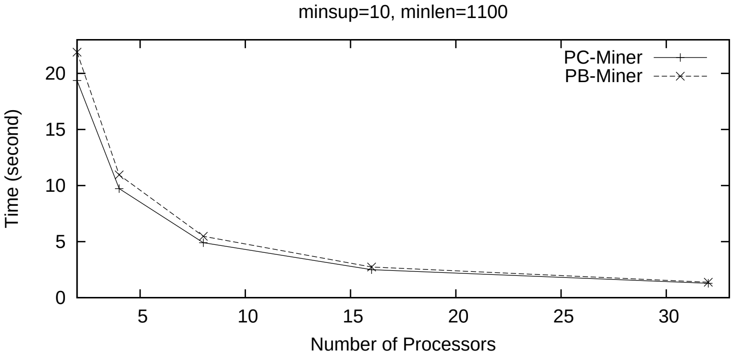
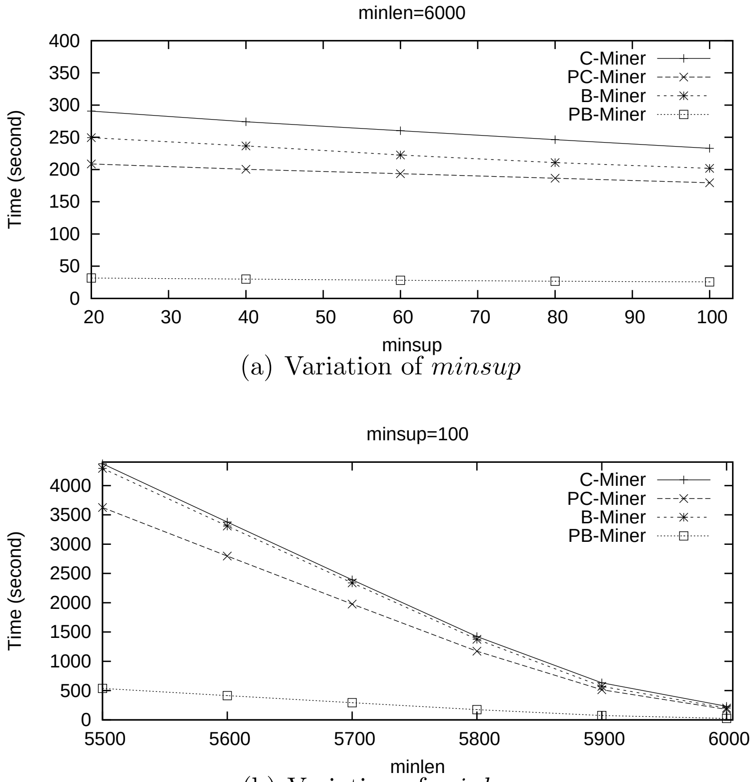


Figure 3.10: Vary Number of Processors.

3.4.3 Varying the number of processors

We also study the effects of processor number on the execution time of *PC-Miner* and *PB-Miner*. Since the results for both BC and PC datasets show similar relative performance, we only show the results of PC dataset. We set the $minsup = 10$, $minlen = 1100$, $N_{cluster} = 10$ for *PC-Miner* and $GL = 1$ for *PB-Miner*. Figure 3.10 shows the execution time of *PC-Miner* and *PB-Miner* with the variation of processor number. The execution time of both algorithms decreases with the increasing of processor number. From the experiments, we find that our schemes balance the workload very well.



(a) Variation of *minsup*

(b) Variation of *minlen*

Figure 3.11: Scalability.

3.4.4 Scalability

To study the scalability of our proposed schemes, we generate a synthetic dataset of 1000 rows, 100000 columns, and 10% in density using the IBM data generator. We set $N_{cluster} = 2$ and $GL = 1$ to optimize *C-Miner* and *B-Miner*, and vary the *minsup* and *minlen* for the experiments. The results are presented in Figure 3.11. From the results, we see that our proposed schemes can scale well for large volume datasets. *B-Miner* is slightly quicker than *C-Miner*. And *PB-Miner*'s load is better balanced due to more subspaces. As for D-Miner, it takes more than $30,000$ seconds for each execution, and hence are not shown in the figures.

Table 3.6: Sample of Known Co-regulated Genes from the FCPs.

M/G1 Boundary Regulated: CLN3(YAL040C),TEC1(YBR083W), SWI5(YDR146C), ASH1(YKL185W), SIC1(YLR079W), CTS1(YLR286C), CHS1(YNL192W), EGT2(YNL327W)
Late G1(SCB) Regulated: TIP1(YBR067C), CLN1(YMR199W), CLN2(YPL256C)
LateG1(MCB) Regulated: POL30(YBR088C), MCD1(YDL003W), CDC9(YDL164C), GIC2(YDR309C), SRS2/HPR5(YJL092W), RFA3(YJL173C), PRI2(YKL045W), CLB5(YPR120C), CDC45(YLR103C), RFA2(YNL312W), NIK1/HSL1(YKL101W)
S-phase Regulated: HTB2(YBL002W), HTA2(YBL003C), HHF1(YBR009C), HHT1(YBR010W), HTB1(YDR224C), HTA1(YDR225W), HHF2(YNL030W), HHT2(YNL031C)
S/G2-phase Regulated: NUM1(YDR150W), TIR1(YER011W), CWP1(YKL096W), CWP2(YKL096W-A)
G2/M-phase Regulated: MST2(YDR033W), SWI5(YDR146C), FAR1(YJL157C), ACE2(YLR131C)

3.4.5 Biological Significance

To test the biological significance of our proposed frameworks, we explore a known real Yeast gene expression dataset[9] with 2884 genes under 17 conditions. We preprocess the dataset by taking genes with expression values exceeding the average expression value under each condition as high expressed, noted as “1”, and low expressed noted as “0”, otherwise. Thus results in a (2884 × 17) binary matrix of 47.3% density.

We set the minimum support condition as 10 and minimum support gene as 1000 for the experiment. We get 6812 FCPs from which we identify some interesting co-regulated genes that have already been established in the literature [38, 8]. Table 3.6 shows a sample of such co-regulated genes identified from our results. Among the

[9] The data is downloaded from http://arep.med.harvard.edu/biclustering/yeast.matrix.

6812 FCPs, 664 FCPs fail to identify G2/M-phase Regulated Genes. 90% of the FCPs generated contain the whole six categories of known co-regulated genes.

3.5 Summary

In this chapter, we have proposed a novel framework for mining FCPs (2D co-attribute patterns) on dense datasets. The key idea is to partition the original datasets into smaller subspaces such that mining the subspaces will produce the same answers as mining from the original space. Based on this framework, we proposed two new algorithms, *C-Miner* and *B-Miner*. The two schemes adopt different partitioning and pruning strategies. We also show how the framework can facilitate parallel FCP mining in a straightforward manner. Our performance study showed that both schemes and their parallel versions are efficient and scalable. Moreover, we test the biological significance of our frameworks on known real Yeast microarray data and identify some interesting known co-regulated gene patterns.

Chapter 4

Mining Frequent Closed Cubes in 3D Datasets

4.1 Overview

While several efficient FCP mining algorithms have been studied in Chapter 3, those algorithms are all limited to 2D dataset analysis, for example, the *gene-time*, *gene-sample* biological datasets in microarray dataset analysis, and the *transaction-itemset* datasets in 'market basket' analysis. With recent advances in microarray technology, the expression levels of a set of genes under a set of samples can be measured simultaneously over a series of time points, which results in 3D *gene-sample-time* microarray data [32]. New patterns delivering *gene-sample-time* relationships are certainly more valuable in the study of gene pathways. Even in the traditional 'market-basket' analysis, it is not uncommon to have consumer information on a number of dimensions, e.g., *region-time-items* data that stores the sales of itemsets in certain locations over certain time periods. This trend motivates us to extend existing 2D frequent closed pattern analysis to 3D context. We refer to the frequent closed pattern in 3D context as *frequent closed cube* (FCC). Designing efficient algorithms to discover FCCs is the theme of this chapter.

Association analysis based on FCCs can deliver more interesting information in 3D context. Let us first take biological microarray datasets for example. Association analysis based on FCCs can reveal patterns about how the expression of one gene may be associated with the expression of a set of genes under a set of environments during a set of time points. Given such information, we can easily infer that the genes involved participate in some kind of gene networks. Moreover, such association rules can be used to relate the expression of genes to their cellular environments and time periods simultaneously. Such associations can help to detect cancer genes in different cancer developing stages, especially when cancer is caused by a set of genes acting together instead of a single gene. Like clustering, gene function can be inferred based on the other genes in such association rule. Next, we give an example in ‘market basket’ analysis. While the association analysis based on 2D frequent pattern represents a set of items that are likely to be purchased together in a set of transactions, a 3D FCC over a sales (region-time-items) dataset would represent a set of items that are likely to be purchased together in several locations over a set of time periods. Such information would enable suppliers to deploy their products to chains located at different places during certain periods where consumers share similar purchasing behaviors.

In this chapter, we tackle the problem of mining FCCs from 3D datasets. The FCCs deliver “close” relationships among three dimensions. That is, we identify the *maximum* patterns in a 3D context. The 3D pattern is maximum in that an increase in any dimension will cause a direct decrease in at least one of the other two dimensions; i.e., no further expansion in any dimension can be made on the pattern.

Our contributions are as follows. First, we introduce the notion of FCC and formally define it. Second, we propose two approaches to mine FCCs. The first approach is a three-phase framework, called *Representative Slice Mining* algorithm *(RSM)* that exploits 2D FCP mining algorithms to mine FCCs. The basic idea is to transform a 3D dataset into a set of 2D datasets, mine the 2D datasets using an existing 2D FCP mining algorithm, and then prune away any frequent cubes that are not closed. The second method is a novel scheme, called *CubeMiner*, that operates directly on the 3D dataset to mine FCCs. Third, we also show how *RSM* and *CubeMiner* can be easily extended to exploit parallelism. Finally, we have implemented *RSM* and *CubeMiner*, and conducted experiments on both real and synthetic datasets. To our knowledge, there has been no prior work that mine FCCs. We also show the biological significance of FCCs delivered from the real microarray datasets.

The rest of this chapter is organized as follows. In Section 4.2, we formally define the FCC mining problem. Section 4.3 presents the proposed *RSM* framework, while Section 4.4 presents the proposed *CubeMiner* algorithm. In Section 4.5, we show how *RSM* and *CubeMiner* can be extended to exploit parallelism. Section 4.7 reports experimental results on *RSM* and *CubeMiner*, and finally, we conclude in Section 4.8.

4.2 Preliminaries

We shall first define some notations that we will be using throughout this chapter, and then give the problem description.

Let $R = \{r_1, r_2, \ldots, r_n\}$ denote a set of rows, $C = \{c_1, c_2, \ldots, c_m\}$ denote a set of columns, and $H = \{h_1, h_2, \ldots, h_l\}$ denote a set of heights. Then a three-dimension dataset can be represented by a $l \times n \times m$ binary matrix $O = H \times R \times C = \{O_{k,i,j}\}$

with $k \in [1, l]$, $i \in [1, n]$ and $j \in [1, m]$. Each cell o_{kij} corresponds to the relationship among height h_k, row r_i, and column c_j. The value true (i.e., "1") denotes the relationship that any two dimensions are "simultaneously contained (S-contained)" in the third one.

Table 4.1 shows an example of a three-dimension dataset in Boolean context. In Table 4.1, h_1 and r_4 are S-contained in c_3 and c_5, denoted as $C(h_1 \times r_4) = \{c_3, c_5\}$; h_2 and c_5 are S-contained in r_1 and r_4, denoted as $R(h_2 \times c_5) = \{r_1, r_4\}$; r_2 and c_1 are S-contained in h_1 and h_3, denoted as $H(r_2 \times c_1) = \{h_1, h_3\}$.

Table 4.1: Example of Binary Data Context.

$H = h_1$

R/C	c_1	c_2	c_3	c_4	c_5
r_1	1	1	1	0	1
r_2	1	1	1	0	0
r_3	1	1	1	1	1
r_4	0	0	1	0	1

$H = h_2$

R/C	c_1	c_2	c_3	c_4	c_5
r_1	1	1	1	1	1
r_2	0	1	1	1	0
r_3	1	1	1	1	0
r_4	1	1	1	0	1

$H = h_3$

R/C	c_1	c_2	c_3	c_4	c_5
r_1	1	1	1	0	0
r_2	1	1	1	0	0
r_3	1	1	1	1	0
r_4	1	1	0	1	1

Definition 4.1 Height Support Set and H-Support: *Given a set of rows $R' \subseteq R$ and a set of columns $C' \subseteq C$, the maximal set of heights that simultaneously contain R' and C' is defined as the Height Support Set $H(R' \times C') \subseteq H$. The number of heights in $H(R' \times C')$ is defined as the H-Support of $(R' \times C')$, denoted as $|H(R' \times C')|$.*

For example, in Table 4.1, let $R' = \{r_1, r_2\}$ and $C' = \{c_1, c_2, c_3\}$, then $H(R' \times C') = \{h_1, h_3\}$ since both h_1 and h_3 simultaneously contain $\{r_1, r_2\}$ and $\{c_1, c_2, c_3\}$, and no other heights contain them simultaneously.

Definition 4.2 Row Support Set and R-Support: *Given a set of columns $C' \subseteq C$ and a set of heights $H' \subseteq H$, the maximal set of rows that simultaneously contain C' and H' is defined as the Row Support Set $R(C' \times H') \subseteq R$. The number of rows in $R(C' \times H')$ is defined as the R-Support of $(C' \times H')$, denoted as $|R(C' \times H')|$.*

For example, in Table 4.1, let $C' = \{c_1, c_2, c_3\}$ and $H' = \{h_1, h_3\}$, then $R(C' \times H') = \{r_1, r_2, r_3\}$ since r_1, r_2 and r_3 simultaneously contain $\{c_1, c_2, c_3\}$ and $\{h_1, h_3\}$, and no other rows contain them simultaneously.

Definition 4.3 Column Support Set and C-Support: *Given a set of rows $R' \subseteq R$ and a set of heights $H' \subseteq H$, the maximal set of columns that simultaneously contain R' and H' is defined as the Column Support Set $C(R' \times H') \subseteq C$. The number of columns in $C(R' \times H')$ is defined as the C-Support of $(R' \times H')$, denoted as $|C(R' \times H')|$.*

For example, in Table 4.1, let $R' = \{r_3, r_4\}$ and $H' = \{h_2, h_3\}$, then $C(R' \times H') = \{c_1, c_2\}$ since both c_1 and c_2 simultaneously contain $\{r_3, r_4\}$ and $\{h_2, h_3\}$, and no other columns contain them simultaneously.

Definition 4.4 Closed Cube: *Given a set of rows $R' \subseteq R$, a set of columns $C' \subseteq C$, and a set of heights $H' \subseteq H$, a cube $A = (H' \times R' \times C') \subseteq O$ is defined as a Closed Cube if (1) $R' = R(C' \times H')$; (2) $C' = C(R' \times H')$; and (3) $H' = H(R' \times C')$.* For clarity, $A = (H' \times R' \times C')$ is written as $A = (H', R', C')$. Moreover, conditions (1), (2) and (3) are referred to as “closed” in row set, column set and height set respectively. Intuitively, a closed cube is complete (with all ‘1’s inside) and maximal (no larger complete cubes contain it).

Definition 4.5 Frequent Closed Cube (FCC): *A cube $A = (H', R', C') \subseteq O$ is called a frequent closed cube if (1) the H-Support $|H(R' \times C')|$, R-Support $|R(H' \times C')|$, and C-Support $|C(R' \times H')|$ are higher than the minimum H-Support threshold (minH), minimum R-Support threshold (minR), and minimum C-Support threshold (minC) respectively; and (2) A is a closed cube.*

For example, given that $minH = minR = minC = 2$, the cube $A = \{h_1, h_3\} \times \{r_1, r_2, r_3\} \times \{c_1, c_2, c_3\}$ will be a frequent closed cube in Table 4.1. However, $A' = \{h_1, h_3\} \times \{r_2, r_3\} \times \{c_1, c_2, c_3\}$ is not a frequent closed cube in that $\{r_2, r_3\} \neq R(\{h_1, h_3\} \times \{c_1, c_2, c_3\}) = \{r_1, r_2, r_3\}$. For clarity, cube $A' = \{h_1, h_3\} \times \{r_2, r_3\} \times \{c_1, c_2, c_3\}$ is written as $A' = (h_1h_3, r_2r_3, c_1c_2c_3)$.

Problem Definition: Given a three-dimension dataset O, our problem is to discover all frequent closed cubes with respect to the user support thresholds *minH*, *minR*, and *minC*.

4.3 Representative Slice Mining

In this section, we propose a framework, called *Representative Slice Mining (RSM)*, to mine FCCs. Under this framework, any 2D FCP mining algorithms can be adapted to work on the 3D dataset. This framework is based on the idea that the 3D dataset $O = H \times R \times C$ can be presented as $O = H \times Slice_{R \times C}$. Hence, any dimension such as H set can be enumerated first, which results in all possible combinations of slices. Then on each combination of slices, 2D FCP algorithms can be applied on the other two dimensions such as R and C. Finally, a post-processing strategy is applied on the results to remove unclosed cubes due to the first enumerated dimension H. Based on this idea, we divide the *RSM* framework into three phases as shown in Algorithm 1. In phase 1, representative slice is generated based on one-dimension enumeration and slices combination; in phase 2, any 2D frequent closed pattern mining algorithm can be applied to mine 2D FCPs on each representative slice; in phase 3, a post-pruning scheme is applied to remove FCCs unclosed in the enumerated dimension. We shall present the details of the three phases below, before discussing the correctness of the scheme.

Algorithm 1 *RSM* Framework

```
Global variables: $H$ the set of heights, $R$ the set of rows, $C$ the set of columns,
  monotonic constraints $minH$, $minR$, and $minC$ on $H$, $R$, $C$ respectively. $\alpha$ the
  set of height subsets, $\beta$ the set of representative slices, $\gamma$ the set of 2D FCPs. Let
  $MineFCP(\beta)$ denote the algorithm to mine the set of 2D FCPs for a slice $\beta$.
Input: 3D Matrix $O$ with $l$ heights, $n$ rows and $m$ columns.
Output: $\xi$ the set of FCCs.
Phase 1: Representative Slice Generation
$\alpha \leftarrow \emptyset$;
while $|EnumerateSubset(H)| >= minH$ do
    $\alpha \leftarrow \alpha \cup EnumerateSubset(H)$;
end while
$\beta \leftarrow SliceCombine(\alpha)$;
Phase 2: 2D FCP Generation
$\gamma \leftarrow MineFCP(\beta)$;
Phase 3: Post-Pruning
$\xi \leftarrow PostPruning(\gamma)$;
```

4.3.1 Representative Slice Generation

In phase 1, we first take the height dimension H as our *base* dimension[1], and enumerate set $H = \{h_1, h_2, \ldots, h_l\}$ to get all subsets of H (denoted H') such that

[1] Note that we can pick any of the dimensions as the base dimension. In fact, as we shall see, because the base dimension has to be enumerated over all combinations of its values, picking the dimension that has the smallest number of values is a good heuristic. WLOG, we shall use the height dimension for our discussion.

$|H'| >= minH$. Given the dataset in Table 4.1 for example, let $minH = 2$, we will get the subsets $\{h_1, h_2\}$, $\{h_1, h_2, h_3\}$, $\{h_1, h_3\}$, and $\{h_2, h_3\}$.

Second, slices within the same subset are combined to form a new representative slice (RS). Given a 3D dataset $O = H \times R \times C = \{O_{k,i,j}\}$ with $k \in [1, l]$, $i \in [1, n]$ and $j \in [1, m]$, and let $H' = \{h_1, \ldots, h_x\}$ be the subset to be combined. Then the RS of H' can be represented as a $n \times m$ matrix such that $\forall O'_{i,j} \in RS, O'_{i,j} = \sum_{k=1}^{x} \cap O_{k,i,j}$ where $i \in [1, n]$ and $j \in [1, m]$. That is, the cell value of the representative slice is 1 only when all of its make-up values are 1; otherwise, the cell value is 0. And we say that the heights in H' "contribute to" the RS of H'. The 2nd column of Table 4.2 shows the representative slices of the above example.

Table 4.2: *RSM* Example ($minH = minR = minC = 2$).

Height Set	Representative Slices	2D FCPs	3D FCCs
h_2, h_3	11100 01100 11110 11001	$r_1r_3 : c_1c_2c_3, 2 : 3$ $r_1r_3r_4 : c_1c_2, 3 : 2$ $r_1r_2r_3 : c_2c_3, 3 : 2$	$h_2h_3 : r_1r_3r_4 : c_1c_2, 2 : 3 : 2$
h_1, h_3	11100 11100 11110 00001	$r_1r_2r_3 : c_1c_2c_3, 3 : 3$	$h_1h_3 : r_1r_2r_3 : c_1c_2c_3, 2 : 3 : 3$
h_1, h_2	11101 01100 11110 00101	$r_1r_4 : c_3c_5, 2 : 2$ $r_1r_3 : c_1c_2c_3, 2 : 3$ $r_1r_2r_3 : c_2c_3, 3 : 2$	$h_1h_2 : r_1r_4 : c_3c_5, 2 : 2 : 2$
h_1, h_2, h_3	11100 01100 11110 00001	$r_1r_3 : c_1c_2c_3, 2 : 3$ $r_1r_2r_3 : c_2c_3, 3 : 2$	$h_1h_2h_3 : r_1r_3 : c_1c_2c_3, 3 : 2 : 3$ $h_1h_2h_3 : r_1r_2r_3 : c_2c_3, 3 : 3 : 2$

4.3.2 2D FCP Generation

In phase 2, any existing FCP mining algorithm can be applied on each representative slice to mine 2D FCPs based on dimensions R and C. In our experiments, we adopted D-Miner [7] as it has been shown to be efficient on relatively dense datasets with long patterns. After mining, we will have a set of 2D FCPs for R and C dimensions. For our running example, the FCPs are shown in the 3rd column of Table 4.2.

4.3.3 3D FCC Generation by Post-pruning

In phase 3, 3D frequent patterns are generated by combining each 2D FCP with the heights contributing to its representative slice. However, not all those 3D frequent patterns are FCCs. Some of them are not closed in the height set and should be pruned off. For example, in Table 4.2, after combining the first 2D FCP "$r_1r_3 : c_1c_2c_3, 2 : 3$" with the contributing heights "h_2, h_3", a 3D frequent pattern "$h_2h_3 : r_1r_3 : c_1c_2c_3, 2 : 2 : 3$" is generated. This 3D frequent pattern is not a FCC in that it is unclosed in the height set and has a superset "$h_1h_2h_3 : r_1r_3 : c_1c_2c_3, 3 : 2 : 3$"(the 4th FCC in the 4th Column of Table 4.2). That is, the 2D FCP is not only contained in slices h_2 and h_3, but also contained in slice h_1.

To remove all unclosed 3D frequent closed patterns, we develop a post-pruning strategy based on Lemma 7. If a 2D FCP is contained in other height slices besides its contributing height slices, it is unclosed and hence can be removed; otherwise, it is retained.

Lemma 7. Post-pruning Strategy: Let $O' = H' \times R' \times C'$ be a 3D frequent pattern and H be the whole height set. If $\exists H'' \in (H \setminus H')$ such that $\forall h_k \in H'', \forall r_i \in R', \forall c_j \in C', O_{k,i,j} = 1$, O' is unclosed in the height set and can be pruned off; otherwise, O' is retained.

Proof: $\exists H'' \in (H \setminus H')$ such that $\forall h_k \in H'', \forall r_i \in R', \forall c_j \in C', O_{k,i,j} = 1$. So, there exists $O_s = ((H'' \cup H') \times R' \times C')$, which is the superset of $O' = (H' \times R' \times C')$. Hence, O' is not closed in the height set, which contradicts the condition (3) of Closed Cube definition. That is, O' is not a closed cube and should be pruned off. □

In the post pruning process, not all relative cells in all non-contributing slices are checked. As shown in Algorithm 2, during each slice checking, the column checking loop (from line 12 to 17) is terminated whenever a cell with value '0' is detected, which directly leads to the termination of the row checking loop (from lines 10 to 22). That is, any one cell with value '0' can stop one slice checking. And if we detect that any slice passes the column and row checking loops (all relative cells value '1') without early termination, the whole slice checking loop (from lines 7 to 28) can be terminated in that the pattern is already confirmed to be unclosed. The strategy of Algorithm 2 ensures that we finish the close checking as early as possible. For the example in Table 4.2, after the post-pruning process, the resulting FCCs are shown in the 4th column.

