MINING SUBSPACE AND BOOLEAN PATTERNS FROM DATA

By

Lizhuang Zhao

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

Approved by the
Examining Committee:

_____
Mohammed Javeed Zaki, Thesis Adviser

_____
Chris Bystroff, Member

_____
David Musser, Member

_____
Mukkai Krishnamoorthy, Member

_____
Malik Magdon-Ismail, Member

Rensselaer Polytechnic Institute
Troy, New York

September 2006
(For Graduation December 2006)

MINING SUBSPACE AND BOOLEAN PATTERNS FROM DATA

By

Lizhuang Zhao

An Abstract of a Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

The original of the complete thesis is on file
in the Rensselaer Polytechnic Institute Library

Examining Committee:

Mohammed Javeed Zaki, Thesis Adviser

Chris Bystroff, Member

David Musser, Member

Mukkai Krishnamoorthy, Member

Malik Magdon-Ismail, Member

Rensselaer Polytechnic Institute
Troy, New York

September 2006
(For Graduation December 2006)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

First and foremost I would like to express my sincere gratitude and appreciation to my advisor, Professor Mohammed J. Zaki, for all his contributions of insightful ideas, patience, time and funding to make my Ph.D. experience productive and stimulating. The joy and enthusiasm he has for data mining research was contagious and motivational for me.

Thanks to the members of my examining committee, for their comments, for their insightful questions and for their interests in my studies.

To the members of our research group, to my friends and fellow graduate students, I thank you all for your help and friendship which gave me an unforgettable period of enjoyable time.

To the staff in the department, Terry Hayden, Chris Coonrad and many others, I also extend a most grateful thank you for helping me wade through plenty of administrative procedures.

Finally, my gratitude must go to my family for all their love, support and encouragement. To my parents who raised and supported me with deep love, to my parents-in-law for their continuing help, and to my loving wife Jingjing for her unwavering love and faithful support throughout our years at RPI.

This research was supported in part by NSF Career Award IIS-0092978, DOE Career Award DE-FG02-02ER25538, NSF Grants EIA-0103708 and EMT-0432098.

Troy, September, 2006

ABSTRACT

In this thesis, we set up a systematic framework for mining subspace and boolean patterns from data. Subspace patterns are extracted from real-valued datasets, whereas boolean patterns