4.3.4 Correctness

Theorem 3 shows that *RSM* can correctly generate all and only all FCCs.

Theorem 3. Let $FCCs$ be the set of frequent closed cubes of a 3D dataset. Let ξ denote the resultant frequent closed cubes obtained from running RSM on the dataset. Then $FCCs = \xi$. In other words, RSM correctly generates all and only all FCCs.

Proof: Let $MineFCP(RS)$ denote the 2D FCP mining algorithm on slice RS. First, we prove that $FCCs \subseteq \xi$. Let δ be the set of unclosed 3D frequent patterns removed

Algorithm 2 *RSM* Post Pruning

```
Input: 3D Pattern Set $\gamma$.
Output: 3D FCC Set $\xi$.
for $a = 1$ upto $|\gamma|$ do
    $(H', R', C') \leftarrow \gamma[a]$;
    $flag_1 \leftarrow 1$;
    for $k = 1$ upto $|H|$ do
        if $h_k \in (H \setminus H')$ then
            $flag_2 \leftarrow 1$;
            for $i = 1$ upto $|R|$ do
                if $r_i \in R'$ then
                    for $j = 1$ upto $|C|$ do
                        if $c_j \in C'$ and $O_{k,i,j} = 0$ then
                            $flag_2 \leftarrow 0$;
                            break;
                        end if
                    end for
                end if
                if $flag_2 = 0$ then
                    break;
                end if
            end for
            if $flag_2 = 1$ then
                $flag_1 \leftarrow 0$;
                break;
            end if
        end if
    end for
    if $flag_1 = 0$ then
        $\gamma \leftarrow \gamma \setminus \gamma[a]$;
    end if
end for
return $\gamma$;
```

by the post-pruning strategy. Given any FCC $O' = H' \times R' \times C'$, then there must exist a representative slice $RS_{H'}$ such that H' contributes to $RS_{H'}$. That is, $(R' \times C') \subseteq RS_{H'}$. Since $R' \times C'$ is closed for H', $(R' \times C') \subseteq MineFCP(RS_{H'})$. Hence, $O' \in (\xi \cup \delta)$. As proved in Lemma 7, the post-pruning strategy only removes unclosed 3D frequent patterns, so $O' \notin \delta$. Thus, $O' \in \xi$. Hence, we conclude that $FCCs \subseteq \xi$.

Next, we prove $\xi \subseteq FCCs$ by contradiction. Assume there exists a 3D pattern $O' \in \xi$ but $O' \notin FCCs$. Then O' is either not satisfied by monotonic support constraints or not closed. Suppose that $O' = H' \times R' \times C'$ does not satisfy $minH$ threshold, then $RS_{H'}$ will be pruned off during subset enumeration, and O' will not be generated. Suppose that O' does not satisfy $minR$ or $minC$ threshold, then $(R' \times C')$ of O' will be pruned off during 2D FCP generation, and O' will not be generated. This is contrary to the assumption. Hence, we gather that O' satisfies monotonic support constraints but it is not closed.

Suppose that O' is not closed in the H set, then there exists a closed FCC $O'' = (H' \cup H_a) \times R' \times C'$ such that $\forall h_k \in H_a, r_i \in R', c_j \in C', O_{k,i,j} = 1$, where $H_a \in (H \setminus H')$. Hence, in the post-pruning process, O' is pruned off, which is contrary to the assumption that $O' \in \xi$. Thus, we conclude that O' is closed in the H set.

Suppose that O' is not closed in the R set, then there exists a closed FCC $O'' = H' \times (R' \cup R_a) \times C'$ such that $\forall h_k \in H', r_i \in (R' \cup R_a), c_j \in C', O_{k,i,j} = 1$, where $R_a \subseteq (R \setminus R')$. Hence, $((R' \cup R_a) \times C') \subseteq RS_{H'}$. Then $((R' \cup R_a) \times C') \subseteq MineFCP(RS_{H'})$ and $R' \times C'$ is pruned off by the 2D FCP mining algorithm in that it is unclosed in the row set. Hence, O' cannot be generated, which is contrary to the assumption that $O' \in \xi$. Thus, we conclude that O' is closed in the R set. Using the same logic, we can prove that O' is closed in the C set.

Now that we conclude that O' is closed and satisfies all monotonic constraints. Hence, $O' \in FCCs$ and our assumption that there exists a 3D pattern $O' \in \xi$ but $O' \notin FCCs$ is wrong. That is, $\xi \subseteq FCCs$. So, our *RSM* framework for mining $FCCs$ is correct in that $\xi = FCCs$. □

4.4 *CubeMiner*

While *RSM* has the advantage that it can reuse existing FCP mining algorithms, the number of 2D slices could be large. In this section, we present a novel approach that mine FCCs directly from the 3D dataset. We shall first present the principle behind our proposed *CubeMiner* scheme. Then, we will look at the algorithm, and finally we shall show the correctness of *CubeMiner*.

4.4.1 *CubeMiner* Principle

CubeMiner is a novel algorithm for mining FCC (H', R', C') under constraints. It builds the sets H', R', and C' and uses monotonic support threshold constraints simultaneously on H, R, and C to reduce the search space. A FCC indicates that all its heights, rows, and columns are in "S-contained" relation. Since we are to identify maximal cubes with all its cells valued "1". If we could remove off useless "0s" from the original whole data cube without changing the forms of the rest cubes, we would narrow the search space greatly.

Figure 4.1 illustrates the principle of *CubeMiner*. Let cube O represent the whole dataset and the left-corner cube O' inside O represent the useless "0-zone" to be removed. From the surface of cube O', three planes α, β and γ are derived. Those

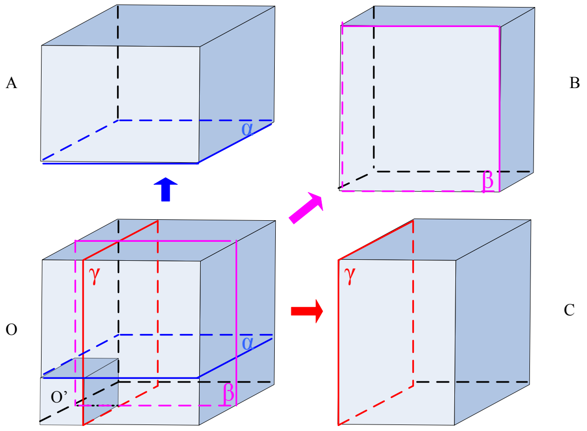


Figure 4.1: *CubeMiner* Principle.

three planes could split cube O into three pieces: upper-cube A, back-cube B and right-cube C. And the equation $A \cup B \cup C = O \setminus O'$ is satisfied. In any of the three pieces A, B, C, there may still exist "0-zones". The same splitting principle can be applied until all "0-zones" are removed off. We try to remove as many "0s" as possible in each splitting. While scanning in the data, "0s" are summarized together on the largest dimension for efficiency.

We use Z to denote a set of cell groups which are partitions of the false values (i.e., "0") of the boolean matrix. An element $(W, X, Y) \in Z$ is called a "cutter" if $\forall h_k \in W$, $\forall r_i \in X$, and $\forall c_j \in Y$, $O_{k,i,j} = 0$. And we call W, X, Y the left atom, middle atom, and right atom of cutter (W, X, Y) respectively. We summarize the "0" cells row by row, hence, Z contains as many cutters as rows in all height slices of the 3D data matrix. Each cutter is composed of the cells valued by 0 in the row. Table 4.3 shows the 10 cutters of the matrix in Table 4.1. The cutters are sorted by ascending order of left atom first and middle atom second.

CubeMiner starts with the whole dataset $O(H, R, C)$ and then splits it recursively using the cutters of Z until all cutters in Z are used and consequently all cells in each

Table 4.3: Z(cutter set).

W, X, Y
h_1, r_1, c_4
h_1, r_2, c_4c_5
$h_1, r_4, c_1c_2c_4$
h_2, r_2, c_1c_5
h_2, r_3, c_5
h_2, r_4, c_4
h_3, r_1, c_4c_5
h_3, r_2, c_4c_5
h_3, r_3, c_5
h_3, r_4, c_3

resulting cube have the value 1. A cutter (W, X, Y) in Z is used to cut a cube (H', R', C') if $W \cap H' \neq \emptyset$, $X \cap R' \neq \emptyset$, and $Y \cap C' \neq \emptyset$. In this case, we say that the cutter is "applicable" to the cube. By convention, we define the left son of (H', R', C') by $(H' \setminus W, R', C')$, the middle son by $(H', R' \setminus X, C')$ and the right son by $(H', R', C' \setminus Y)$. Recursive splitting leads to all FCCs, but also some non-maximal unclosed cubes. Pruning Strategies need to be applied to ensure that we obtain all FCCs and only the FCCs. We shall consider how to develop such pruning strategies. Figure 4.2 shows the tree generated from the 3D matrix in Table 4.1.

From Figure 4.2, we see that the 10 cutters in Table 4.3 split the original dataset into the resulting leaves in 10 steps (levels). We define the steps from the root to a node as the node's "path". Each node is split into three new nodes in the next level if the cutter is applicable. We only keep and show nodes satisfying support thresholds (given $minH = minR = minC = 2$) due to space limitation. However, in each level, not all nodes generated are useful for further splitting. There are four categories of useless nodes:

(a) Left son from a middle/right branch by the cutter whose left atom has cut

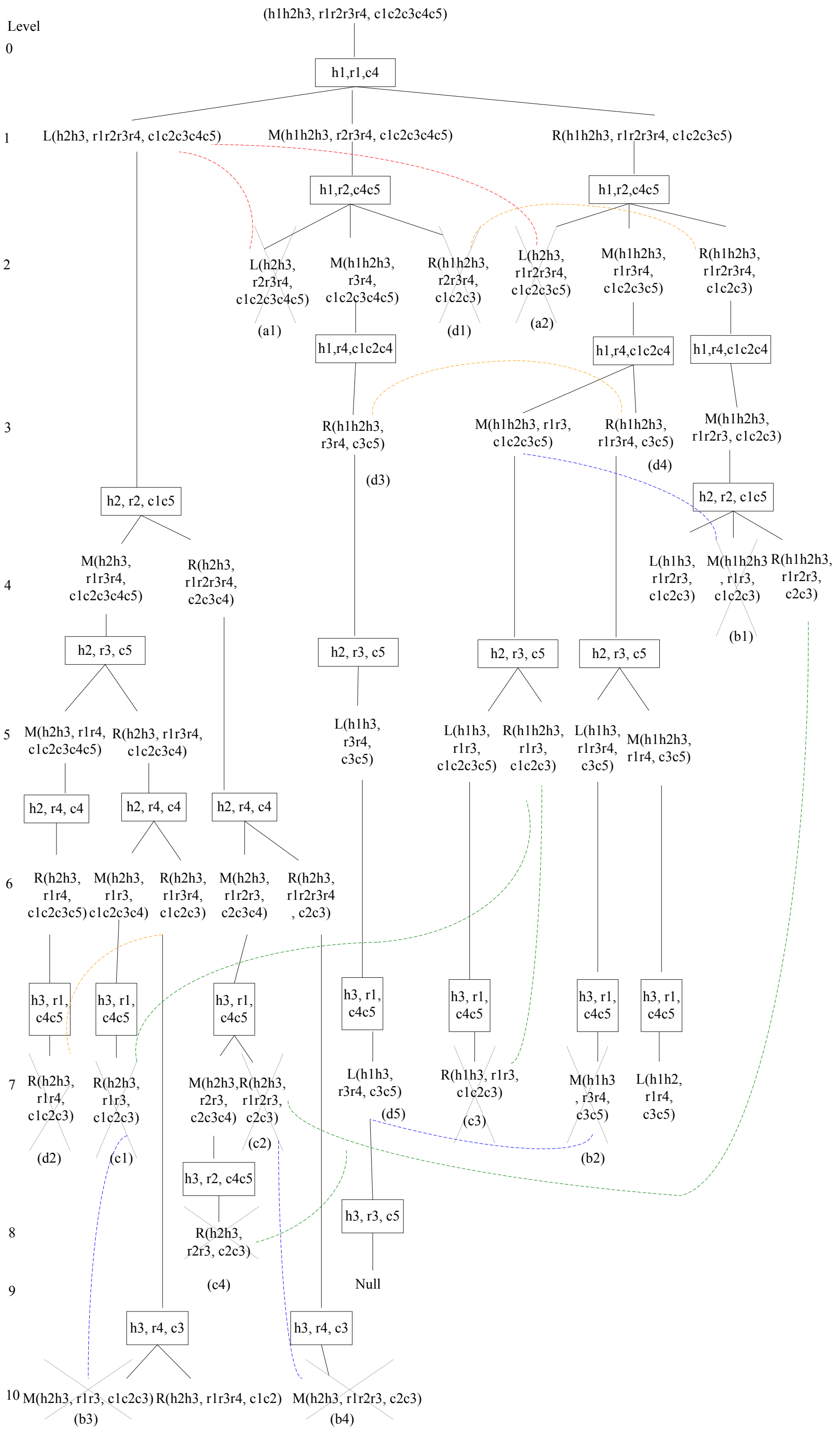


Figure 4.2: FCC Mining Tree.

the node's path before. For example, the left atom h_1 of cutter (h_1, r_2, c_4c_5) has already cut the paths of left sons $L(h_2h_3, r_2r_3r_4, c_1c_2c_3c_4c_5)$ (a_1 in Level 2) and $L(h_2h_3, r_1r_2r_3r_4, c_1c_2c_3c_5)$ (a_2 in Level 2) in Level 2. a_1 from the middle branch is unclosed in row set and a_2 from the right branch is unclosed in column set. They are to be pruned off as the subsets of node $L(h_2h_3, r_1r_2r_3r_4, c_1c_2c_3c_4c_5)$ (1st node in Level 1).

(b) Middle son from a right branch by the cutter whose middle atom has cut the node's path before. For example, the middle atom r_2 of cutter (h_2, r_2, c_1c_5) has already cut the path of middle son $M(h_1h_2h_3, r_1r_3, c_1c_2c_3)$ (b_1 in Level 4). This middle son is unclosed in column set and should be pruned off as the subset of node $M(h_1h_2h_3, r_1r_3, c_1c_2c_3c_5)$ (2nd node in Level 3). Middle sons b_2, b_3 and b_4 are all in such cases: they are either duplicates or subsets of other nodes.

(c) Nodes that are unclosed in height set. For example, node $R(h_2h_3, r_1r_3, c_1c_2c_3)$ (c_1 in Level 7) is unclosed in height set because there exists its superset node $R(h_1h_2h_3, r_1r_3, c_1c_2c_3)$ (5th node in Level 5). Such nodes should be pruned off to ensure closure in height set. Nodes c_2, c_3, c_4 are all such examples.

(d) Nodes that are unclosed in row set. For example, node $R(h_1h_2h_3, r_2r_3r_4, c_1c_2c_3)$ (d_1 in Level 2) is unclosed in row set because there exists its superset node $R(h_1h_2h_3, r_1 r_2r_3r_4, c_1c_2c_3)$ (6th node in Level 2). Such nodes should be pruned off to ensure closure in row set. Node $R(h_2h_3, r_1r_4, c_1c_2c_3)$ (d_2 in Level 7) is also one such example to be pruned off as it is not closed due to row r_3. Note that there exists some nodes that are closed in row set although they may have a temporary superset node in the processing. For example, node $R(h_1h_2h_3, r_3r_4, c_3c_5)$ (d_3 in Level 3) has a temporary superset node $R(h_1h_2h_3, r_1r_3r_4, c_3c_5)$ (d_4 in Level 3). Though node d_3 appears to be temporarily 'unclosed' due to row r_1, we detect that after applying a later cutter (h_3, r_1, c_4c_5) in level 7, node d_4 loses its superset status, and d_3's offspring $L(h_1h_3, r_3r_4, c_3c_5)$ (d_5 in Level 7) just serves as a reason to remove the middle son $M(h_1h_3, r_3r_4, c_3c_5)$ (b_2, an offspring of d_4) safely. Hence, such row set nodes which are temporary unclosed during processing are retained in that they are row set closed in the whole scenario.

To remove useless nodes of (a) and (b) types, we maintain two sets $TL = \{W_1, W_2, \ldots, W_p\}, TM = \{X_1, X_2, \ldots, X_q\}$ in each node to keep track of the left and middle atoms of cutters that cut its path. And based on the two sets, we develop Left Track Checking in Lemma 8 and Middle Track Checking in Lemma 9. In the initial status, $TL = TM = \emptyset$ for the root. Since only left sons from a middle/right branch need to be checked, TL set is updated only on a newly generated middle/right son. Similarly, since only middle sons from a right branch need to be checked, TM set is updated only on a newly generated right son. We shall denote the TL (and TM) set of node O as TL_O (and TM_O).

Lemma 8. Left Track Checking: Let $L = (H' \setminus W, R', C')$ be the left son of node $O' = (H', R', C')$ by cutter $z = (W, X, Y)$. If $W \cap TL_{O'} \neq \emptyset$, L can be pruned off.

Proof: Since $W \cap TL_{O'} \neq \emptyset$, $W \subseteq TL_{O'}$, hence $\exists z' = (W, X', Y') \in Z$ cuts O''s ancestor $O'_a = (H'_a, R'_a, C'_a)$. Let $O_l = (H_l, R_l, C_l)$ be the left sibling of O'_a by cutter z'. Then, either (1) $H_l = H'_a \setminus W, R'_a = R_l \setminus X' \subset R_l, C'_a = C_l$ or (2) $H_l = H'_a \setminus W, R'_a = R_l, C'_a = C_l \setminus Y' \subset C_l$. Since cutters between z' and z are all with left item W, which are not applicable to O_l, O_l remains unchanged after all z' to z cuttings and $H' \setminus W = H'_a \setminus W = H_l, R' \subseteq R_a, C' \subseteq C'_a$. So, in both (1) and (2), we can draw the conclusion that $L \subset O_l$. Hence, L can be pruned off. □

For example, in Figure 4.2, the left son $L(h_2h_3, r_2r_3r_4, c_1c_2c_3c_4c_5)$ (a1 in level 2) of parent $P(h_1h_2h_3, r_2r_3r_4, c_1c_2c_3c_4c_5)$ (2nd node in level 1) by cutter (h_1, r_2, c_4c_5) is pruned off in that $W \cap TL_p = \{h_1\} \neq \emptyset$.

Lemma 9. Middle Track Checking: Let $M = (H', R' \setminus X, C')$ be the middle son of node $O' = (H', R', C')$ by cutter $z = (W, X, Y)$. If $X \cap TM_{O'} \neq \emptyset$, M can be pruned off.

Proof: Since $X \cap TM_{O'} \neq \emptyset$, $X \subseteq TM_{O'}$, hence $\exists z' = (W', X, Y') \in Z$ cuts O''s ancestor $O'_a = (H'_a, R'_a, C'_a)$. Let $O_m = (H_m, R_m, C_m)$ be the middle sibling of O'_a by cutter z'. Then, we get $H_m = H'_a, R_m = R'_a \setminus X, C'_a = C_m \setminus Y' \subset C_m$. Hence, after the application of cutters between z' and z, the offspring of O_m, say $O'_m = (H'_m, R'_m, C'_m)$, satisfies the condition that $H'_m = H', R'_m = R' \setminus X, C' \subseteq C'_m$. Since z is not applicable to O'_m due to $R'_m \cap X = \emptyset$, O'_m remains unchanged after z cutting, and $M \subseteq O'_m$. Hence, M can be pruned off. □

For example, in Figure 4.2, the middle son $M(h_1h_2h_3, r_1r_3, c_1c_2c_3)$ (b1 in level 4) of parent $P(h_1h_2h_3, r_1r_2r_3, c_1c_2c_3)$ (4th node in level 3) by cutter (h_2, r_2, c_1c_5) is pruned off in that $X \cap TM_p = \{r_2\} \neq \emptyset$.

To remove useless nodes of (c) and (d) types, we develop Close Height Set Checking in Lemma 10 and Close Row Set Checking in Lemma 11.

Lemma 10. Close Height Set Checking: Let $O'' = (H'', R'', C'')$ be the middle/right son of node O' and Z be the whole cutter set. If $\exists h_w \in (H \setminus H'')$ (H is the full height set of O) such that $\forall(\{h_w\}, \{r_x\}, C_y) \in Z$ where $r_x \in R''$, $C'' \cap C_y = \emptyset$, then O'' is unclosed in the height set and can be pruned off. Since the left son never satisfies the condition, only the middle and right sons need this checking.

Proof: $\exists h_w \in (H \setminus H'')$ such that $\forall(\{h_w\}, \{r_x\}, C_y) \in Z$ where $r_x \in R''$, $C'' \cap C_y =$

$\emptyset$, that is, $\forall O_{w,x,y} \in (\{h_w\}, R'', C''), O_{w,x,y} = 1$. So, there exists $O_s = (H'' \cup \{h_w\}, R'', C'')$, which is the superset of $O'' = (H'', R'', C'')$. Hence, O'' is not closed in the height set and can be pruned off. □

For example, in Figure 4.2, node $R(h_2h_3, r_1r_2r_3, c_2c_3)$ (c2 in level 7) is not closed in the height set because there is $h_1 \in (H \setminus \{h_2, h_3\})$ such that for cutters (h_1, r_1, c_4) and (h_1, r_2, c_4c_5), $\{c_2, c_3\} \cap \{c_4\} = \emptyset$ and $\{c_2, c_3\} \cap \{c_4c_5\} = \emptyset$. And we find c2's superset in node $R(h_1h_2h_3, r_1r_2r_3, c_2c_3)$ (5th node in level 4).

Lemma 11. Close Row Set Checking: Let $O'' = (H'', R'', C'')$ be the left/right son of node O' and Z be the whole cutter set. If $\exists r_x \in (R \setminus R'')$ (R is the full row set of O) such that $\forall(\{h_w\}, \{r_x\}, C_y) \in Z$ where $h_w \in H''$, $C'' \cap C_y = \emptyset$, then O'' is unclosed in the row set and can be pruned off. Since the middle son never satisfies the condition, only the left and right sons need this checking.

Proof:$\exists r_x \in (R \setminus R'')$ such that $\forall(\{h_w\}, \{r_x\}, C_y) \in Z$ where $h_w \in H''$, $C'' \cap C_y = \emptyset$, that is, $\forall O_{w,x,y} \in (H'', \{r_x\}, C''), O_{w,x,y} = 1$. So, there exists $O_s = (H'', R'' \cup \{r_x\}, C'')$, which is the superset of $O'' = (H'', R'', C'')$. Hence, O'' is not closed in the row set and can be pruned off. □

For example, in Figure 4.2, node $R(h_2h_3, r_1r_4, c_1c_2c_3)$ (d2 in level 7) is not closed in the row set because $\exists r_3 \in (R \setminus \{r_1, r_4\})$ such that for cutters (h_2, r_3, c_5) and (h_3, r_3, c_5), $\{c_1, c_2, c_3\} \cap \{c_5\} = \emptyset$. And we find d2's superset in node $R(h_2h_3, r_1r_3r_4, c_1c_2c_3)$ (3th node in level 6).

4.4.2 Algorithm *CubeMiner*

We are now ready to present *CubeMiner* algorithmically. *CubeMiner* is a depth-first method to mine FCCs. Algorithm 3 contains the pseudo-code of *CubeMiner*. First, the left/middle track set TL/TM is initialized with empty set and the set Z of cutters is computed, and then the recursive function *cut*() in Algorithm 6 is called.

Algorithm 3 *CubeMiner*

```
CubeMiner()
Global variables: $H$ the set of heights, $R$ the set of rows, $C$ the set of columns,
  monotonic constraints $minH$, $minR$, and $minC$ on $H$, $R$, $C$ respectively.
Input: 3D Matrix $O$ with $l$ heights, $n$ rows and $m$ columns.
Output: $\xi$ the set of FCCs.
$TL \leftarrow empty(), TM \leftarrow empty()$;
$Z$ and $|Z|$ are computed from $O$;
$\xi \leftarrow cut((H, R, C), Z, 0, |Z|, TL, TM)$;
```

Algorithm 4 Close Row Set Check

```
Rcheck($(H', R', C'), Z$)
Input: node $(H', R', C')$ and cutters list $Z$.
Output: flag $\beta$.
if $\exists r_x \in (R \setminus R')$ such that $\forall(\{h_w\}, \{r_x\}, C_y) \in Z$ where $h_w \in H'$, $C' \cap C_y = \emptyset$
  then
    $\beta \leftarrow 0$;
else
    $\beta \leftarrow 1$;
end if
return $\beta$;
```

Function *cut*() cuts a node $O' = (H', R', C')$ with the first cutter $Z[i] = (W, X, Y)$ that satisfies the following constraints. First, (H', R', C') must have a non empty intersection with $Z[i]$. If it is not the case, *cut*() is called with the next cutter.

To build the left son $L = (H' \setminus W, R', C')$ (lines 9-14), three checks are required: monotonic constraint check $minH(H' \setminus W)$, left track check, and close row set check

Algorithm 5 Close Height Set Check

```
**Hcheck**$((H', R', C'), Z)$
Input: node $(H', R', C')$ and cutters list $Z$.
Output: flag $\alpha$.
**if** $\exists h_w \in (H \setminus H')$ such that $\forall(\{h_w\}, \{r_x\}, C_y) \in Z$ where $r_x \in R'$ , $C' \cap C_y = \emptyset$ **then**
    $\alpha \leftarrow 0$;
**else**
    $\alpha \leftarrow 1$;
**end if**
**return** $\alpha$;
```

(*Rcheck*() in Algorithm 4). If L is not pruned off by the three checks, *cut*() is called to process L, and there is no update on TL, TM sets for L.

To build the middle son $M = (H', R' \setminus X, C')$ (lines 15-20), three checks are required: monotonic constraint check $minR(R' \setminus X)$, middle track check, and close height set check (*Hcheck*() in Algorithm 5). If M is not pruned off by the three checks, *cut*() is called to process M, and the TL set for L is updated to $TL \cup W$.

To build the right son $R = (H', R', C' \setminus Y)$ (lines 21-29), three checks are required: monotonic constraint check $minC(C' \setminus Y)$, close height set check and close row set check. If R is not pruned off by the three checks, *cut*() is called to process R, and the TL, TM sets for L are updated to $TL \cup W, TM \cup X$ respectively.

Since the size of Z and the order of cutters inside Z are important to performance, the algorithm can be optimized by preprocessing the 3D dataset. We adopt two heuristics. First, we transpose the 3D data matrix to make $|H| < |C|$ and $|R| < |C|$, which helps to minimize the size of $|Z|$. Second, we sort the height slices such that height slices with more 0s are always in front of those with fewer 0s. This helps to accelerate the mining by pruning off the search space as early as possible.

Algorithm 6 Cutting

```
cut($(H', R', C'), Z, 0, |Z|, TL, TM$)
Input: Node $(H', R', C')$, cutters list $Z$, iteration number $i$, $|Z|$ the size of $Z$, left
  and right atoms tracks $TL$ and $TM$.
Output: $\xi$ the set of FCCs.
$(W, X, Y) \leftarrow Z[i]$;
if $i \leq |Z| - 1$ then
  if $W \cap H' = \emptyset$ or $X \cap R' = \emptyset$ or $Y \cap C' = \emptyset$ then
    $\xi \leftarrow \xi \cup cut((H', R', C'), Z, i + 1, |Z|, TL, TM)$;
  else
    if $minH(H' \setminus W)$ satisfied and $W \cap TL = \emptyset$ then
      $\beta \leftarrow Rcheck((H' \setminus W, R', C'), Z)$;
      if $\beta = 1$ then
        $\xi \leftarrow \xi \cup cut((H' \setminus W, R', C'), Z, i + 1, |Z|, TL, TM)$;
      end if
    end if
    if $minR(R' \setminus X)$ satisfied and $X \cap TM = \emptyset$ then
      $\alpha \leftarrow Hcheck((H', R' \setminus X, C'), Z)$;
      if $\alpha = 1$ then
        $\xi \leftarrow \xi \cup cut((H', R' \setminus X, C'), Z, i + 1, |Z|, TL \cup W, TM)$;
      end if
    end if
    if $minC(C' \setminus Y)$ satisfied then
      $\alpha \leftarrow Hcheck((H', R', C' \setminus Y), Z)$;
      if $\alpha = 1$ then
        $\beta \leftarrow Rcheck((H', R', C' \setminus Y), Z)$;
        if $\beta = 1$ then
          $\xi \leftarrow \xi \cup cut((H', R', C' \setminus Y), Z, i + 1, |Z|, TL \cup W, TM \cup X)$;
        end if
      end if
    end if
  end if
else
  $\xi \leftarrow (H', R', C')$;
end if
return $\xi$;
```

4.4.3 Correctness

CubeMiner constructs the root (H, R, C) and then reduces simultaneously H, R, C to have the collection of leaves derived from (H, R, C). Theorem 4 shows that *CubeMiner* can correctly generate all and only all FCCs.

Theorem 4. Let $FCCs$ be the set of frequent closed cubes of a 3D dataset. Let LV be the set of leaf nodes derived from *CubeMiner* on the dataset. Then, $FCCs = LV$. In other words, *CubeMiner* can correctly generate all and only all FCCs.

Proof: First, we prove that $FCCs \subseteq LV$. Let (H, R, C) be the original dataset, Z be the whole cutter set and P be the set of pruned nodes. Since $FCCs \subseteq (H, R, C)$, and in the tree building process, only cells valued '0' are removed off by cutters (verified by node's son definition) and only useless nodes (subsets of other nodes) are pruned off (verified by Lemma 8 to Lemma 11), hence, $FCCs \subseteq (H, R, C) \setminus Z \setminus P$, that is, $FCCs \subseteq LV$.

Second, we prove $LV \subseteq FCCs$ by contradiction. Assume there exists a leave $A \in LV$ but $A \notin FCCs$. Then A is either not satisfied by monotonic support constraints or not closed. Let $A = (H_a, R_a, C_a)$ and Z_l, Z_m, Z_r be the set of cutters associated to the left, middle, right branches of the path from the root to A respectively. During the tree building process, each time we cut off a node's height set, the monotonic constraint $minH$ is checked to be satisfied, hence, H_a satisfies the monotonic support constraint. Similarly, R_a and C_a both satisfy their monotonic constraints. Hence, we gather that A is not closed.

Suppose that A is not closed in the column set, then there exists $A' = (H_a, R_a, C_a \cup C'_a)$ where $C'_a \subseteq C \setminus C_a$, and $\forall h_k \in H_a, \forall r_i \in R_a, \forall c_j \in C_a \cup C'_a, O_{k,i,j} = 1$. And since the whole column set C is cut into C_a from the root to A by cutters in Z_r, so there exists a set of cutters $Z_a \subseteq Z_r$ to cut off C'_a and $\forall(W,X,Y) \in Z_a$, either (a) $W \subseteq H \setminus H_a$ or (b) $X \subseteq R \setminus R_a$. Given any of A's ancestor $B = (H_b, R_b, C_b)$ derived from a cutter $(W,X,Y) \in Z_a$. Since B is a right son, $W \subseteq H_b, X \subseteq R_b$, and the TL and TM sets are updated into $TL \cup W$ and $TM \cup X$ respectively. For case (a), $W \nsubseteq H_a$, there must exist a cutter in Z_l to remove off W on the path from B to A. That is, between B and A, there must exist a left-son offspring of B. However, since the left atom of the cutter $W_l \cap TL = W \neq \emptyset$, the left-son offspring is pruned off and hence no A will be generated, which is contrary to the previous assumption. For case (b), it is similar to (a): during the process to remove off X from R_b, the middle-son offspring of B is pruned off due to $X_m \cap TM = X \neq \emptyset$. As a result, A will not be generated and it is contrary to the assumption too. Hence, we conclude that the assumption is wrong and A is closed in the column set.

Suppose that A is not closed in the row set, then there exists $A' = (H_a, R_a \cup R'_a, C_a)$ where $R'_a \subseteq R \setminus R_a$, and $\forall h_k \in H_a, \forall r_i \in R_a \cup R'_a, \forall c_j \in C_a, O_{k,i,j} = 1$. And since the whole column set R is cut into R_a from root to A by cutters in Z_m, so there exists a set of cutters $Z_a \subseteq Z_m$ to cut off R'_a and $\forall(W,X,Y) \in Z_a$, either (c) $W \subseteq H \setminus H_a$ or (d) $Y \subseteq C \setminus C_a$. Given any of A's ancestor $B = (H_b, R_b, C_b)$ obtained from a cutter $(W,X,Y) \in Z_a$. Since B is a middle son, $W \subseteq H_b, Y \subseteq C_b$, and the TL set is updated into $TL \cup W$. Case (c)'s proof is the same as case (a) above. As for case (d), $Y \nsubseteq C_a$, there must exist cutters in Z_r to remove off Y on the path from B to A. Let $B' = (H_b, R_b, C_a)$ be the right-son offspring of B after removing Y. Since $X \cap R_b = \emptyset$, and $X \cap R'_a \neq \emptyset$, $\exists r_u = X \cap R'_a$ such that $\forall h_k \in H_a, \forall c_j \in C_a, O_{k,u,j} = 1$. Hence B' is not row set closed due to r_u and will be pruned off in the close row set checking of right son building process. As a result, A will not be generated, which is contrary to the assumption. Hence we conclude that the assumption is wrong and A is closed in the row set.

Suppose that A is not closed in the height set, then there exists $A' = (H_a \cup H'_a, R_a, C_a)$ where $H'_a \subseteq H \setminus H_a$, and $\forall h_k \in H_a \cup H'_a, \forall r_i \in R_a, \forall c_j \in C_a, O_{k,i,j} = 1$. And since the whole height set H is cut into H_a from the root to A by cutters in Z_l, so there exists a set of cutters $Z_a \subseteq Z_l$ to cut off H'_a and $\forall(W, X, Y) \in Z_a$, either (e) $X \subseteq R \setminus R_a$ or (f) $Y \subseteq C \setminus C_a$. Like the proof in case (d), in case (e)/(f), the ancestor of A will be pruned off as it will be unclosed in the height set checking during middle/right son building process. Hence, A will not be generated, and it is contrary to the assumption. We conclude that the assumption is wrong and A is closed in the height set.

Now, we have concluded that A is closed and satisfies all monotonic constraints. Hence, $A \in FCCs$ and our assumption that there exists a leave $A \in LV$ but $A \notin FCCs$ is wrong. That is, $LV \subseteq FCCs$. So, our algorithm for mining $FCCs$ is correct in that $LV = FCCs$. □

4.5 Parallel FCC Mining

Given that FCC mining is computationally expensive, a solution to reduce the response time is to exploit parallelism. In this section, we shall show how our proposed *RSM* and *CubeMiner* can be parallelized easily.

In general, a parallel algorithm typically comprises three logical phases: (a) a task generation phase that splits the original task into smaller sub-tasks; (b) a task allocation phase that assigns the sub-tasks to the processors; (c) a task execution phase where every processor operates on the allocated sub-tasks. An important factor in parallelism is to minimize interference during the execution phase, so that all processors can operate independently and concurrently without having to communicate with one another.

It turns out that both *RSM* and *CubeMiner* fit nicely into the above framework: tasks can be generated and allocated to processors to be executed independently.

- **Parallel *RSM.*** In *RSM*, the mining of each representative slice corresponds to a task, in other words, the maximum number of tasks is the number of eumerations of the base dimensions (those enumerations that do not meet the minimum thresold requirement are dropped). Each of these tasks can be allocated to a processor, and can be processed independently.

- **Parallel *CubeMiner.*** In *CubeMiner*, each branch of the tree splitting process can be processed independently, and thus, each branch corresponds to a task. In other words, we can allocate a branch of the tree splitting process to a processor.

For both *Parallel-RSM* and *Parallel-CubeMiner*, to ensure that the tasks can be processed independently, each processor requires a copy of the entire dataset. This is necessary so that the post-pruning phase can be performed independently. Fortunately, the communication overhead (to transmit the dataset to all processors) is not significant: (a) the dataset can be transmitted while the tasks are being generated, so the response time is not much affected; (b) the communication cost is relatively small compared to the mining cost.

4.6 Time Complexity

The time complexity of mining FCCs is exponential in the number of patterns. For the 3D dataset $O = H \times R \times C$, where $|H| = L$, $|R| = N$, $|C| = M$, the time complexity of *RSM* and *CubeMiner* is $O(2^{L+N} \times N^2 \times M)$ and $O(2^{LN} \times M)$ repectively, without applying any pruning strategy. By applying $minH$, $minR$, $minC$ and closeness constraints and early pruning strategies, the efficiency of *RSM* and *CubeMiner* can be improved significantly.

4.7 Experimental Results

We have implemented the *RSM* framework and *CubeMiner* in C. For the *RSM* framework, we employed D-Miner [7] in phase two as the 2D FCP mining scheme. This is because D-Miner keeps the supporting row set of each FCP during the processing, which is important for close check of 3D FCC. Moreover, D-Miner has been shown to perform well in relatively dense datasets. We conducted a performance study to evaluate the efficiency of *RSM* against *CubeMiner*, and study the optimization of *CubeMiner*. In addition, we also study the parallel versions of *RSM* and *CubeMiner*. For our experiments, we use two real 3D microarray datasets: the yeast cell-cycle regulated genes [51] (http://genomewww.stanford.edu/cellcycle) in the Elutriation Experiments and CDC15 Experiments respectively. To study the effect of the proposed schemes on scalability, we also use synthetic datasets generated by the IBM data generator. (The generator is available at http://www.cs.umbc.edu/∼cgiannel/assoc_gen.html.) All the experiments are run on a Pentium 4 PC with 1 GB RAM.

4.7.1 Results from Real Microarray Datasets

In this section, we experiment on two real microarray datasets. For the Elutritration Experiments, there are a total of 7161 genes whose expression values are measured from time 0 to 390 minutes at 30 minute intervals (a total of 14 time points). And for the CDC15 Experiments, there are a total of 7761 genes whose expression values are measured from time 70 to 250 minutes at 10 minute intervals (a total of 19 time points). Finally, we use 9 of the attributes of the raw data as the samples (e.g., the raw values for the average and normalized signal for Cy5 and Cy3 dyes, the ratio of those values, etc.) [64]. Thus, from the Elutritration dataset, we obtain a 3D expression matrix of size: $T \times S \times G = 14 \times 9 \times 7161$; and from the CDC15 dataset, we obtain a 3D expression matrix of size: $T \times S \times G = 19 \times 9 \times 7761$.

Data Preprocessing

We normalize the 3D datasets to make its cell value '1' or '0', where value '1' means high expression value and '0' otherwise. For dataset $O' = T \times S \times G = \{O'_{k,i,j}\}$ with $k \in [1, l]$, $i \in [1, n]$ and $j \in [1, m]$. We normalize O' into a $T \times S \times G$ matrix O as follows:

$$O_{k,i,j} = \begin{cases} 1 & \text{if } O'_{k,i,j} \geq \frac{\sum_{j=1}^{m} O'_{k,i,j}}{m}, \\ 0 & \text{if } O'_{k,i,j} < \frac{\sum_{j=1}^{m} O'_{k,i,j}}{m}. \end{cases}$$

Table 4.4 shows an example of Original Dataset O' at time point $Time = 30min$. And its normalized matrix O is in Table 4.5. After normalization, we get two 3D datasets with a density around 30%, that is, 30% of the cells value 1.

Table 4.4: Example of Original Data O' ($T = 30min$).

G/S	CH1B	CH1D	CH2I	CH2B
YAL014C	463	301	162	374
YAL015C	528	299	229	392
YAL016W	810	321	489	734
YAL017W	478	283	195	359

Table 4.5: Example of Normalized Matrix O ($T = 30min$).

G/S	CH1B	CH1D	CH2I	CH2B
YAL014C	0	1	0	0
YAL015C	0	0	0	0
YAL016W	1	1	1	1
YAL017W	0	0	0	0

CubeMiner **Optimization**

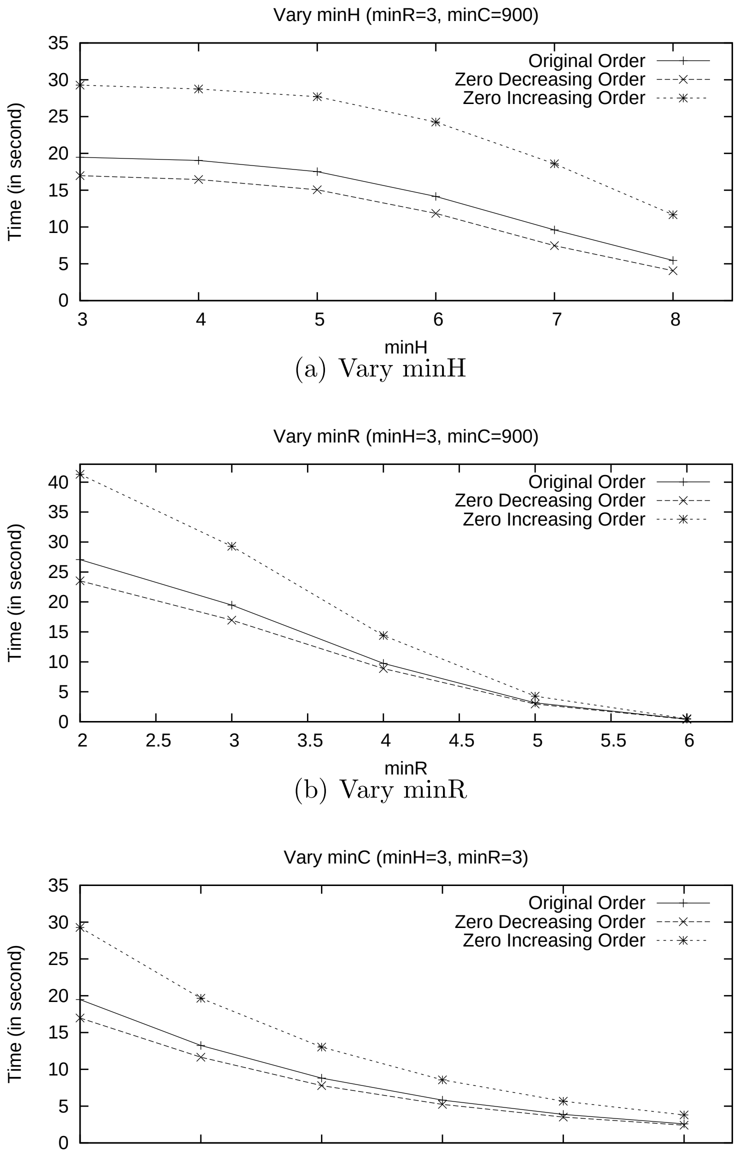
Before comparing the performance of *CubeMiner* and the *RSM* framework, we first study the optimization of *CubeMiner*. We experiment on the Elutritration dataset and sort the original dataset by Time Slice. We first sort the time slice such that time slices with more 0s are always in front of those with fewer 0s, which is called "Zero Decreasing Order"; then we sort the time slice such that time slices with fewer 0s are always in front of those with more 0s, which is called "Zero Increasing Order". We compare the performance of *CubeMiner* on the original order, Zero Decreasing Order and Zero Increasing Order. Figure 4.3 shows the results as we vary $minH$, $minR$ and $minC$ respectively. First, we observe that with the increase in $minH$, $minR$ and $minC$ values, regardless of the ordering of the datasets, the processing time of *CubeMiner* decreases. This is expected since a larger threshold value means that we can prune a larger space as the answer size is smaller. Second, in all the three cases, we observe that *CubeMiner* performs best on the dataset with Zero Decreasing Order, and worst on the dataset with Zero Increasing Order. The performance of *CubeMiner* on the original dataset stays in the middle position. *CubeMiner* performs best when the dataset is sorted by the Zero Decreasing Order because applying cutters with more 0s first will remove the patterns that do not satisfy the minimum thresholds early. That is, it helps to prune off the search space early, and hence accelerates the mining process. Based on these results, in the following experiments, we adopt an optimized version of *CubeMiner* that pre-sorts the datasets in Zero Decreasing Order before performing FCC mining.

Vary Monotonic Constraints

In this experiment, we vary the monotonic support constraints $minH$, $minR$ and $minC$, and study the performance of *RSM* and *CubeMiner* respectively. For *RSM*, we examine two versions using dimensions H and R as the base dimensions respectively. We denote these versions as *'RSM-H'* and *'RSM-R'* respectively. As we will be enumerating the H and R dimensions, the constraint $minC$ on dimension C will have a relatively smaller effect. Hence, we study the effect of $minC$ first.

The results are shown in Figure 4.4. In Figure 4.4(a), we see clearly that *RSM-R* is much faster than *RSM-H*. This is because $|R| < |H|$ and a larger enumerated dimension leads to more representative slices. Hence, the enumeration on the smallest dimension always leads to better performance of *RSM*. When we refer to *RSM* in the following experiments, we default it as taking the smallest dimension to enumerate.

We also see that the execution time of *CubeMiner* and *RSM-R* both decrease with the increase of $minC$. Moreover, for the Elutritration dataset, *RSM-R* is faster than

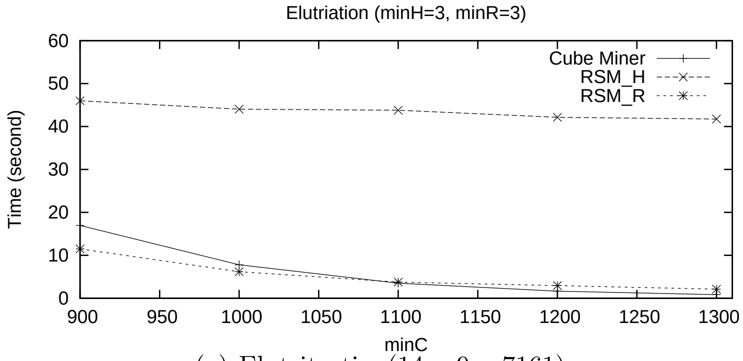


(a) Vary minH

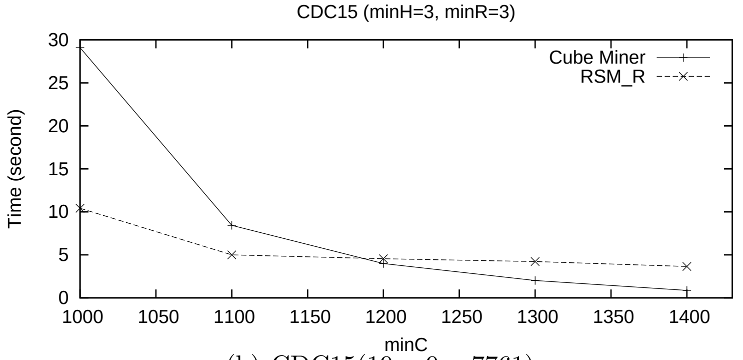
(b) Vary minR

(c) Vary minC

Figure 4.3: *CubeMiner* Optimization.



(a) Elutritration(14 × 9 × 7161)



(b) CDC15(19 × 9 × 7761)

Figure 4.4: Vary *minC*.

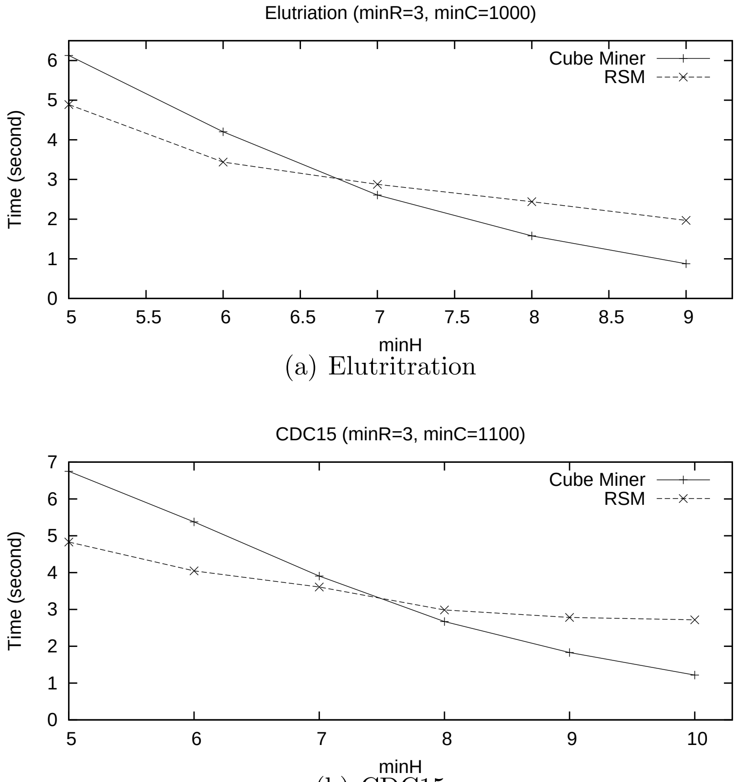
CubeMiner when *minC* is below 1000. However, *CubeMiner* catches up and performs better when *minC* increases above 1100. Similarly, for the CDC15 dataset, *RSM* is faster when *minC* is less than 1100. This is due to the underlying working strategies of *RSM* and *CubeMiner*. As we know, the number of cutters in *CubeMiner* has an important effect on the tree's depth, and hence affects its performance. *RSM* mines on each representative slice, which has much fewer rows compared with the number of cutters in *CubeMiner*. That is, the datasets (representative slice) that *RSM* works on, is much smaller than the ones (whole dataset) that *CubeMiner* does. And the execution time of *RSM* is the sum of the execution time on each representative slice. This makes *RSM* efficient if the number of representative slices is not large. However, the number of representative slices increases very quickly with the increase of the dimension size to be enumerated, which limits the advantage of *RSM* to a great extent. That's why *RSM* runs faster when the enumerated dimension is very small but runs much slower as the smallest dimension grows. As we may see from *RSM-H* in Figure 4.4 (a), when the enumerated dimension has a size of 14, *RSM-H* performs worse than *CubeMiner*. And, as we shall see shortly, in the synthetic datasets where larger dimension size is used, this trend is more obvious. In application, the size of smallest dimension in 3D dataset is usually not very small, which makes *CubeMiner* more efficient than *RSM* in practice.

Even when the enumerated dimension has a small size of 9 for *RSM*, with the increase in $minC$, *CubeMiner* catches up quickly. This is because *CubeMiner* directly works on the 3D dataset which prunes off the search space as soon as possible while *RSM* takes time in representative slice generation before performing space pruning.

Next, we study the variation of $minH$, $minR$ on the two 3D datasets and set $minC = 1000$ for the Elutritration dataset and $minC = 1100$ for the CDC15 dataset. The $minC$ values are selected such that *CubeMiner* has a nearly similar but little longer processing time than *RSM*, to minimize the effects of $minC$ on the performance. Figure 4.5 and Figure 4.6 show the results respectively. The relative performance between *RSM* and *CubeMiner* remains largely the same for the same reasons given in the other experiments.

Effect of Parallelism

To make efficient the processing, we also propose the parallel version of *RSM* and *CubeMiner*. In the parallel schemes, each processing node holds the whole dataset



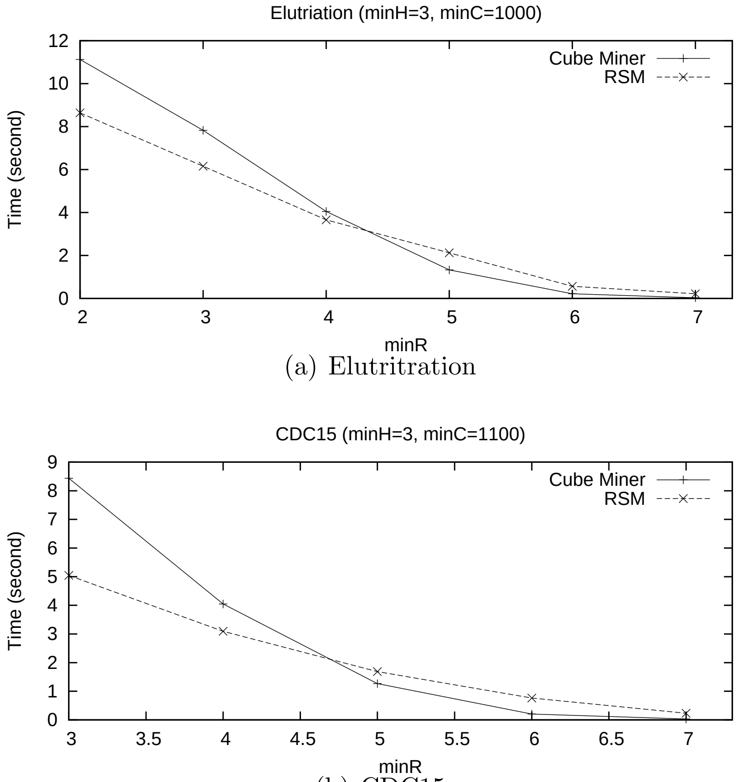
(a) Elutritration

(b) CDC15

Figure 4.5: Vary *minH*.

at the initial status. Then as the tree grows, new generated branches are sent to available nodes for processing. We only send tasks to nodes, and the tasks can be independently executed without further information communication. Hence, the information communication between nodes is very small.

In this experiment, we study the effect of the number of processors on the processing time. The number of processors is varied from 1 to 32. We present the results on the CDC15 dataset. The results are shown in Figure 4.7. First, we observe that the parallel version of *RSM-R* outperforms the parallel version of *CubeMiner*. This is because, for this experiment, the experimental setup favors *RSM-R*, i.e., this is the



(a) Elutritration

(b) CDC15

Figure 4.6: Vary *minR*.

setting where the uniprocessor *RSM-R* also outperforms *CubeMiner* (see Figure 4.4 where $minC = 1000$). Second, we note that as the degree of parallelism increases, the response time also decreases. Moreover, as in traditional parallel processing, there is a certain "optimal" number of processors beyond which additional parallelism leads to only marginal gain. In this experiment, for both schemes, the speedup is good for upto 8 processors. Beyond 8 processors, the speedup starts to degrade.

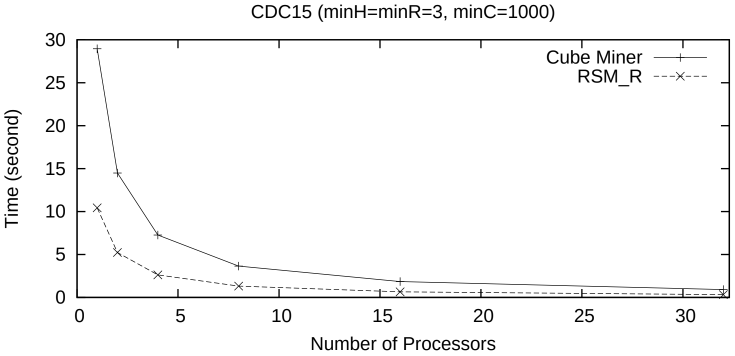


Figure 4.7: Vary Number of Processors.

4.7.2 Results on Synthetic Datasets

To study the scalability of our proposed schemes, we generate synthetic datasets using the IBM data generator. Since *RSM*'s efficiency depends greatly on the size of the smallest dimension, in the first set of experiments, we study the effects of the size of smallest dimension on the execution time. We experiment on seven synthetic datasets with 30% density (percentage of cells with value one), 20 rows, 1000 columns, and the number of heights varied from 8 to 20. We set $minH = minR = 3$, and $minC = 30$ for all the experiments. Figure 4.8 shows the execution time in logarithm (second) scale. We see that the execution time of *RSM* and *CubeMiner* increase with increasing height number. We also observe that *RSM*'s execution time increases much faster as the size of the heights increases. For larger datasets, *CubeMiner* is clearly much more efficient than *RSM*.

To study the scalability on large dataset, we generate synthetic datasets with 100 heights, 100 rows, 10000 columns, and 10% density. We study the execution time of *RSM* and *CubeMiner* with the variation of $minH$, $minR$, and $minC$. *RSM* failed to finish processing after long hours even on very high support constraints,

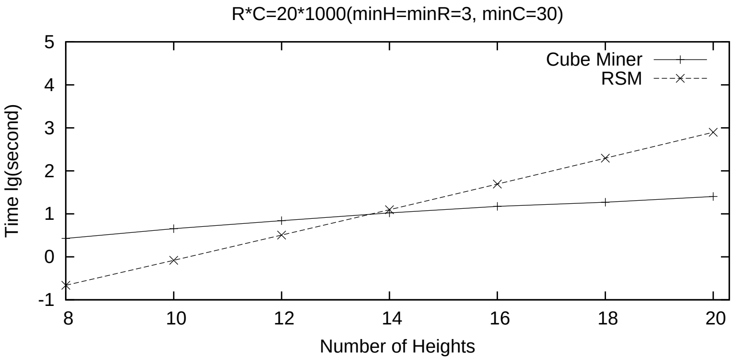


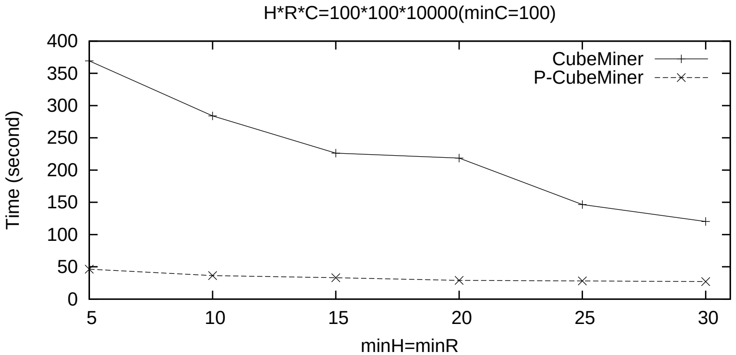
Figure 4.8: Vary Size of Height Dimension.

which is incomparable to *CubeMiner*. Even its parallel version takes longer time than *CubeMiner*. This is because, with 100 heights, the number of slices to be enumerated is very large. Hence, we only report the execution time of *CubeMiner* and its parallel version *P-CubeMiner* with 8 processors (the "optimal" number) in Figure 4.9. From the results, we can confirm that (for the dataset used) 8 processors offer very good speedup. Moreover, we note that the parallel version of *CubeMiner* can reduce the computational cost significantly.

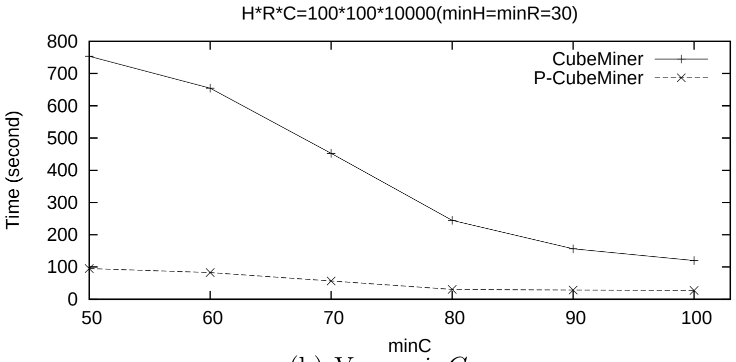
From the experiments on synthetic datasets, we see that *CubeMiner* scales well on large datasets while *RSM* works efficiently only on datasets with a small size in one dimension.

4.7.3 Biological Significance

The FCCs mined from the real microarray datasets are able to deliver some interesting patterns for biologists. In the final group of experiments, we set $minH = 5$, $minR = 5$, and $minC = 1200$ for the Elutritration and CDC15 datasets and get 13 and 250 FCCs respectively. Some known co-regulated genes already established



(a) Vary *minH* and *minR*



(b) Vary *minC*

Figure 4.9: Vary *minH*, *minR* and *minC*.

Table 4.6: Known Co-regulated Genes from Elutritration Dataset.

M/G1 Boundary Regulated:
CLN3(YAL040c), SIC1(YER120w), STE2(YFL026w)
SIC1(YER120w), STE2(YFL026w), ASH1(YKL185w)
CLN3(YAL040c), SWI5(YDR146c)
SIC1(YER120w), RME1(YGR044c)
SWI4(YER111c), SIC1(YER120w)
CDC47(YBR202w), SIC1(YER120w)
Late G1(SCB) Regulated:
PSA1(YDL055c),MNN1(YER001w),FKS1/CWH53(YLR342w),GAS1(YMR307w), VAN2(YGL225w)
PSA1(YDL055c), MNN1(YER001w), FKS1/CWH53(YLR342w), GAS1(YMR307w), CLN2(YPL256c)
PSA1(YDL055c), SCD2/CHS3(YBR023c)
S-phase Regulated:
HTB2(YBL002w), HTB1(YDR224c), HTA1(YDR225w), HHF2(YNL030w), HHT2(YNL031c), HHT1(YBR010w)
HTB2(YBL002w), HTA2(YBL003c)
G2/M-phase Regulated:
SED1(YDR077w), MST1(YBR054w)
SED1(YDR077w), SPO12(YHR152w)
SED1(YDR077w), CLB1(YGR108w)
SED1(YDR077w), SWI5(YDR146c)
SED1(YDR077w), MOB1(YIL106w)

Table 4.7: Known Co-regulated Genes from CDC15 Dataset.

M/G1 Boundary Regulated:

CLN3(YAL040c), CDC47(YBR202w), FUS1(YCL027w), SIC1(YER120w)
FUS1(YCL027w), SIC1(YER120w), CDC46(YLR274w), AGA1(YNR044w)
TEC1(YBR083w), CDC47(YBR202w), FUS1(YCL027w)
SIC1(YER120w), CDC6(YJL194w)
SIC1(YER120w), CTS1(YLR286c)
FUS1(YCL027w), PCL2(YDL127w)

Late G1(SCB) Regulated:

PSA1(YDL055c), MNN1(YER001w), VAN2(YGL225w), KRE6(YPR159w)
PSA1(YDL055c), TIP1(YBR067c)

S-phase Regulated:
HTB2(YBL002w), HTA2(YBL003c), HHF1(YBR009c), HHT1(YBR010w), HTB1(YDR224c), HTA1(YDR225w)
G2/M-phase Regulated:
MST1(YBR054w), MST2(YDR033w), SED1(YDR077w), CDC20(YGL116w), CLB1(YGR108w), MOB1(YIL106w), CDC5(YMR001c)

by biological research [38, 8] are found in our resulting FCCs. Table 4.6 and Table 4.7 show the M/G1 Boundary co-regulated genes, Late G1(SCB) regulated genes, S-phase, and G2/M-phase co-regulated genes identified from the Elutritration and CDC15 experimental datasets respectively. From the 13 FCCs generated from the Elutritration datasets, 4 FCCs contain co-regulated genes of the four categories, while 9 FCCs fail to identify co-regulated genes from G2/M-phase. All 250 FCCs generated from the CDC15 datasets contains the co-regulated genes of the four categories. Late G1(MCB) Regulated Genes and S/G2-phase Regulated Genes fail to be identified in the FCCs from both experiments.

4.8 Summary

In this chapter, we have generalized 2D frequent closed pattern mining into 3D context. We defined the model of 3D frequent closed pattern – Frequent Closed Cube (FCC). We proposed two schemes to mine FCCs - while the *Representative Slice Mining* framework *(RSM)* enables us to reuse existing 2D frequent closed pattern mining algorithm, *CubeMiner* operates on the 3D space directly. We also presented parallel versions of the two schemes. We conducted extensive performance study on both real and synthetic datasets. Our results showed that both schemes can mine FCC efficiently, in particular, *CubeMiner* is superior for large datasets, while *RSM* performs best when one of the dimensions is small. Moreover, the parallel versions of both schemes can further reduce the computation time significantly. Furthermore, the FCCs mined from the real microarray datasets (Elutritration and CDC15 datasets) are able to deliver some known co-regulated genes already established by biological research.

Chapter 5

Quick Hierarchical Biclustering on 2D Expression Data

5.1 Overview

In this chapter, we propose an efficient top-down hierarchical biclustering algorithm called *Quick Hierarchical Biclustering (QHB)*, to mine biclusters with consistent trends. *QHB* continuously partitions the whole dataset into subsets such that genes with more consistent trends during condition transitions are grouped together while genes with inconsistent trends are set apart. To measure the trend consistency of a bicluster, we define a new score that reflects the similarity of fluctuating degrees in the changing trends. Compared with previous biclustering models, we have made five main contributions:

First, we define a new bicluster quality measurement called *Mean Fluctuating Degree* (*MFD*) to reflect the trend consistency of biclusters. Since a similarity score is not enough to ensure trend consistency, we use our *MFD* only as a supplementary control agent. Instead, the trend consistency is mainly controlled and embedded in the partitioning strategy of *QHB*, which ensures the high quality of consistent trends within each bicluster.

Second, instead of improving on only part of the “seeds”, *QHB* takes the entire dataset into consideration. During the hierarchical partitioning process, all valuable information of a parent node is kept into the child nodes without any loss.

Third, *QHB* adopts a partition based refinement that can simultaneously processed several rows/columns. This is much more efficient than existing techniques.

Fourth, *QHB* provides a very clear hierarchical inter-bicluster relationships. Such graphical representation of the relationships among biclusters provides more valuable knowledge to the biologists. To the best of our knowledge, no previous work has established a clear relationship between biclusters.

Finally, the hierarchical partitioning strategy of *QHB* facilitates a progressive refinement of results. Biclusters are refined from generality to details progressively. This is very helpful in biological application. Instead of waiting long hours for all detailed results, biologists now would be provided with a general picture of the whole results from the upper levels of the hierarchical tree in a very short response time. Then biologists could freely choose their focus, rolling up to generalize it or rolling down to detail it, progressively. This would help biologists quickly focus on their most interested patterns for further exploration.

The rest of this chapter is organized as follows. We will introduce the *QHB* algorithm in Section 5.2. In Section 5.3, we report results of an experimental study on the real time-series yeast gene expression data. We compare our *QHB* algorithm against a recently proposed DBF scheme [63]. We also show the inter-bicluster relationships obtained from *QHB*. In Section 5.4, we extend the *QHB* scheme to process datasets with non-consecutive condition transitions. Finally, in Section 5.5, we draw conclusions with directions for future research.

5.2 *QHB*: Quick Hierarchical Biclustering Algorithm

In this section, we present the proposed *QHB* framework. The *QHB* algorithm comprises 3 phases. In the first phase, the original matrix is transformed into a binary matrix that captures the changing trend of the gene expression value between each consecutive conditions. This trend could either be a rising trend, a falling trend or one that is considered to have no significant change. In the second phase, an iterative partitioning procedure is applied to the transformed binary matrix such that genes with different trends under subsets of consecutive conditions are split into different sub-matrices. Each sub-matrix forms a "coarse" biclustering seed that reveals a subset of genes exhibiting consistent rising/falling trends under a subset of consecutive conditions. In the final phase of *QHB*, the trends in "coarse" seeds are further binned such that the seeds could be further partitioned and refined into biclusters where the trends exhibit similar degrees of fluctuation under consecutive conditions. A new score that reflects the similarity of trends' fluctuating degrees is defined to measure the bicluster quality. Biclusters in which genes display consistent trends with similar degrees of fluctuation under consecutive conditions are considered as biclusters of good quality.

In this section, for ease of presentation, we focus on datasets that emphasize the order of conditions, e.g., time series gene expression data. For such datasets, the gene expression values' variation under non-consecutive conditions (time points) are meaningless, hence we only consider a pattern's changing trends under consecutive conditions. We defer the discussion on extending our scheme to non-consecutive conditions to Section 5.4.

Table 5.1: Original Data Matrix O.

O	c_1	c_2	$c3$	$c4$
g_1	2.4	2.95	2.45	2.99
g_2	1.95	1.71	1	0.29
g_3	0.5	1.1	0.38	1.56

Table 5.2: Slope Angle Matrix O'.

O'	c_1c_2	c_2c_3	c_3c_4
g_1	28.81°	−26.57°	28.37°
g_2	−13.50°	−35.37°	−35.37°
g_3	30.96°	−35.75°	49.72°

5.2.1 Phase 1: Matrix Transformation

Let $G = \{g_1, g_2, \ldots, g_m\}$ be the set of genes, and $C = \{c_1, c_2, \ldots, c_n\}$ be the set of experimental conditions (samples/time points), then the gene expression data can be represented as an $m \times n$ matrix O with each cell $O_{i,j}$ corresponding to the expression value of gene g_i under condition c_j.

To measure the fluctuating degrees of trends when conditions change, the original matrix O is first transformed into a slope angle matrix O', such that rows of O' represent genes while columns of O' represent transitions between two consecutive conditions. And the cells in O' contain the slope angles of changing trends under condition transitions. Given an $m \times n$ matrix $O = G \times C$, its slope angle matrix is an $m \times (n-1)$ matrix $O' = G \times C'$ such that $C' = \{c_1c_2, c_2c_3, \ldots, c_{n-1}c_n\}$ and $O'_{i,j} = arctan(O_{i,j+1} - O_{i,j})$. Given the running example O in Table 5.1, its transformed slope angle matrix O' is shown in Table 5.2.

In O', a positive angle indicates a rising trend while a negative one indicates a falling trend. O' is further transformed into a binary matrix O'' as follows: each column c_ic_j in O' is replaced by two binary columns c_ic_j and $(c_ic_j)'$ in O'' to capture the rising and falling trends. For example, a rising trend in $O'_{i,j}$ will be replaced by two cells $O''_{i,j}$ and $O''_{(i,j)'}$ with values "0" and "1" respectively. Similarly, a falling trend would be represented by "1" and "0". To eliminate trends with trivial changes, we set an angle threshold $t°(t° > 0°)$ to bin the rising/falling trends; the resultant binary representations are "0" and "0". More formally, given the $m \times (n-1)$ slope angle matrix $O' = G \times C'$, we would get an $m \times 2(n-1)$ binary matrix $O'' = G \times C''$ such that $C'' = \{c_1c_2, (c_1c_2)', c_2c_3, (c_2c_3)', \ldots, c_{n-1}c_n, (c_{n-1}c_n)'\}$, and

$$O''_{i,j}, O''_{(i,j)'} = \begin{cases} 0,1 & \text{if } t° < O'_{i,j} < 90°, \\ 1,0 & \text{if } -90° < O'_{i,j} < -t°, \\ 0,0 & \text{otherwise.} \end{cases}$$

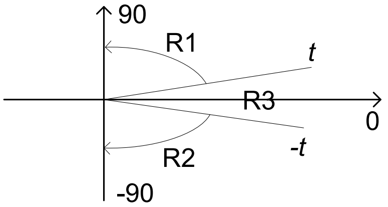


Figure 5.1: Matrix Binning Threshold: $t°$.

Figure 5.1 shows how the angle threshold $t°$ bins the trends. Trends with slope angles in range R_1 are binned as "rising", in range R_2 as "falling" and in range R_3 as "trivial change". As for the running example O' in Table 5.2, its binary matrix O'' is shown in Table 5.3.

From O'', if we take "1" as "present" and "0" as "empty", then $c_{n-1}c_n$ could be regarded as a falling transition and $(c_{n-1}c_n)'$ as a rising transition. In this way, the rising and falling trends are divided apart into two consecutive columns while trends

Table 5.3: Binary Matrix O'': $t = 26.5°$.

O''	c_1c_2	$(c_1c_2)'$	c_2c_3	$(c_2c_3)'$	c_3c_4	$(c_3c_4)'$
g_1	0	1	1	0	0	1
g_2	0	0	1	0	1	0
g_3	0	1	1	0	0	1

with trivial change are blocked out. This transformation serves as an important basis for the efficient processing in phase 2.

5.2.2 Phase 2: Biclustering Seed Generation

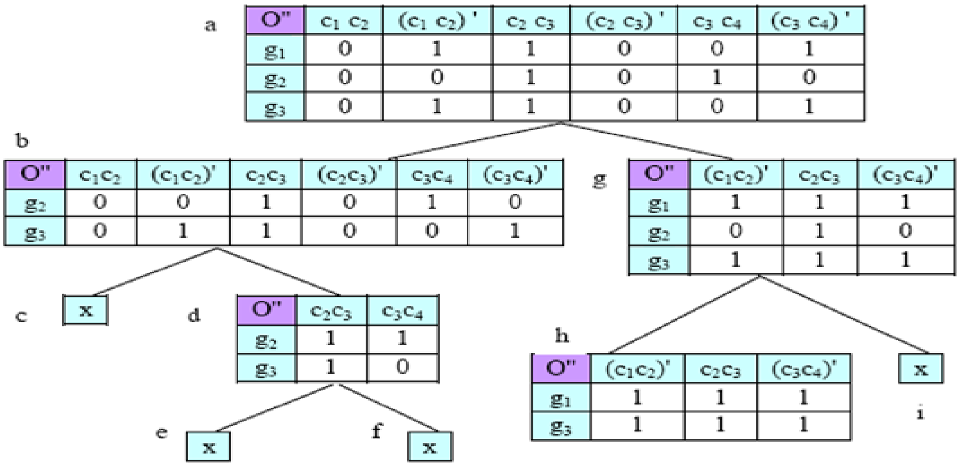
In phase 2, we generate the coarse biclustering seeds where subsets of genes show consistent rising/falling trends under subsets of consecutive conditions. In phase 1, the binary matrix O'' has already set apart different trends and blocked out the trivial trends. Hence, the mining of coarse biclustering seeds is equivalent to mining O'''s "maximal" submatrices with cells all valued "1". The submatrix is "maximal" in the sense that adding one more row/column into the submatrix will bring in cells valued "0". A maximal submatrix is a biclustering seed if its row set and column set satisfy user specified minimum gene number threshold *minGen* and minimum condition transition number threshold *minCon* respectively.

Definition 5.1 Maximal Submatrix: Let $A = G_A \times C_A$ be the submatrix of $O'' = G \times C''$, if (1)$\forall g_i \in G_A, c_j \in C_A, O''_{i,j} = 1$; and (2)$\forall g_k \in G \setminus G_A, \exists c_j \in C_A$ such that $O''_{k,j} = 0$; and (3)$\forall c_l \in C'' \setminus C_A, \exists g_i \in G_A$ such that $O''_{i,l} = 0$ are satisfied, A is defined as the maximal submatrix of O''.

Definition 5.2 Biclustering Seed: Given the minimum gene number threshold *minGen* and minimum condition transition number threshold *minCon*, if (1) $A =$

$G_A \times C_A$ is a maximal submatrix of O''; and (2) $|G_A| \geq minGen$; and (3)$|C_A| \geq minCon$ are satisfied, A is defined as the biclustering seed.

To mine the biclustering seeds, we adapt the hierarchical partitioning framework of D-Miner [7]. The partitioning starts at the root O'', continuously splits the matrix into two submatrices by removing cells valued "0" row by row (or column by column). Whenever a row/column of "0"s is removed, the node is split into a left child submatrix without the row/column, and a right child submatrix without columns/rows containing "0"s. The whole partitioning process ends when all "0"s are removed from all submatrices. Submatrices that are non-maximal or do not satisfy $minGen$ and $minCon$ are pruned off.



a

O"	$c_1 c_2$	$(c_1 c_2)'$	$c_2 c_3$	$(c_2 c_3)'$	$c_3 c_4$	$(c_3 c_4)'$
g_1	0	1	1	0	0	1
g_2	0	0	1	0	1	0
g_3	0	1	1	0	0	1

b

O"	c_1c_2	$(c_1c_2)'$	c_2c_3	$(c_2c_3)'$	c_3c_4	$(c_3c_4)'$
g_2	0	0	1	0	1	0
g_3	0	1	1	0	0	1

g

O"	$(c_1c_2)'$	c_2c_3	$(c_3c_4)'$
g_1	1	1	1
g_2	0	1	0
g_3	1	1	1

d

O"	c_2c_3	c_3c_4
g_2	1	1
g_3	1	0

h

O"	$(c_1c_2)'$	c_2c_3	$(c_3c_4)'$
g_1	1	1	1
g_3	1	1	1

Figure 5.2: Phase 2: Partitioning Process.

Figure 5.2 shows the partitioning procedure of the running example O'' in Table 5.3. Given $minGen = minCon = 2$, after removing cells valued "0", submatrix h is the final seed where genes g_1, g_2 show consistent rising trends under condition transitions $(c_1c_2)', (c_3c_4)'$, and consistent falling trends under condition transition c_2c_3. Cells valued "0" are removed row by row in this example because the number of rows is smaller than the number of columns (We always take the smaller dimension as it is more efficient to do so).

The partitioning process builds up a hierarchical tree where all valuable upper level information are kept intact into the lower level. This helps to avoid any information loss. Moreover, the binning and partitioning ensure that genes with consistent trends under condition transitions are kept together in the same seeds while genes with inconsistent trends are separated apart into different seeds. This scheme helps maintain the bicluster quality effectively.

5.2.3 Phase 3: Bicluster Refinement

In the final phase, the biclustering seeds are further refined to reflect the similarity of trends' fluctuating degree.

The similarity of trends' fluctuating degrees is mainly controlled by the binning and partitioning procedures as previous phases. Since the biclustering seeds mined from phase 2 are assured to have consistent rising/falling trends, we need not consider the directional movements in this phase, but only bin the trends of the seeds into different degrees of slope angles. In our algorithm, we further bin the trends into "steep" or "gentle" movement.

Consider a seed generated in the phase 2 as a $p \times q$ matrix S, where $S = \{g_1, g_2, \ldots, g_p\} \times \{c_1, c_2, \ldots, c_q\}$. As we bin the trends into two categories ("steep" and "gentle"), we need only one more slope angle threshold $t'^{\circ}(t^{\circ} < t'^{\circ} < 90^{\circ})$. According to the slope angle matrix O', S is further binned into a $p \times 2q$ binary matrix S' with "0,1" indicating a steep trend and "1,0" a gentle trend. Hence, $S' = \{g_1, g_2, \ldots, g_p\} \times \{c_1, c'_1, c_2, c'_2, \ldots, c_q, c'_q\}$ such that

Table 5.4: 2-Bin Binary Matrix S'_h: $t' = 45°$.

S'_h	c_1c_2	$(c_1c_2)'$	c_2c_3	$(c_2c_3)'$	c_3c_4	$(c_3c_4)'$
g_1	1	0	1	0	1	0
g_3	1	0	1	0	0	1

$$S'_{i,j}, S'_{i,j'} = \begin{cases} 0,1 & \text{if } t'^{\circ} \leq |O'_{i,j}| < 90^{\circ}, \\ 1,0 & \text{if } t^{\circ} < |O'_{i,j}| < t'^{\circ}. \end{cases}$$

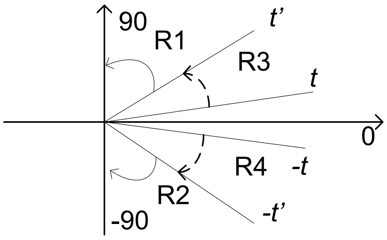


Figure 5.3: Matrix Binning Threshold: t'°.

Figure 5.3 shows how the angle threshold t'° bins the trends. Trends with slope angles in range R_1, R_2 are binned as "steep" and in range R_3, R_4 as "gentle". As for the seed S_h in the running example in Figure 5.2, its 2-bin binary matrix S'_h is shown in Table 5.4.

In practice, users can freely bin the trends into more details according to their special needs. We find that the n-binned trends need $n-1$ more slope angle thresholds, and result in a binary matrix with $n \times q$ columns. For example, if the trends are further binned into three categories ("steep", "medium", and "gentle"), two more slope angle thresholds t'° and $t''^{\circ}(t^{\circ} < t'^{\circ} < t''^{\circ} < 90^{\circ})$ are needed. Thus, S is further binned into a $p \times 3q$ binary matrix S' with "0,0,1" indicating a steep trend, "1,0,0" a gentle trend, and "0,1,0" a medium trend. Hence, $S' = \{g_1, g_2, \ldots, g_p\} \times \{c_1, c'_1, c''_1, c_2, c'_2, c''_2 \ldots, c_q, c'_q, c''_q\}$ such that

Table 5.5: 3-Bin Binary Matrix S'_h: $t' = 35°$, $t'' = 45°$.

S'_h	c_1c_2	$(c_1c_2)'$	$(c_1c_2)''$	c_2c_3	$(c_2c_3)'$	$(c_2c_3)''$	c_3c_4	$(c_3c_4)'$	$(c_3c_4)''$
g_1	1	0	0	1	0	0	1	0	0
g_3	1	0	0	0	1	0	0	0	1

$$S'_{i,j}, S'_{i,j'}, S'_{i,j''} = \begin{cases} 0,0,1 & \text{if } t''° \le |O'_{i,j}| < 90°, \\ 0,1,0 & \text{if } t'° \le |O'_{i,j}| < t''°, \\ 1,0,0 & \text{if } t° < |O'_{i,j}| < t'°. \end{cases}$$

Taking the seed S_h in Figure 5.2 for example, its 3-bin binary matrix S'_h is shown in Table 5.5.

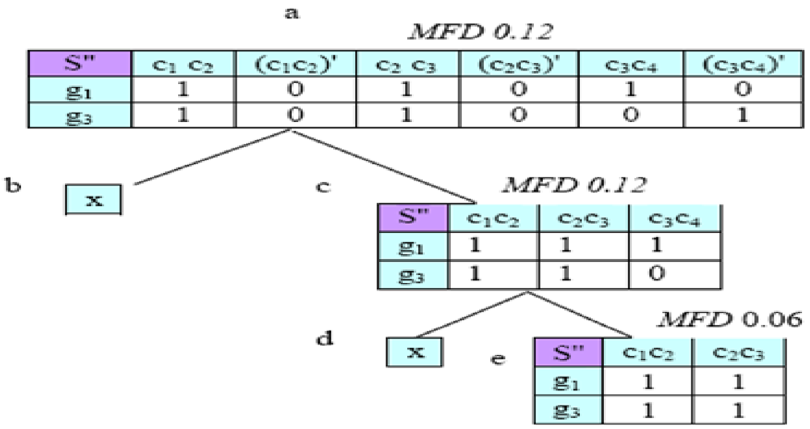
To measure the bicluster similarity, in this phase, we define a new score called *Mean Fluctuation Degree* (*MFD*). *MFD* is calculated from the slope angle matrix O' generated in phase 1. Note that in the calculation of the *MFD*, the entries used in O' are expressed in radians rather than degrees.

Let the submatrix $A \subset O'$ be denoted as a pair (I, J) where $I \subset G$ and $J \subset C'$. Then the *MFD* of A is defined as

$$MFD(I,J) = \sqrt{\tfrac{1}{|I||J|} \sum_{i \in I, j \in J} (O'_{ij} - O'_{Ij})^2} \text{ where } O'_{Ij} = \tfrac{1}{|I|} \sum_{i \in I} O'_{ij}.$$

In a bicluster, if genes have similar degree of fluctuating trends under each condition transition, the *MFD* of the bicluster will be relatively lower. If all genes in a bicluster have exactly the same degree of fluctuating trend under each condition transition, the bicluster's *MFD* is zero. As we have mentioned earlier, no single similarity score is sufficient to ensure that a bicluster will exhibit consistent trends with similar degrees of fluctuations. Hence, we only employ *MFD* as a supplementary merit function to evaluate the bicluster quality and remove biclusters that do not satisfy a user specified maximum *MFD* threshold $maxMFD$.

After re-binning the seeds, the partitioning procedure in phase 2 is applied to each binary seed matrix. The partitioning procedure will then group together genes that have trends with similar fluctuating degrees. In this phase, during each partitioning step, the *MFD* of the resulting submatrix is calculated and checked against the $maxMFD$ threshold. The partitioning process terminates whenever the submatrix has an *MFD* lower than $maxMFD$. All submatrices that satisfy the *minGen*, *minCon* and $maxMFD$ thresholds are returned to users with their original genes and conditions, which are the finial results of our refined biclusters. Figure 5.4 shows the refining procedure of Seed S_h generated by the running example in Figure 5.2. In this example, given $maxMFD$ = 0.10, submatrix e is the finial bicluster with $MFD = 0.06$.



S"	c1 c2	(c1c2)'	c2 c3	(c2c3)'	c3c4	(c3c4)'
g1	1	0	1	0	1	0
g3	1	0	1	0	0	1

S"	c1c2	c2c3	c3c4
g1	1	1	1
g3	1	1	0

S"	c1c2	c2c3
g1	1	1
g3	1	1

Figure 5.4: Phase 3: Refining Process.

5.2.4 Time Complexity

Bicluster Mining has been proved to be NP problem [11]. For the 2D dataset $O = G \times C$, where $|G| = M$, $|C| = N$, the time complexity of *QHB* is $O(2^N \times M)$, without applying any pruning strategy. By applying *minGen*, *minCon* and maximal bicluster constraints, the efficiency of *QHB* is greatly improved due to the early pruning.

5.3 Experimental Results

We implemented our algorithm in C, and evaluated its effectiveness on the Yeast gene expression dataset downloaded from http://arep.med.harvard.edu/biclustering/yeast .matrix. The dataset consists of 2884 genes under 17 conditions, forming a (2884×17) matrix. The number in each entry is obtained by scaling and logarithm $x \rightarrow 100 \log(10^5 x)$ and the result is a matrix of integers in the range between 0 and 600. The experiments are studied on a desktop computer with an Intel Pentium 4 processor and 1 G main memory. We compare our *QHB* scheme against the DBF algorithm in [63]. As shown in [63], the DBF scheme outperforms existing algorithms in terms of quality of biclusters and efficiency.

5.3.1 Data Prepossessing

In the transformation from original matrix O to slope angle matrix O', instead of applying $O'_{i,j} = arctan|O_{i,j+1} - O_{i,j}|$, we take $O'_{i,j} = arctan|\frac{O_{i,j+1} - O_{i,j}}{\delta}|$, where $|\frac{max(O_{i,j})}{\delta}| < 10$. This helps to avoid the distribution of slope angles falling into a narrow range, hence decreasing the sensitivity of binning thresholds. Since the entries in our experimental matrix O are in the range between 0 and 600, we set $\delta = 100$. Figure 5.5 shows the slope angle distribution after matrix transformation.

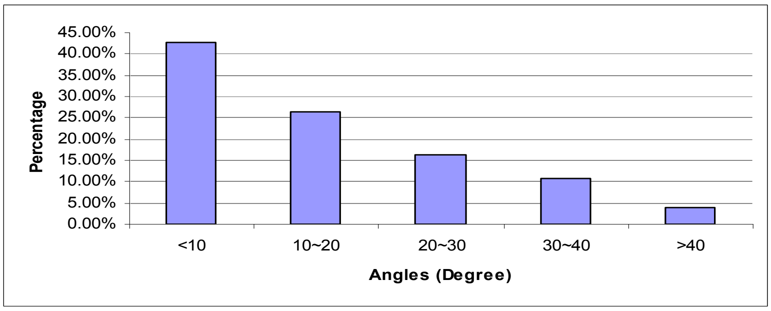


Figure 5.5: Slope Angle Distribution.

Both the QHB and DBF algorithms employ the same data preprocessing phase of getting the angles of fluctuating trends on consecutive condition transitions and grouping similar angles into bins. We set the first angle threshold $t = 10°$ for the two algorithms, so that they have the same input (dataset of density 57.35%) for all experiments. As for bicluster refinement, we set the second angle threshold $t' = 21.5°$ for QHB, so that 27.65% entries are gentle changes and 29.70% entries are steep changes. And we set $maxMSR = 400$ and row variance $\beta = 100$ for DBF. The above parameters are applied to all experiments.

5.3.2 Bicluster Quality Comparison

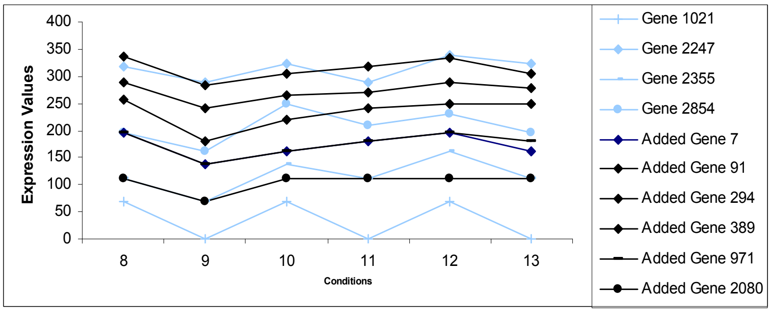
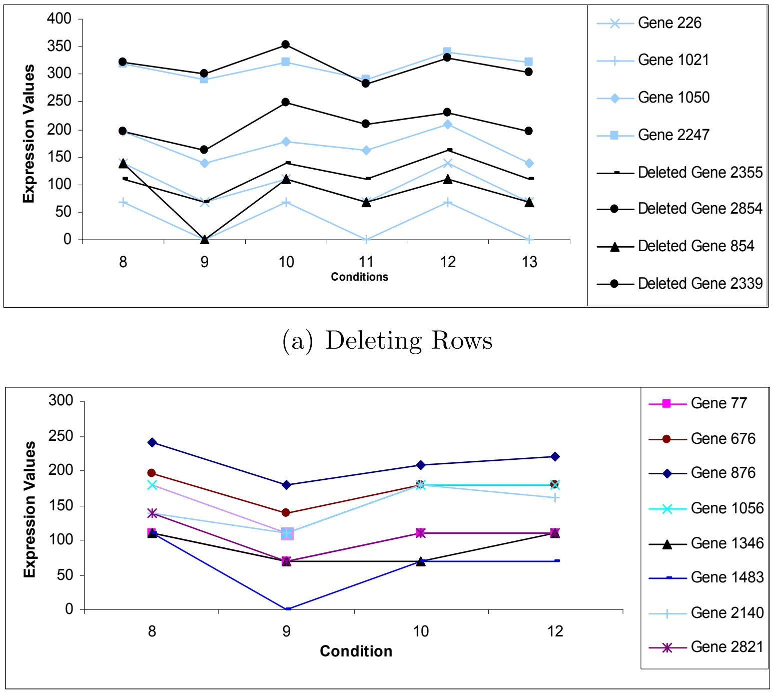


Figure 5.6: Row Adding: the 61th bicluster by DBF.

In the first group of experiments, we compare the quality of biclusters generated



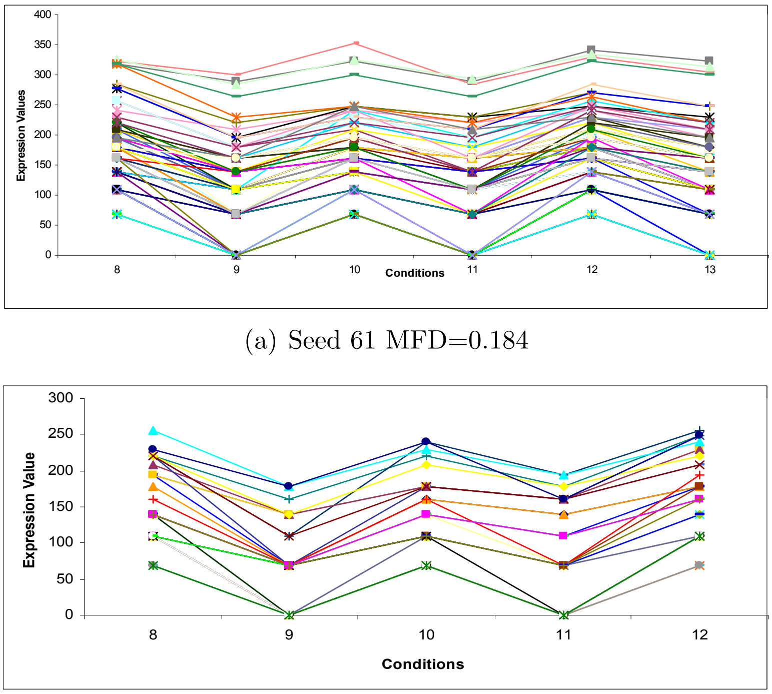
(a) Deleting Rows

(b) Deleting Columns

Figure 5.7: Deleting: the 61th bicluster.

by the two algorithms. We set the minimum gene support $minGene = 60$, minimum condition transition support $minCon = 4$, and $maxMFD = 0.16$. Both algorithms generate the same 4527 biclustering seeds. DBF sorts the seeds by $\frac{MSR}{Volume}$ score increasingly, and takes only the top 100 seeds as qualified seeds for further exploration. In the second phase of DBF, the top 100 seeds are further refined by interactively adding rows/columns if the MSR of each bicluster still satisfies the $maxMSR$ threshold. However, this MSR oriented row/column adding process inevitably destroys the trend consistency of the original seeds. From the biclusters generated by DBF, we find that genes have inconsistent changing trends under condition transitions. Figure 5.6 shows an example of some original genes in Seed 61 and the genes added into Seed 61 by DBF's MSR oriented row addition. It is clear that the additional genes destroy the original similar trends of the whole bicluster.

Previous work based on *MSR* refinement usually integrates row/column deletion to achieve a smaller *MSR* value. Although the deletion scheme is not integrated into DBF framework, we also show examples to demonstrate that the *MSR* oriented deletion will remove good patterns. Figure 5.7 shows an example of *MSR* oriented row/column deletion on the 61th bicluster. Figure 5.7(a) shows a portion of genes retained and removed from the 61th bicluster. Some genes with good changing trends have been removed, which is a great loss of valuable information. Figure 5.7(b) shows a portion of genes finally retained after column deletion. Although the bicluster volume decreases, there still exist inconsistent trends.

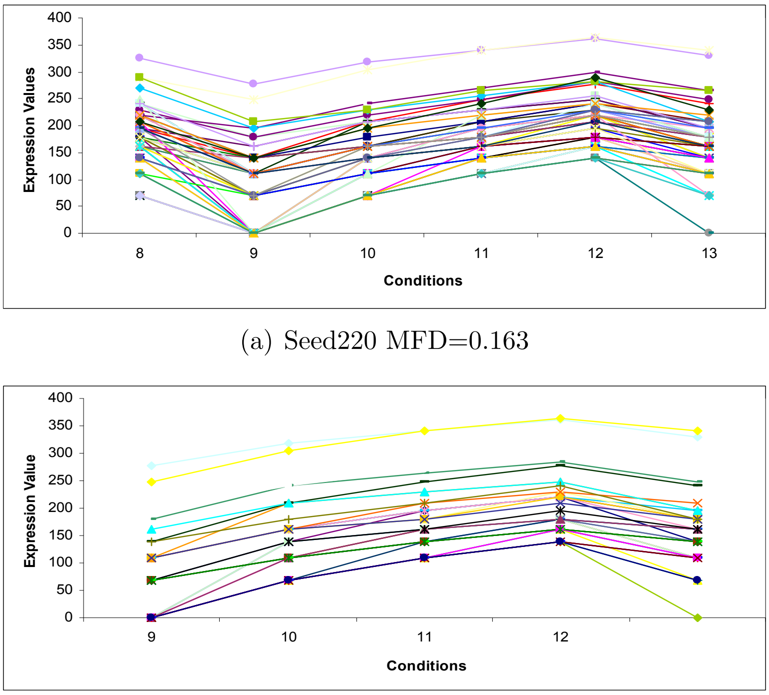


(a) Seed 61 MFD=0.184

(b) Refined Bicluster MFD=0.159

Figure 5.8: *QHB* Refinement.

Hence, we conclude that the *MSR* oriented row/column adding/deleting refinement would destroy the trend consistency of original good seeds. Instead, our *QHB* further refines the seeds to make the trends within a bicluster more similar, without adding any genes with inconsistent trends. Figure 5.8 shows the whole genes in Seed 61 and one of its refined subset bicluster. It is clear that our *QHB* is very effective in enhancing the seed quality by grouping together genes that have trends with more similar fluctuating degrees.



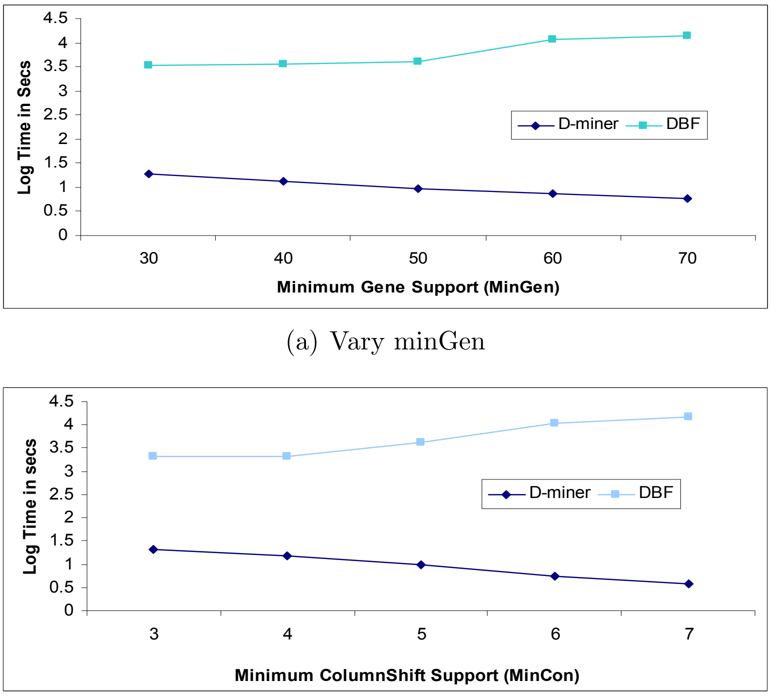
(a) Seed220 MFD=0.163

(b) Refined Bicluster MFD=0.141

Figure 5.9: Seed220: ranking out of top 100.

5.3.3 Information Integrity

In the second group of experiments, we show that DBF may result in information loss. We set $minGene = 40$, $minCon = 4$ and $maxMFD = 0.15$. Recall that DBF only employs the first top 100 seeds. Among the seeds removed by DBF due to the ranking, we pick up Seed 220 to refine it with *QHB*. From Figure 5.9, we find that, after refinement, the bicluster pattern has very consistent changing trends, which should not have been omitted from the whole results. Hence, the $\frac{MSR}{Volume}$ score by DBF may not always be a good criteria to remove seeds. Instead, our algorithm can work on all seeds without missing valuable information with efficiency. The efficiency of *QHB* will be shown next.



(a) Vary minGen

(b) Vary minCon

Figure 5.10: Execution Time.

5.3.4 Efficiency

In this group of experiments, we set $maxMFD = 0.15$ and vary the *minGene* and *minCon* thresholds and compare the execution time of *QHB* against DBF. The execution time for *QHB* includes processing all seeds while the execution time for DBF only includes processing the top 100 seeds ranked by the algorithm. However, compared with DBF, *QHB* is still much more efficient as shown in Figure 5.10 (time calculated in log(Second)). This is because *QHB* simultaneously groups several genes and conditions at the same time and the grouping (submatrix partition) process is oriented by bins. This makes the whole processing very efficient. However, while refining the seeds, DBF tends to randomly try the row/column one by one to decide which row/column to add. This process is very time-consuming.

5.3.5 Hierarchical Structure

One important advantage of *QHB* is that *QHB* can deliver a hierarchical structure of inter-bicluster relationship. Based on the hierarchical structure, users may freely roll up or down to get a more general or detailed insight into biclusters. Figure 5.11 shows an example of seed refining process in a hierarchical structure. The root bicluster is refined further level by level, generating child biclusters with higher degree of similarity in fluctuating trends.

5.3.6 Parameter Study

In the final group of experiments, we study the effects of *QHB* parameters on the number and volume of final results. These parameters include *minGen*, *minCon* and $maxMFD$. From Figure 5.12, we can see that a smaller value in *minGen* or *minCon* will lead to more number of final biclusters. From Figure 5.12(a), we find

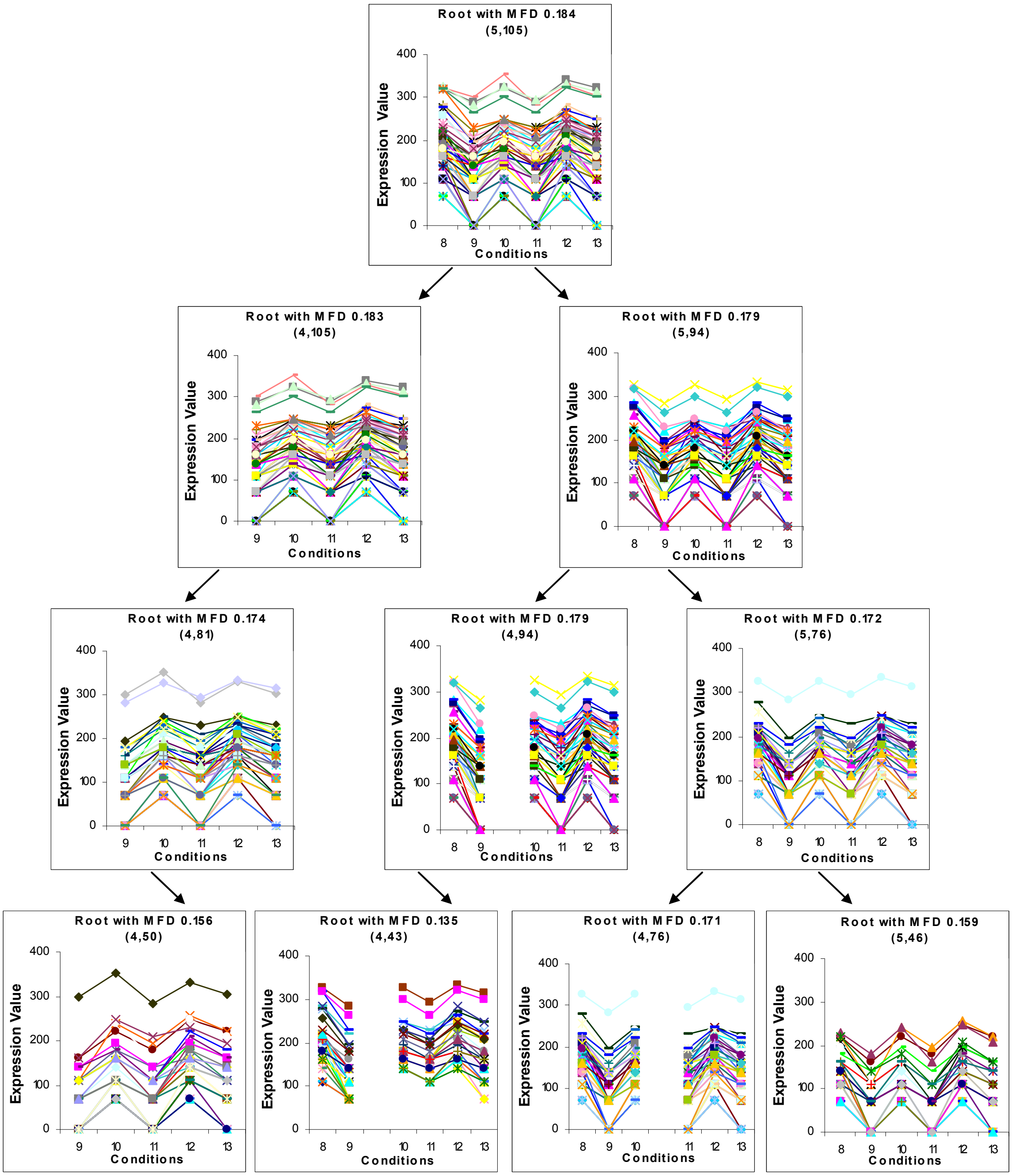
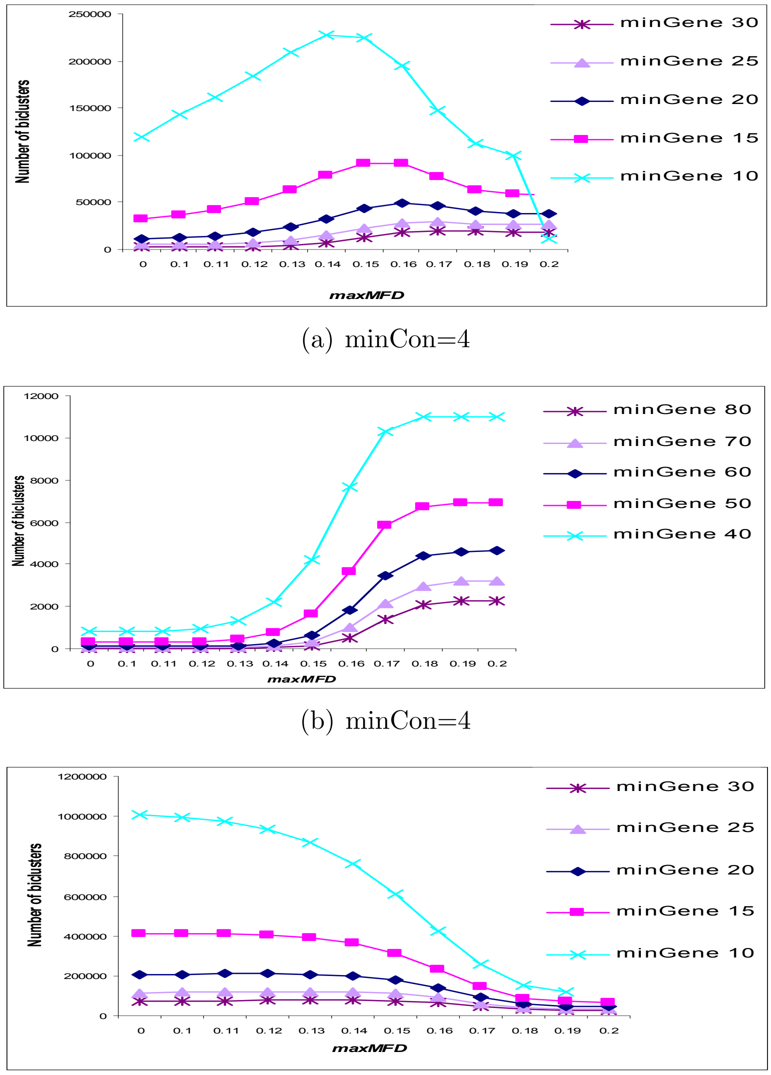


Figure 5.11: Hierarchical Structure.



(a) minCon=4

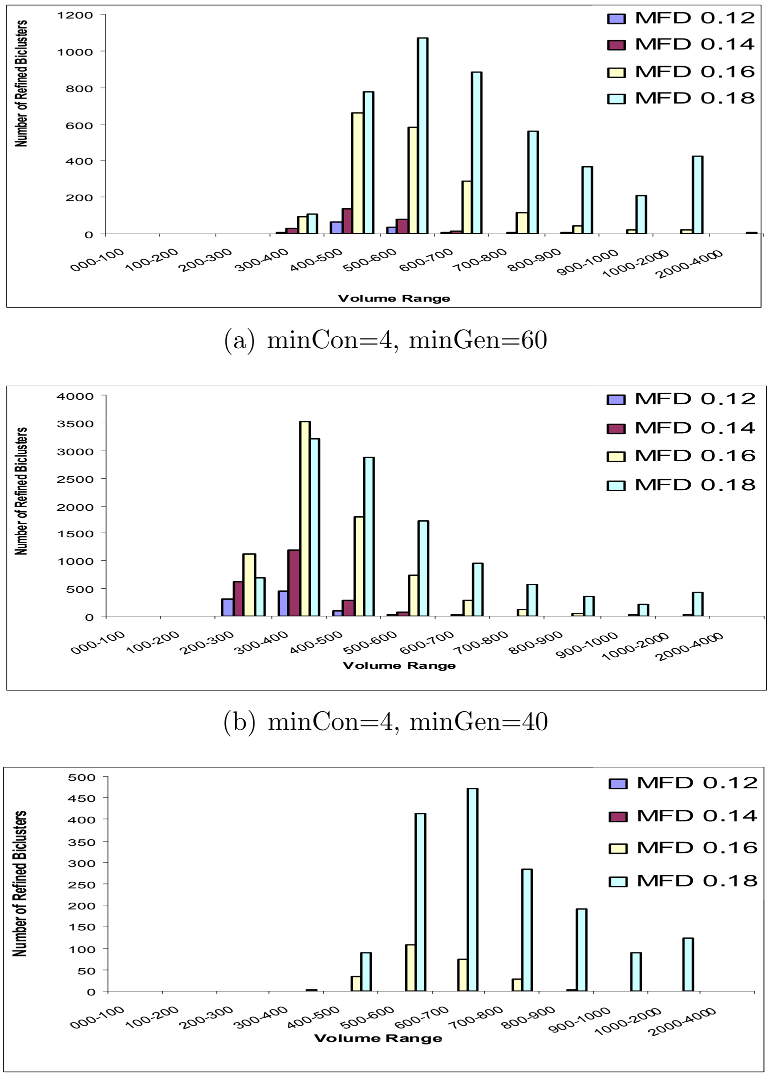
(b) minCon=4

(c) minCon=2

Figure 5.12: Number of Biclusters vs. maxMFD.

that as $maxMFD$ increases, the number of biclusters increases for a while and then begins to decrease. As we know, $maxMFD$ has an effect on the levels to which the biclusters are further refined. A lower $maxMFD$ value will cause the tree to split into deeper levels while a higher $maxMFD$ value will stop the tree's splitting early. While the tree splits into very deep levels, most biclusters cannot satisfy the *minGen* or *minCon* threshold, and hence they are pruned off. This explains why when $maxMFD$ value is very low, the number of biclusters is small. As the $maxMFD$ increases, most biclusters will satisfy the *minGen* and *minCon* thresholds, hence the number of biclusters increases. When $maxMFD$ increases to a certain value, the splitting tree stops partitioning early, hence the number of biclusters decreases again. Figure 5.12(b) and (c) further confirm this point. Based on (a), we increase the *minGen* threshold in (b). We find that the number of biclusters increases as $maxMFD$ increases. This is because the increased *minGen* thresholds prune much more biclusters and have relatively more effect on the number of biclusters in this group of experiments. A higher $maxMFD$ leads to larger volume of biclusters that tend to satisfy the *minGen* threshold. Moreover, based on (a), we decrease the *minCon* threshold in (c). We find that with the increase of $maxMFD$, the number of biclusters decreases. This is because when the *minCon* threshold is low, most of the biclusters will pass the threshold and hence the number of biclusters will be affected more by $maxMFD$ value. Therefore, we conclude that the number of biclusters depends on the co-effects of *minGen*, *minCon* and $maxMFD$.

We also study the bicluster volume distribution with different parameters in Figure 5.13. We find that a higher $maxMFD$ keeps the average bicluster volume larger, while a lower $maxMFD$ decreases the average bicluster volume. And a more constraint threshold (higher value) on $minGen$ or $minCon$ will prune off more small volume biclusters.



(a) minCon=4, minGen=60

(b) minCon=4, minGen=40

(c) minCon=5, minGen=60

Figure 5.13: Bicluster Volume Distribution.

Table 5.6: Known Co-regulated Genes from Biclusters.

M/G1 Boundary Regulated:
CDC47(YBR202W), CHS1(YNL192W), STE2(YPR122W)
CTS1(YLR286C), STE2(YPR122W)
ASH1(YKL185W), STE2(YPR122W)
SIC1(YLR079W), SST2(YLR452C), CHS1(YNL192W), STE2(YPR122W)
Late G1(MCB) Regulated:
PDS1(YDR113C), RFA1(YAR007C)
PDS1(YDR113C), ASF1(YJL115W)
PDS1(YDR113C), SRS2/HPR5(YJL092W)
PDS1(YDR113C), CLB6(YGR109C), SPK1(YPL153C)
PDS1(YDR113C), CLB6(YGR109C), PMS1(YNL082W)
PDS1(YDR113C), CLB6(YGR109C), GIC2(YDR309C)
PDS1(YDR113C), CLB6(YGR109C), RAD27(YKL113C), CDC21(YOR074C), DPB2(YPR175W)
CLB6(YGR109C), ASF1(YJL115W), CDC21(YOR074C), DPB2(YPR175W)
S/G2-phase Regulated:
NUM1(YDR150W), CWP2(YKL096W-A)

5.3.7 Biological Significance

QHB is able to identify known co-regulated genes already established by biologists. In the final experiment, we set $minCon = 4$, $minGen = 50$, and $maxMFD = 0.1$, and get 604 biclusters. Table 5.6 shows some M/G1 Boundary co-regulated genes, Late G1(MCB) regulated genes, and S/G2-phase co-regulated genes identified from our results. From the 604 biclusters, 288 biclusters contain the known co-regulated genes. That is, 47.68% of the resulting patterns are of biological significance regarding to the known gene co-regulations established by the biological works [38, 8] already.

Table 5.7: Non-consecutive Slope Angle Matrix O'.

O'	c_1c_2	c_1c_3	c_1c_4	c_2c_3	c_2c_4	c_3c_4
g_1	$28.81°$	$2.86°$	$30.54°$	$-26.57°$	$28.37°$	$28.37°$
g_2	$-13.50°$	$-43.53°$	$-58.93°$	$-35.37°$	$-35.37°$	$-35.37°$
g_3	$30.96°$	$-6.84°$	$46.67°$	$-35.75°$	$49.72°$	$49.72°$

5.4 Non-consecutive Conditions Adaptation

So far, we have focused on biclusters with changing trends under consecutive condition transitions. However, this may be extended to other datasets whereby non-consecutive condition transitions are to be considered as well. In such cases, the combination of any two conditions should be considered. Given an $m \times n$ matrix $O = G \times C$, its non-consecutive slope angle matrix is an $m \times \frac{n\times(n-1)}{2}$ matrix $O' = G \times C'$ such that $C' = \{c_1c_2, c_1c_3, \ldots, c_1c_n, c_2c_3, \ldots, c_{n-1}c_n\}$ and $O'_{i,jk} = arctan|O_{i,k} - O_{i,j}|$. Given the running example O in Table 5.1, its transformed non-consecutive slope angle matrix O' is shown in Table 5.7.

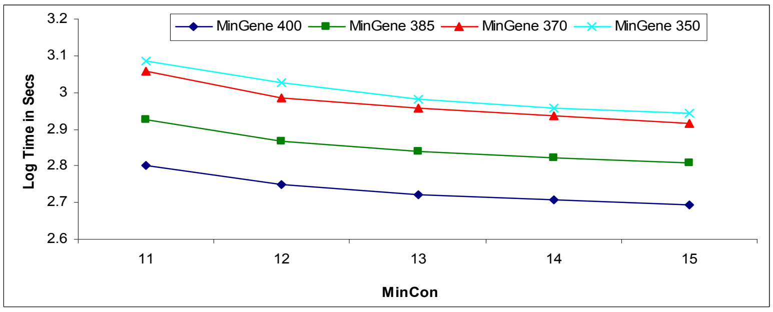


Figure 5.14: Execution Time: Non-consecutive Biclustering.

Again this angle matrix can be binned into the binary matrix and mined by the same partitioning methods described in Phase 2 and Phase 3 of *QHB*. We still take the same yeast gene dataset for experiments. We set $maxMFD = 0.15$ and vary the *minGen* and *minCon* thresholds to test the execution time. Figure 5.14 shows the execution time of mining biclusters with non-consecutive condition transitions. From the results, we find that the executive time decreases with the increase of *minGen* and *minCon*. And the adapted *QHB* is still efficient to mine biclusters under non-consecutive condition transitions. Figure 5.15 illustrates an example of the bicluster with non-consecutive condition transitions.

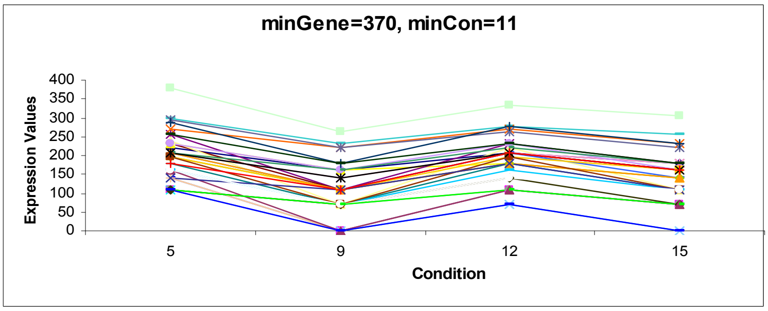


Figure 5.15: Bicluster with Non-consecutive Condition Transitions.

5.5 Summary

Mining biclusters that exhibit both consistent trends and trends with similar degrees of fluctuations is vital to bioinformatics research. In this chapter, we have re-examined how biclusters are extracted from the gene expression data and introduced our framework *QHB* to ensure that the final bicluster trends are not only consistent but exhibit similar degrees of fluctuation between consecutive conditions. We have also provided a new merit function that gauges the degree of similarity in the fluctuations of the bicluster, enabling us to extract biclusters that fulfill this condition and filter off those that have a wide range of degree fluctuations. As shown in our experiments, our framework is able to efficiently mine biclusters of a better quality, compared with the more recent DBF framework. Furthermore, *QHB* provides a hierarchical picture of inter-bicluster relationships, maintains information integrity and offers users a progressive way of knowledge exploration. We also show that some known interesting co-regulated genes are found in our results. All the above features of *QHB* make it an attractive tool for microarray data analysis.

Chapter 6

Time-Lagged Clustering on 2D Expression Data

6.1 Overview

In the last few chapters, we have seen the application of frequent closed pattern mining techniques to identify co-attribute patterns, and biclustering techniques to mine co-tendency patterns. However, these techniques usually consider gene expression levels in the same conditions or time points but do not take the time-lagged relationships into consideration. In fact, for time series gene expression data, most genes do not regulate each other simultaneously, but after a certain time lag. That is, the products that a gene produces during expression process may affect other genes' expression later. Such regulations can be divided into two types: activation and inhibition. In the activation process, an increase in certain genes' expression levels will increase some other genes' expression levels after a certain time lag. Conversely, during the inhibition process, an increase in some genes' expression levels will result in a decrease in other genes' expression levels accordingly.

In this chapter, we design a new algorithm to identify *localized* time-lagged co-regulations between genes/gene clusters efficiently. Since the gene co-regulations include both "activation" and "inhibition", we consider time-lagged patterns with *opposite* changing tendency as well. Our approach is to extract clusters, which we referred to as *q-clusters*, of (time-lagged) co-regulated genes over a subset of q consecutive conditions. Each such cluster essentially contains information of genes that have similar expression pattern over q consecutive conditions (the q conditions may be different for different genes). These information include the *(geneID, st)*-pairs that indicate that the gene with identifier *geneID* has the corresponding pattern of the *q-cluster* starting from the time point *st*. In our work, the pattern of a *q-cluster* is represented as a string of $(q - 1)$ *changing tendency* that reflects how the expression value changes from condition i to condition $i + 1$ for the q conditions. We have discretized the changing tendency into 3 distinct classes. Thus, there are in total 3^{q-1} *q-clusters*, and each *q-cluster* can be easily mapped to a unique value, *q-value*, based on the q conditions, where $0 \leq$ *q-value* $\leq 3^{q-1}$.

Now, we can determine the following types of co-regulations from each *q-cluster*:

- All genes with the same start time point may be co-regulated.
- All genes with start time point st_1 may activate those genes with start time point st_2 where $st_1 < st_2$.

Moreover, we can determine the following co-regulations/ inhibitions across *q-clusters*:

- All genes with start point st_1 from *q-cluster* Q_1 may inhibit the genes with start point st_2 from *q-cluster* Q_2 if the expression pattern of Q_1 is complement to that of Q_2 (i.e., the changing tendency of the q conditions in Q_1 is opposite those of Q_2).
- All genes with start point st_1 from *q-cluster* Q_1 may co-regulate the genes with

start point st_2 from *q-cluster* Q_2 if the expression pattern of the q conditions of Q_1 is *similar to* (but not exactly the same as) that of Q_2.

Moreover, since we keep track of the start time points, we can easily determine the detailed information of the interacting portions, e.g., how far one gene lags behind another.

The rest of this chapter is organized as follows. In the next section, we present the proposed *q-cluster* algorithm to identify time-lagged gene clusters. In Section 6.3, we compared our scheme with the *Event Method* ([34]) on the time series Yeast gene dataset, and finally, we conclude in Section 6.4.

6.2 Algorithm to Identify Time-Lagged Gene Clusters

In this section, we propose to identify localized time-lagged gene clusters. We develop an algorithm *q-cluster* that can quickly determine a set of genes that co-regulate either simultaneously or after some time lag, as well as genes that may inhibit others. Our basic idea is to group genes with similar patterns over a subset of consecutive time points (conditions) together. Because these genes share similar (or opposite) patterns (over a subset of conditions), those with earlier start time may have activated (inhibited) those with later start time. The scheme comprises three phases. In the first phase, the original gene expression matrix is transformed to a "*slope*" matrix to reflect the genes' changing tendency along time. In phase two, we generate *q-clusters* that contain information of genes with similar pattern over (any) q consecutive conditions. Finally, in phase three, the time-lagged information is extracted from each *q-cluster* and between *q-clusters*. Algorithm 7 presents a summary of the

Algorithm 7 *q-cluster*

```
q-cluster()
Global variables: $O''$ transformed matrix, $O'$ binned matrix, $\beta$ binary sequence of length $q-1$, $\delta$ q-clusterID, $(geneID, st)$-pair and $maxZero$ the maximum zeros allowed in the pattern.
Input: 2D Matrix $O$ with $n$ rows and $m$ columns.
Output: $Q$ time-lagged q-clusters.
Initialization:
$Q \leftarrow \emptyset$;
for $k = 0$; $k <= 3^{q-1}$; $k++$ do
    q-cluster($k$) $\leftarrow \emptyset$;
end for
Phase 1:
$O'' \leftarrow transform(O)$;
$O' \leftarrow bin(O'')$;
Phase 2:
for $i = 0$; $i < n$; $i++$ do
    for $j = 0$; $j <= m - q$; $j++$ do
        $\beta \leftarrow SlidingWindow(O'_{i,j}, O'_{i,j+q-2})$;
        if $ZeroNumber(\beta) <= maxZero$ then
            $\delta \leftarrow Hash(\beta)$;
            q-cluster($\delta$)$\leftarrow$ q-cluster($\delta$)$\cup(i,j)$-pair
        end if
    end for
end for
Phase 3:
for $k = 0$; $k <= 3^{q-1}$; $k++$ do
    if q-cluster($k$)$\neq \emptyset$ then
        $Q \leftarrow Q\cup$Sort(q-cluster($k$), $st$);
    end if
end for
return $Q$;
```

whole process. $SlidingWindow(O'_{i,j}, O'_{i,j+q-2})$ is the function to scan from the starting position $O'_{i,j}$ to the ending position $O'_{i,j+q-2}$ while $Sort$(q-cluster(k), st) is the function to sort the $(geneID, st)$-pairs within q-cluster(k) by the st value.

6.2.1 Phase 1: Matrix Transformation

Let $T = \{T_1, T_2, \ldots, T_m\}$ be the set of time points, and $G = \{G_1, G_2, G_3, \ldots, G_n\}$ be the set of genes. The time series gene expression data can be represented as a $O = n \times m$ matrix, where entry $O_{i,j}$ in this matrix corresponds to the expression value of gene G_i on time point T_j. In the first phase, matrix O is transformed into a $O' = n \times (m-1)$ matrix to reflect the changing tendency of each gene expression value along time. Each entry $O'_{i,j}$ in matrix O' reflects the directional change from the expression value $O_{i,j}$ to the expression value $O_{i,j+1}$. Essentially, there are three possible changing tendencies: an expression value may increase from time point T_j to T_{j+1}; it may decrease; or it may remain unchange. As we shall see shortly, we discretize these three changing tendencies into three classes, and denote them by 1, -1 and 0 respectively.

The matrix O' is obtained in two steps. In the first step, O is transformed into a $O'' = n \times (m-1)$ matrix such that

$$O''_{i,j} = \begin{cases} \frac{O_{i,j+1} - O_{i,j}}{|O_{i,j}|} & \text{if } O_{i,j} \neq 0, \\ 1 & \text{if } O_{i,j} = 0 \ \& \ O_{i,j+1} > 0, \\ -1 & \text{if } O_{i,j} = 0 \ \& \ O_{i,j+1} < 0, \\ 0 & \text{if } O_{i,j} = 0 \ \& \ O_{i,j+1} = 0. \end{cases}$$

O'' essentially indicates how much a gene's expression value changes from one time point to the next (a positive value implies an increase, a negative value a decrease, and 0 means unchange). Once matrix O'' is generated, in step 2, we can obtain O'

Table 6.1: Original Matrix O.

Gene/Time	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
G2163	-0.44	-0.44	0.08	0.35	0.26	0.17	-0.46	-0.13	-0.05	-0.36
G1223	-1.51	-1.57	-1.35	0.04	1.3	1.15	0.94	-0.08	-0.13	-0.72

Table 6.2: Binned Slope Matrix O'.

Gene/Time	T1T2	T2T3	T3T4	T4T5	T5T6	T6T7	T7T8	T8T9	T9T10
G2163	0	1	1	0	0	-1	0	0	-1
G1223	0	0	1	1	0	0	-1	0	-1

by binning the values of the transformed matrix. Binning the values is a good way to handle noise that may be introduced by experimental errors. Moreover, it allows us to focus on the more general increasing or decreasing tendency of gene values. We set a *Normalization Threshold* $t(t > 0)$ to bin the new matrix as follows:

$$O'_{i,j} = \begin{cases} 1 & \text{if } O''_{i,j} \geq t, \\ -1 & \text{if } O''_{i,j} \leq -t, \\ 0 & \text{otherwise.} \end{cases}$$

As an example, let's take two genes from the Yeast dataset: YGL207W (G2163) and YDR224C (G1223). The original matrix O of their expression values in the first ten time points are shown in Table 6.1; and the resultant binned slope matrix O' with a *Normalization Threshold* $t = 1.0$ is shown in Table 6.2.

6.2.2 Phase 2: Generation of q-clusters

We note that each sequence of "-1", "0" and "1" in matrix O' provides us with an indication of the changing pattern of a gene expression over time. Thus, two genes that share the same subsequence may be co-regulated. In this phase, we generate a set of *q-clusters*. Each *q-cluster* has the following property: all genes in the cluster have the same expression pattern over some q consecutive time points (conditions). This turns out to be none other than finding genes that share similar subsequences of length $q-1$. We note that q is a user-defined parameter. Since entries of O' have only 3 possible distinct values, there are at most 3^{q-1} *q-clusters*. Each *q-cluster* has a unique identifier, called *q-clusterID* which is generated as follows. Let $P = \{p[1], p[2], \ldots, p[q-1]\}$ be a pattern. Note that $p[i] = -1,\ 0$ or $1\ \forall i \in [1, q-1]$. Let

$$f(p[i]) = \begin{cases} p[i] & \text{if } p[i] = 0, \\ p[i] & \text{if } p[i] = 1, \\ 2 & \text{if } p[i] = -1. \end{cases}$$

Then, the q-clusterID of P is determined as follows:

$$q - clusterID(P) = \sum_{i=1}^{q-1} f(p[i]) * 3^{q-1-i}$$

Clearly, $0 \leq$ *q-clusterID* $\leq 3^{q-1}$. We note that a small value of q will result in a small number of *q-clusters* but there are also likely to be more genes with the same (sub)patterns. On the contrary, a large value for q implies a larger number of *q-clusters* with fewer genes with the same patterns.

We are now ready to describe how q-clusters are generated. For each row (gene) of matrix O', we apply a sliding window of length $(q-1)$. As each $(q-1)$-substring is examined, its q-clusterID is determined, and the *(geneID, st)*-pairs are inserted to the corresponding *q-cluster*. Here, *geneID* is the gene identifier of the gene and *st* is the position of the start time point of the $(q-1)$-substring. For example, suppose we set $q = 7$. Consider gene YGL207W (G2163) again, which has the sequence "01100(−1)00(−1)". By applying a sliding window of length 6 $(= 7-1)$, we have the

Table 6.3: q-clusters.

Pattern	q-clusterID	Gene information
. . . . . .	. . . . . .	. . . . . .
0 0 1 1 0 0	36	(1223,1)
. . . . . .	. . . . . .	. . . . . .
0 0 -1 0 0 -1	56	(2163,4)
. . . . . .	. . . . . .	. . . . . .
0 1 1 0 0 -1	110	(2163,1) (1223,2)
. . . . . .	. . . . . .	. . . . . .
1 0 0 -1 0 0	261	(2163,3)
1 0 0 -1 0 -1	263	(1223,4)
. . . . . .	. . . . . .	. . . . . .
1 1 0 0 -1 0	326	(2163,2) (1223,3)

subsequence “01100(-1)” in the first window. Now, the q-clusterID(“01100(-1)”)= 110. Thus, we have $(2163, 1)$ inserted into *q-cluster* 110. Similarly, examining the second pattern “1100(-1)0” results in $(2163, 2)$ being inserted into *q-cluster* 326. Table 6.3 shows the *q-clusters* generated by the two genes YGL207W (G2163) and YDR224C (G1223).

From the set of *q-clusters*, we can extract gene co-regulations in three aspects. First, each *q-cluster* corresponds to an interesting pattern under which genes with similar expression pattern gather together. In fact, we can determine two relationships here. For those genes with the same start time point, they may be co-regulated simultaneously. Such a set of genes and conditions actually form a bicluster ([11]). Examples will be given when we look at the next phase. For those genes with different pattern starting positions, those with smaller starting positions may be activators of those with larger starting positions. For example, *q-cluster* 110 in Table 6.3 gathers together Gene2163 and Gene1223 whose pattern starting positions are different by 1.

This implies that Gene2163 may have activated Gene1223 after 1 time point. Second, to handle noise, it may be necessary to look for patterns with approximate match (rather than exact match as in the above case). For example, if "100(−1)00" (third window of Gene2163) is considered similar to "100(−1)0(−1)" (fourth window of Gene1223), we can determine relationships between genes in *q-clusters* 261 and 263. Third, we can also determine inhibition relationships between genes by comparing *q-clusters* with opposite patterns (where "1" is the opposite of "-1"). For example, "100(−1)0(−1)" is the opposite of "(−1)00101".

Besides capturing all relationships between genes/gene clusters, our approach also allows several genes to be simultaneously compared rather than the existing "two genes one relationship" approaches. Moreover, our *q-clusters* can deliver more detailed but concise information. This explains why our scheme works more efficiently and effectively compared to previous methods.

6.2.3 Phase 3: Generate Time-Lagged Co-regulated Relationships Between Genes/Genes Clusters

At the end of phase 2, we have a set of *q-clusters*. In phase 3, four main processing tasks are carried out on the *q-clusters* to extract (time-lagged) co-regulated relationships between genes/genes clusters. For efficiency, each *q-cluster* is first sorted on the starting position, so that all *(GeneID,st)*-pairs with the same starting position *st* are grouped together.

The first task is the mining of biclusters. According to the characteristics of a *q-cluster*, all genes with the same starting position share the same pattern under the same q conditions. Hence, the subset of genes and conditions essentially form a bicluster[11]. Let's take *q-cluster* 551 for example. As shown in Table 6.4, Gene906,

Table 6.4: Q-Cluster 551 for Gene Pattern (-1) 0 (-1) 1 0 (-1).

Starting Position	Gene Identifier
15	580 836 1681 4516
16	679 1308 1527 1622 1875 2045 4448 5222 6049
17	906 1518 1811 2704 5535 5758

Table 6.5: Q-Cluster 289 for Gene Pattern 1 0 1 (-1) 0 1.

Starting Position	Gene Identifier
10	868 968 1254 1434 1609 1973 2256 2330 4064 5733
14	3962 4210 4378 5415 6118
15	320 321 344 393 419 1699 6147

Gene1518, Gene1811, Gene2704, Gene5535, and Gene5758 with the same pattern starting position 17 form a bicluster, i.e., as shown in figure 6.1, these genes have similar changing tendency from T_{17} to T_{23}. Similarly, we can find a bicluster for the set of genes that share the same starting position at time point 15 (see figure 6.2). To draw additional relationships among biclusters, we can carry out task two to identify the promising activation co-regulations and task three to identify the inhibition regulations.

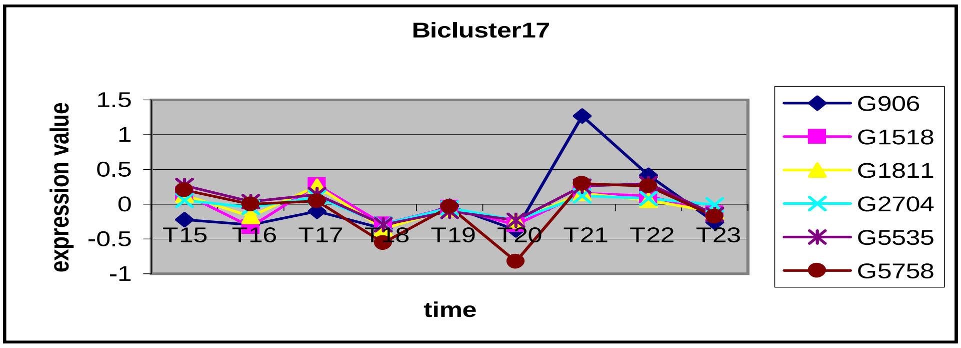


Figure 6.1: Bicluster 17.

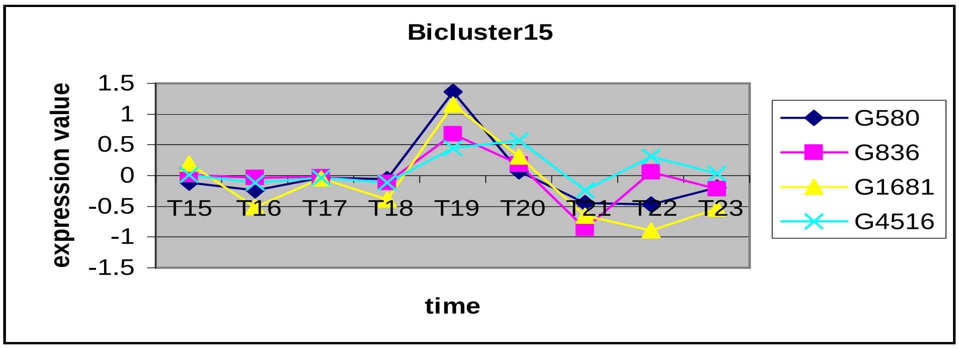


Figure 6.2: Bicluster 15.

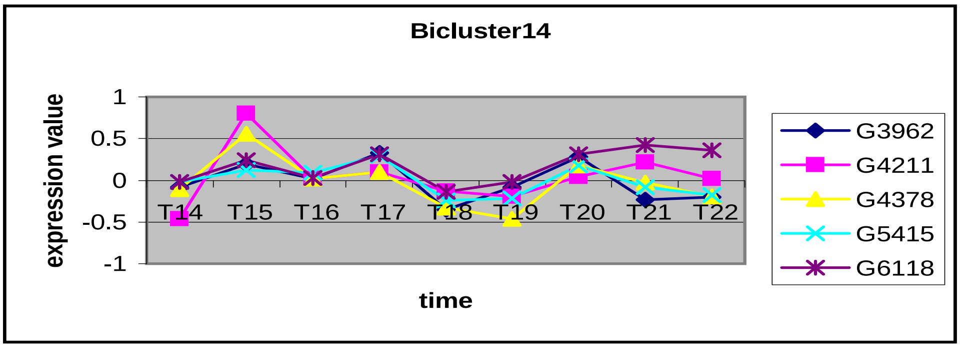


Figure 6.3: Bicluster 14.

Task two deals with gene relationships within a *q-cluster* by comparing the starting positions of biclusters obtained from the *q-cluster*. Since biclusters (within a *q-cluster*) with different starting positions share the same pattern, there is a promising time-lagged activation co-regulation relationships between these biclusters. In particular, given two biclusters, the one with the smaller starting position is a potential activator of the bicluster with the larger starting position. The time lag between the two activations is given by the difference in the starting positions. We note that there are two possible relationships that need further biological study: (a) it may be the case that only certain of the genes in one bicluster individually activates another gene in the other bicluster; (b) it may be the case that all or most of the genes in a bicluster collectively activate some (or all) genes in the other bicluster. As an example, within *q-cluster* 551 in Table 6.4, Gene580, Gene836, Gene1681, and Gene4516 with starting position 15 form a bicluster (in figure 6.2) which is a promising activator (either individual gene or combinations of the genes) to the bicluster (in figure 6.1) with starting position 17 at a time lag of 2.

Task three attempts to find inhibition regulations. This task is quite straightforward. Essentially, we need to first find a pair of *q-clusters* with opposite patterns. Such a pair of *q-clusters* is a promising inhibition pairs. Two patterns are opposite to one another if the corresponding elements between the two patterns are either both “0” or opposite to one another, and element “1” is opposite to “-1”. Genes/biclusters of one of the *q-cluster* with a smaller start position may inhibit genes/biclusters of the other *q-cluster* with a larger start position. For example, the pattern “-1 0 -1 1 0 -1” (*q-cluster* 551) is the opposite of “1 0 1 -1 0 1” (*q-cluster* 289). Thus, the pair of *q-clusters* (551, 289) is a promising inhibition pairs. Genes/biclusters within *q-cluster* 289 have the promising time-lagged inhibition regulations with those within *q-cluster* 551. Gene3962, Gene4210, Gene4378, Gene5415, and Gene6118 with starting position 14 in Table 6.5 forms a bicluster (in figure 6.3) which is the promising inhibitor to the bicluster (Gene906, Gene1518, Gene1811, Gene2704, Gene5535, Gene5758 in figure 6.1) with starting position 17 (see Table 6.4) at a time lag of 3.

Finally, task four handles approximate matching. Similar/opposite patterns with only one or two elements’ exception may still be regarded as interesting by some users. Our scheme is able to deal with this approximation as follows. For each *q-cluster*, we allow changes to be made to certain positions of the pattern. The corresponding *q-cluster* of the changed pattern is potentially a candidate for co-regulation. For inhibition regulation, we only need to find the *q-cluster* that has an opposite pattern from the changed pattern.

Before leaving this section, we would like to make one final observation. Since “0” indicates no obvious increasing or decreasing changing tendency, patterns with too many “0” are not interesting enough to be figured out. As such, in our algorithm, we have introduced another user-specified parameter, *Maximum Zero*, to control the maximum number of “0” allowed in interesting patterns. This implies that the number of “useful” *q-clusters* are fewer than 3^{q-1}.

Compared with the previous methods, our algorithm is more efficient at identifying both activation and inhibition relationships between co-regulated genes. And it also simplifies the identification of approximating patterns. As for the results, our algorithm provides a clear time-lagged relationship structure between genes and gene biclusters. Moreover, the results contain all user needed information with concise structure. Through our results, users can know exactly the starting point and ending point of the co-regulation period. And they can even know how many times two genes co-regulate with each other by counting how many *q-clusters* contain both of them within the user permitted time lag (illustrated in next Section). Depending on the information delivered by our results, deeper exploration can be made focusing on interesting genes/biclusters according to users’ needs.

6.2.4 Time Complexity

The time complexity of *q-cluster* mainly depends on the hash table construction. For the 2D dataset $O = G \times T$, where $|G| = N$, $|T| = M$, the time complexity of hash table construction is $O(N \times M)$, without applying any constraint-based pruning.

6.3 Experimental Results

We implemented our algorithm in C, and studied the time-lagged gene clusters obtained. As reference, we compare our results with the results generated by the *Event Method* [34]. All the experiments are run on a Pentium 4 PC with 256 MB RAM.

6.3.1 Experimental Setup

For our experiments, we employ *Spellman's* data set (source is downloaded from http://genome-www.stanford.edu/cellcycle/data/rawdata/). The data set contains all the data for the *alpha* factor, *cdc15*, and *elutriation* time courses. Further, it includes the data for the *Clb2* and *Cln3* induction experiments. Finally it includes the analysis of the data from [17]. We used only the *alpha*-factor and *CDC28* data sets for our experiments as the *Event Method* did. The data set we used contains 6178 genes under 35 time points, forming a (6178×35) matrix (http://www.comp.nus.edu.sg/ jiliping/p2/dataset.txt).

For the proposed algorithm *q-cluster*, the matrix is transformed into a (6178×34) slope matrix and then binned with the $NormalizationThreshold = 1.0$. We generated *q-clusters* for $q = 7$. We also set the maximum number of "0" allowed in the pattern, $MaximumZero$, to 3.

For the *Event Method* [34] we first encode the "$-1, 0, 1$" into "*F, C, R*" (representing "*Falling, Constant, Rising*" status) respectively, and then apply the *Needleman-Wunsch* algorithm (source is downloaded from http://neobio.sourceforge.net) to align all gene pairs. The *Needleman-Wunsch* alignment algorithm uses the score system to sort the gene pairs. Gene pairs with relatively high score are regarded as promising pairs. We set up the scoring matrix according to the idea of *Event Method.* As shown

Table 6.6: Scoring Matrix Used in Event Model.

Event	R	C	F	Deletion Penalty
R (Rising)	3	0	-2	-1
C (Constant)	0	0	0	-1
F (Falling)	-2	0	3	-1
Insertion Penalty	-1	-1	-1	1

in Table 6.6, the matrix is a form of similarity matrix used to evaluate how well two gene expression profiles match. Insertion penalties are specified by the last row while deletion penalties are located at the last column, which are equivalent to the time delay penalty. The time delay penalty is considered for the fact that if two genes' regulation is too far apart from each other, it is unlikely that they reflect a regulatory relationship. According to the original paper, the top-10000 ranking pairs form the interesting results. In our study, we take the top-12744 ranking pairs as the last 4529 pairs have the same score.

6.3.2 Comparative Study

We run the proposed algorithm and the Event Model on the data set. From the results, we made several interesting observations. First, our method can identify the relationships between gene pairs detected by the *Event Method*. Among the top-12744 ranking pairs generated by the *Event Method*, 98.9% are detected within the same *q-cluster* of our results. In addition, our approach can provide more detailed information. Consider, for example, the co-regulated gene pairs YGL207W (Gene 2163) and YDR224C (Gene 1223). The *Event Method* only gives the score of the alignment, as shown in Table 6.7. Our method not only identifies their relationship, but also shows that there are two basic regulated periods between YGL207W and YDR224C.

Table 6.7: Alignment for Event Method.

```
C-RRCCFCCF-RCCCRCFCFCRCFFCCRRCCFRRFC - YGL207W
CCRRCCF-CFCCR-CCCCF-FCRCCFCFCRCFFRCCC - YDR224C
Score: 30
```

As shown in Table 6.8, the first time lag is 1 with the pattern "01100(-1)" while the second time lag is 7 with the pattern "0(-1)0(-1)01". The whole sequences of the two genes are presented in Figure 6.4, which clearly shows the time lag relationship between the patterns of the two sequences.

Table 6.8: Q-Clusters for patterns 01100(-1) and 0(-10)0(-1)01.

01100(-1)	
1	594 969 1506 (2163) 3035
2	390 842 (1223) 1296 2730 3289 3640 4184 4746 4997 5379 5543 5544 6115
0(-1)0(-1)01	
15	23 234 629 1010 1035 1751 1874 1906 (2163) 2234 2235 2565 2747 2782 2814 3146 3346 3448 3640 4321 4393 5539
22	171 291 757 907 942 1075 (1223) 1224 1326 1344 1398 1416 1578 1704 2003 2218 2280 2377 2409 2412 2424 3470 3478 3704 3710 3786, 3820 3954 3986 4058 4104 4187 4392 4667 4746 4786 4826 5069 5327 5861 5925 5936

Second, *Event Method* may not always provide the correct ranking order between gene pairs. In other words, it is possible for a truly time-lag co-regulated gene pairs to be ranked lower than a gene pair that has no co-regulation relationship. Given the large number of results (e.g., 10000), it is likely that some of the truly co-regulated pairs be missed out. For example, Gene YHR200W and YJL115W are known co-regulated gene pairs while Gene YHR200W does not have any co-regulation with

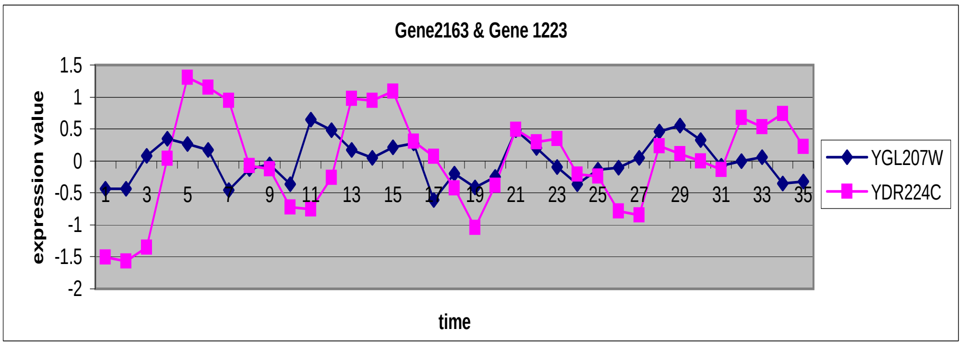


Figure 6.4: Gene2163 and Gene1223.

Table 6.9: Scores of Event Method.

`CCFRFRF---RR-FC-FCCRR-FCCCCRCFCRRCFFRRCC` -YHR200W

`-RCRCCFCCCRRCFFCFCRRRCF--CFCCFCRCCFF---C` -YJL115W

Score: 27

`--FRFRFRRCFRFCCRCFCFCCFR-F-RCFRCCFRFRF-` -YGR282C

`CCFRFRFRR-FCFCCRRF-CCCCRCFCR--RCFFR-RCC` -YHR200W

Score: 44

Gene YGR282C. However, the *Event Method* ranks the latter pair higher than the former one, as shown in Table 6.9. Moreover, there is actually one more similar pattern (with one element approximation) in the former pair than the latter one, as shown in Table 6.10. Our method can detect this information with ease.

Third, our results are complete, containing more information with more concise

Table 6.10: Similar Patterns.

CCFRFRFRR(*FCFCCRR*)FCCCCR(*CFCRRCFF*)RRCC -YHR200W

RCRCCFCCCRRCF(*FCFCRRR*)CFCFC(*CFCRCCFF*)C -YJL115W

(*FRFRFRR*)CFRFCCRCFCFCCFRFRCFRCCFRFRF -YGR282C

CC(*FRFRFRR*)FCFCCRRFCCCCRCFCRRCFFRRCC -YHR200W format. When the number of genes increases, the number of gene pairs will increase tremendously, which greatly enlarges the complete result of *Event Method.* As a result, the *Event Method* has to ignore a large number of lowly ranked gene pairs. This inevitably lose some interesting pairs for the *Event Method* cannot always rank them high as stated above. Our results can give complete information of the whole dataset in a relatively concise format in 3^{q-1} *q-clusters.* Moreover, the users can also decrease the number of *q-clusters* by ignoring patterns with relatively more "0". Moreover, our results are ready for deep exploration of co-regulation relationships between genes according to users' special needs.

6.3.3 Time-Lagged Co-regulated Genes/Gene Clusters

We shall examine the results of the time-lag co-regulated genes and gene clusters produced by our algorithm. In total, there are 640 non-empty *q-clusters* (patterns). Table 6.11 shows one representative *q-cluster* with pattern "0(−1)0(−1)01" and *q-clustersID* 181. The first number of each line indicates the starting position of the pattern in the genes, while the following numbers are the genes' identifier. For example, the last second line means that Gene 951, Gene 2524, Gene 6059 and Gene 6086 have the changing pattern "0(−1)0(−1)01" starting from the 27th time point. Time-lagged relationships between not only genes but also gene clusters are shown clearly in our results. Although those relationships may not be all true existing time-lagged co-regulations, they help researchers to reduce the search space and focus their efforts on the promising relationships. Our results do deliver known co-regulated genes already established by biologists. For example, YGL207W and YDR224C are genes with activation co-regulation, and YHR200W and YGR282C are also such gene pairs in [17]. Moreover, our method is not limited to the "$A \rightarrow B$" relationship. It can

Table 6.11: Sample Result - *q-cluster* 181.

```
0(-1)0(-1)01
1 183 2247 3874
2 1049 2725
3 459 512 970 992 1048 1072 1120 1167 1189 1530 1555 1603 1700
   2012 3832 5995
4 233 555 557 1053 1341 1709 2973 3240 4270 4271 5023 5147 5974
5 947 1626 2735
6 466 844 2442 2576 3107 3412 4206 4982 5278 5670 5691
7 114 236 1837 2226 2534 3074 3260 3480 3572 3941 3961 4211
   4249 4531 4544 4661 5292 5622 5725 5807 5850 6099
8 548 715 1061 1087 1576 5375
9 216 316 2748
10 384 567 928 1213 1329 2541 4157 4386 4442
11 1664
12 466 877 4978 5006 5019 5141 5211 5426 5498 5821 5859 5980
13 1776 2824 4848
14 766 885 1538 1592 2372 2562 3449 3643 4407 4695 4708
15 23 234 629 1010 1035 1751 1874 1906 2163 2234 2235 2565 2747
   2782 2814 3146 3346 3448 3640 4321 4393 5539
16 2658 3452 3470 3489 3809 5944
17 310 398 546 547 1148 1481 1543 1557 1694 2462 2934 2945 2957
   3377 3693 3712 4288 4302 4303 4630 4768 4782 5317 5461 6163
18 2757 2786 3063 3420 3651
19 2120 2215 3599 5123
20 1665 1709 2534 3204 3927
21 664 1117 1512 1520 2613 2873 2962 3049 5097 5567 5655 5863 6024
22 171 291 757 907 942 1075 1223 1224 1326 1344 1398 1416 1578
   1704 2003 2218 2280 2377 2409 2412 2424 3470 3478 3704 3710
   3786 3820 3954 3986 4058 4104 4187 4392 4667 4746 4786 4826
   5069 5327 5861 5925 5936
23 1751 3134 4329
24 176 2718 3015 3452 3706 4515 4721 5498
25 183 1042 1581 1675 1760 1810 1926 1933 2172 2274 2298 2780 6008
26 287 587 927 967 1079 1093 1140 1207 2378 2662 3141 3242 3867
   4305 4366 4520 4739 5401 5615 5619 5884
27 951 2524 6059 6086
28 3789
```

also infer the "$A \rightarrow B \rightarrow C \rightarrow D$" regulation pattern. As shown in Table 6.11, the former Cluster8 (548 715 1061 1087 1576 5375) may activate the latter Cluster10 (384 567 928 1213 1329 2541 4157 4386 4442) after 2 time lags, and the Cluster10 may go on activating an even later Cluster16 (2658 3452 3470 3489 3809 5944) after 6 time lags. We do not find such already known gene regulations in existing biological works. However, these co-regulated patterns may help future discovery of such regulatory pathways.

6.4 Summary

In this chapter, we revisited the problem of analyzing gene expression data for time-lag gene co-regulation relationships. We have presented a localized algorithm to identify the time-lagged gene patterns based on the concept of *q-clusters*. Genes with similar pattern over a subset of q consecutive time points (conditions) are grouped into the same *q-cluster*. As such, we can easily determine the co-regulations of genes within each *q-cluster* and between *q-clusters*. We have experimented on the real time series gene expression dataset and compared our method and results with the *Event Method*. Our study shows that our approach is efficient at detecting both the activation and inhibition time-lagged co-regulations, and our results can draw relationships between both genes and gene clusters with more detailed information. We believe our approach delivers valuable information and provides an excellent tool that facilitates deeper exploration for gene network research.

Chapter 7

Conclusion and Future Work

With the advances in DNA microarray technology, expression levels of thousands of genes can be simultaneously measured efficiently during important biological process and across collections of related samples. Analyzing the microarray data to identify localized co-expressed gene patterns has become the new focuses of researchers as such gene patterns are essential in revealing the gene functions, gene regulations, subtypes of cells, and cellular processes of gene regulation networks. This thesis has categorized the co-expressed patterns into three types (co-attribute patterns, co-tendency patterns, and time-lagged patterns), and proposed several new frameworks and algorithms to effectively and efficiently mine the three types of co-expressed patterns. The application of our research work will give new insights for biological researchers. In the following sections, we will summarize our contributions and give directions for future research.

7.1 Thesis Contributions

The main contributions of the thesis can be summarized as follows.

1. First, we have proposed to mine localized co-expressed gene patterns and categorized the patterns into three types: co-attribute patterns, co-tendency patterns and time-lagged patterns. We consider both the static and the dynamic aspects of gene co-regulations.

2. Second, to identify the co-attribute patterns from 2D dense microarray datasets, we have overcome the limitations of traditional 2D frequent closed pattern mining algorithms, and introduced a framework that progressively returns FCPs to users. We have proposed two schemes, *C-Miner* and *B-Miner*, that are based on this framework. The two schemes adopt different partitioning strategies - *C-Miner* partitions the mining space based on Compact Rows Enumeration whereas *B-Miner* partitions the space based on Base Rows Projection - and hence different pruning strategies. We have implemented *C-Miner* and *B-Miner*, and our performance study on synthetic datasets and real dense datasets shows their effectiveness over existing schemes. We have also implemented the parallel schemes of *C-Miner* and *B-Miner* that further enhance the mining efficiency. This is critical as, to our knowledge, there is no reported work in the literature on parallel frequent closed pattern mining.

3. Third, we have introduced the notion of frequent closed cube (FCC) and formally defined it, which generalizes the notion of 2D frequent closed pattern to 3D context. Based on this notion, we could mine 3D co-attribute patterns, which settles the new challenges coming up with the spurning of 3D microarray data. We have proposed two novel algorithms to mine FCCs from 3D datasets. The first scheme is a *Representative Slice Mining (RSM)* framework that can be used to extend existing 2D frequent closed pattern mining algorithms for FCC

mining. The second technique, called *CubeMiner*, is a novel algorithm that operates on the 3D space directly. We have also shown how *RSM* and *CubeMiner* can be easily extended to exploit parallelism. We have implemented *RSM* and *CubeMiner* and their parallel schemes, and conducted experiments on both real and synthetic datasets. The experimental results showed that the *RSM*-based scheme is efficient when one of the dimensions is small, while *CubeMiner* is superior otherwise. To our knowledge, there has been no prior work that mine FCCs.

4. Forth, to mine co-tendency patterns (biclusters) from 2D microarray data, we have re-examined how biclusters are extracted from the gene expression data and introduced a *quick hierarchical biclustering* algorithm *(QHB)* to ensure that the final bicluster trends are not only consistent but exhibit similar degrees of fluctuation between consecutive conditions. We have also provided a new merit function that gauges the degree of similarity in the fluctuations of the bicluster, enabling us to extract biclusters that fulfill this condition and filter off those that have a wide range of degree fluctuations. As shown in our experiments, our framework is able to efficiently mine biclusters of a better quality, compared with the more recent DBF framework [63]. Furthermore, *QHB* provides a hierarchical picture of inter-bicluster relationships, maintains information integrity and offers users a progressive way of knowledge exploration. This is very helpful in biological application. Instead of waiting long hours for all detailed results, biologists now would be provided with a general picture of the whole results from the upper levels of the hierarchical tree in a very short response time. Then biologists could freely choose their focus, rolling up to generalize

it or rolling down to detail it, progressively. This would help biologists quickly focus on their most interested patterns for further exploration. All the above features of *QHB* make it an attractive tool for microarray data analysis.

5. Finally, we have proposed an efficient algorithm *q-cluster* to identify time-lagged patterns. The algorithm facilitates localized comparison and processes several genes simultaneously to generate detailed and complete time-lagged information between genes/gene clusters. *q-cluster* can deliver time-lagged patterns with both similar and opposite changing tendency, which draw a clear picture of time based co-regulation (activation/inhibition) among genes and gene biclusters. We experimented with the time series Yeast gene dataset and compared our scheme with the *Event Method* [34]. Our results show that our scheme is not only efficient, but delivers more reliable and detailed information of time-lagged co-regulation between genes/gene clusters. We believe our approach delivers valuable information and provides an excellent tool that facilitates deeper exploration for gene network research.

7.2 Future Research Directions

While this thesis has presented efficient algorithms to localized co-expressed gene patterns mining, a number of issues could be further investigated.

- First, although there have been some encouraging results on co-attribute pattern mining from both 2D (FCP) and 3D (FCC) microarray datasets, the number of resulting patterns is still not small. This will bring some difficulty for biologists to analyze them. New approaches may consider how to make use of some biological discoveries on gene networks as prior-knowledge of "interesting" frequent closed pattern mining. The prior-knowledge of gene relationships could not only act as a post-filter to figure out more interesting patterns, but also could be put into the early pruning process to enhance mining efficiency.

- Second, further exploration on the resulting co-attribute patterns (FCPs and FCCs) will be another interesting research approach. Gene association rule mining from 2D FCPs has been well studied in the literature. And cancer classifier built on 2D FCPs has also proven its effectiveness in application [13]. Hence, association rule mining and classifier building on 3D FCCs could be further explored.

- Third, based on the partitioning scheme of the FCC mining algorithm *CubeMiner* and the principle of biclustering algorithm *QHB*, we could further extend the co-tendency patterns from 2D to 3D microarray datasets. That is, new efficient algorithms for hierarchical tri-clusters mining could be designed.

- Finally, although the time-lagged pattern mining algorithm *q-cluster* can deliver the detailed and complete time-lagged information between genes/gene clusters, the genes/gene clusters that act as the activator/inhibitor have the same affecting time periods as the genes/gene clusters that are activated/ inhibited. In genetic regulatory networks, there also exist genes/gene clusters that regulate each other but have different affecting time periods. For example, some genes/gene clusters may have a similar/opposite but "enlarged/shortened" fluctuating shape with their activators/inhibitors. Future work can be done to mine such "enlarged/shortened" time lagged patterns.

Bibliography

[1] K. Aas and M. Langaas, *Microarray data mining: A survey*, Tech. report, NR Note SAMBA/02/01, 2001.

[2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, *Automatic subspace clustering of high dimensional data for data mining applications*, SIGMOD'98, 1998, pp. 94–105.

[3] R. Agrawal and R. Srikant, *Fast algorithms for mining association rules*, VLDB'94, 1994, pp. 487–499.

[4] R. Agrawal and R. Srikant, *Mining sequential patterns*, ICDE'95, 1995, pp. 3–14.

[5] Y. Barash and N. Friedman, *Context-specific bayesian clustering for gene expression data*, Proceedings of the Third Annual International Conference on Research in Computational Molecular Biology (RECOMB'01), 2001, pp. 12–21.

[6] A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini, *Discovering local structure in gene expression data: the order-preserving submatrix problem*, Proceedings of the Sixth Annual International Conference on Computational Biology, 2002, pp. 49–57.

[7] J. Besson, C. Robardet, and J.F. Boulicaut, *Constraint-based mining of formal concepts in transactional data*, PaKDD'04, 2004, pp. 615–624.

[8] N. Bobola, P.R. Jansen, T.H. Shin, and K. Nasmyth, *Asymmetric accumulation of ashlp in postanaphase nuclei depends on a myoscin and restricts yeast mating-type switching to mother cells*, Cell, 1996, pp. 699–709.

[9] D. Burdick, M. Calimlim, and J. Gehrke, *Mafia: A maximal frequent itemset algorithm for transactional databases*, Proceedings of the 17th International Conference on Data Engineering(ICDE'01), 2001.

[10] T. Chen, V. Filkov, and S. Skiena, *Identifying gene regulatory networks from experimental data*, Proceedings of the Third Annual International Conference on Research in Computational Molecular Biology (RECOMB'99), 1999, pp. 94–103.

[11] Y. Cheng and G. Church, *Biclustering of expression data*, Proc. 8th ISMB, 2000, pp. 93–103.

[12] G. Cong, K.L. Tan, A.K.H. Tung, and F. Pan, *Mining frequent closed patterns in microarray data*, ICDM'04, 2004, pp. 363–366.

[13] G. Cong, A.K.H. Tung, X. Xu, F. Pan, and J. Yang, *Farmer: Finding interesting rule groups in microarray datasets*, SIGMOD'04, 2004.

[14] M.B. Eisen, P.T. Spellman, P.O. Brown, and D. Botstein, *Cluster analysis and display of genome-wide expression patterns*, Proc. Natl. Acad. Sci. USA, 1998, pp. 14863–14868.

[15] M. El-Hajj and O.R. Zaiane, *Parallel association rule mining with minimum inter-processor communication*, Proceedings of DEXA'03, 2003, pp. 519–523.

[16] D. Lockhart et. al., *Expression monitoring by hybridization to high-density oligonucleotide arrays*, Nat. Biotechnol, 1996, pp. 1675–1680.

[17] R.J. Cho et. al., *A genome-wide transcriptional analysis of the mitotic cell cycle*, Mol. Cell, 1998, pp. 65–73.

[18] C. Fraley and A.E. Raftery, *How many clusters? which clustering method? answers via model-based cluster analysis*, The Computer Journal, 1998, pp. 578–588.

[19] G. Getz, E. Levine, and E. Domany, *Coupled two-way clustering analysis of gene microarray data*, Proc Natl Acad Sci U S A., 2000, pp. 79–84.

[20] J. Han, G. Dong, and Y. Yin, *Efficient mining of partial periodic patterns in time series database*, ICDE'99, 1999, pp. 106–115.

[21] J. Han and M. Kamber, *Chapter 3: Data preprocessing*, Data Mining: Concepts and Techniques, 2000, pp. 105–130.

[22] J. Han J. Wang and J. Pei, *Closet+: Searching for the best strategies for mining frequent closed itemsets*, KDD'03, 2003, pp. 236–245.

[23] L. Ji, K.W.L. Mock, and K.L. Tan, *Quick hierarchical biclustering on microarray gene expression data*, Proceedings of the 6th IEEE Symposium on Bioinformatics and Bioengineering (BIBE'06), 2006.

[24] L. Ji and K.L. Tan, *Mining gene expression data for positive and negative co-regulated gene clusters*, Bioinformatics, 2004, pp. 2711–2718.

[25] L. Ji and K.L. Tan, *Identifying time-lagged gene clusters on gene expression data*, Bioinformatics, 2005, pp. 509–516.

[26] L. Ji, K.L. Tan, and A.K.H. Tung, *Progressive mining of frequent closed patterns from dense datasets*, Submitted for Publication.

[27] L. Ji, K.L. Tan, and A.K.H. Tung, *Mining frequent closed cubes in 3d datasets*, Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB'06), 2006.

[28] D. Jiang, J. Pei, and A. Zhang, *Dhc: A density-based hierarchical clustering method for time series gene expression data*, Proceeding of BIBE2003: 3rd IEEE International Symposium on Bioinformatics and Bioengineering, 2003.

[29] D. Jiang, J. Pei, and A. Zhang, *Gpx: Interactive mining of gene expression data*, VLDB 2004, 2004, pp. 1249–1252.

[30] D. Jiang, J. Pei, and A. Zhang, *A general approach to mining quality pattern-based clusters from microarray data*, DASFAA 2005, 2005, pp. 188–200.

[31] D. Jiang, J. Pei, and A. Zhang, *An interactive approach to mining gene expression data*, IEEE Trans. Knowl. Data Eng., 2005, pp. 1363–1378.

[32] D. Jiang, C. Tang, and A. Zhang, *Cluster analysis for gene expression data: A survey*, In Proceedings of IEEE Transactions on Knowledge and Data Engineering (TKDE), 2004, pp. 1370–1386.

[33] M. Kato, T. Tsunoda, and T. Takagi, *Lag analysis of genetic networks in the cell cycle of budding yeast*, Genome Informatics, 2001, pp. 266–267.

[34] A.T. Kwon, H.H. Hoos, and R. Ng, *Inference of transcriptional regulation relationships from gene expression data*, Bioinformatics, 2003, pp. 905–912.

[35] L. Lazzeroni and A. Owen, *Plaid models for gene expression data*, Laura Lazzeroni and Art Owen Statistica Sinica, 2002, pp. 61–86.

[36] H. Mannila, H. Toivonen, and A.I. Verkamo, *Efficient algorithms for discovering association rules*, KDD'94, 1994, pp. 181–192.

[37] H. Mannila, H. Toivonen, and A.I. Verkamo, *Discovery of frequent episodes in event sequences*, Data Mining and Knowledge Discovery, 1997, pp. 259–289.

[38] L.J. Oehlen, J.D. McKinney, and F.R. Cross, *Stel2 and mcm1 regulate cell cycle-dependent transcription of far1*, Mol. Cell. Biol., 1996, pp. 2830–2837.

[39] F. Pan, G. Cong, and A.K.H. Tung, *Carpenter: Finding closed patterns in long biological datasets*, KDD'03, 2003, pp. 673–642.

[40] F. Pang, A.K.H. Tung, G. Cong, and X. Xu, *Cobbler: Combining column and row enumeration for closed pattern discovery*, Proceedings of SSDBMS'04, 2004, pp. 21–30.

[41] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, *Discovering frequent closed itemsets for association rules*, ICDT'99, 1999, pp. 398–416.

[42] J. Pei, J. Han, and R. Mao, *Closet: An efficient algorithm for mining frequent closed itemsets*, Proceedings of 2000 ACMSIGMOD Int. Workshop Data Mining and Knowledge Discovery, 2000.

[43] J. Pei, D. Jiang, and A. Zhang, *Mining cross-graph quasi-cliques in gene expression and protein interaction data*, ICDE 2005, 2005, pp. 353–354.

[44] J. Pei, D. Jiang, and A. Zhang, *On mining cross-graph quasi-cliques*, KDD 2005, 2005, pp. 228–238.

[45] H. Pinto, J. Han, J. Pei, K. Wang, Q. Chen, and U. Dayal, *Multi-dimensional sequential pattern mining*, Proceedings of the Tenth International Conference on Information and knowledge Management, 2001, pp. 81–88.

[46] S. Ramaswamy, S. Mahajan, and A. Silberschatz, *On the discovery of interesting patterns in association rules*, Proceedings of the International Conference on Very Large Databases(1998), 1998, pp. 368–379.

[47] M. Schena, D. Shalon, R. Davis, and P.Brown, *Quantitative monitoring of gene expression patterns with a compolementatry dna microarray*, Science, 1995, pp. 467–470.

[48] E. Segal, B. Taskar, A. Gasch, N. Friedman, and D. Koller, *Rich probabilistic models for gene expression*, Bioinformatics, 2001, pp. 243–252.

[49] R. Shamir and R. Sharan, *Click: A clustering algorithm for gene expression analysis*, Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology (ISMB'00), 2000.

[50] P. Shenoy, J. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah, *Turbo-charging vertical mining of large databases*, SIGMOD'00, 2000, pp. 22–23.

[51] P.T. Spellman and G. Sherlock et. al., *Comprehensive identication of cell cycle-regulated genes of the yeast saccharomyces cerevisiae by microarray hybridization*, Molecular Biology of the Cell, 1998, pp. 3273–3297.

[52] R. Sucahyo and A. Rudra, *Efficiently mining frequent patterns from dense datasets using a cluster of computers*, Proceedings of Aust. Conf. on AI, 2003, pp. 233–244.

[53] A. Tanay, R. Sharan, and R. Shamir, *Discovering statistically significant biclusters in gene expression data*, Bioinformatics, 2002, pp. 136–144.

[54] C. Tang and A. Zhang, *Interrelated two-way clustering and its application on gene expression data*, International Journal on Artificial Intelligence Tools, 2005, pp. 577–598.

[55] A. Tefferi, E. Bolander, M. Ansell, D. Wieben, and C. Spelsberg, *Primer on medical genomics part iii: Microarray experiments and data analysis*, Mayo Clin Proc., 2002, pp. 927–940.

[56] H. Wang, W. Wang, J. Yang, and P.S. Yu, *Clustering by pattern similarity in large data sets*, Proceedings of the ACM 2002 SIGMOD International Conference on Management of Data, 2002, pp. 394–405.

[57] G. Yang, *The complexity of mining maximal frequent itemsets and maximal frequent patterns*, In Proceedings of KDD'04, 2004, pp. 344–353.

[58] J. Yang, W. Wang, H. Wang, and P.S. Yu, *δ-cluster: Capturing subspace correlation in a large data set*, Proceedings of the 18th IEEE International Conference on Data Engineering, 2002, pp. 517–528.

[59] L.K. Yeung, L.K. Szeto, A.W. Liew, and H. Yan, *Dominant spectral component analysis for transcriptional regulations using microarray time-series data*, Bioinformatics, 2004, pp. 742–749.

[60] M. Zaki and C. Hsiao, *Charm: An efficient algorithm for closed association rule mining*, SDM'02, 2002.

[61] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, *New algorithms for fast discovery of association rules*, KDD'97, 1997, pp. 283–286.

[62] M.J. Zaki, *Parallel and distributed association mining: A survey*, Proceedings of IEEE Concurrency (Special Issue on Data Mining), 1999, pp. 14–25.

[63] Z. Zhang, M.W. Teo, B.C. Ooi, and K.L. Tan, *Mining deterministic biclusters in gene expresssion data*, Proceedings of the 4rd IEEE Symposium on Bioinformatics and Bioengineering(BIBE'04), 2004, pp. 283–290.

[64] L. Zhao and M.J. Zaki, *Tricluster: An effective algorithm for mining coherent clusters in 3d microarray data*, Proceedings of the 2005 ACM-SIGMOD Conference, 2005, pp. 694–705.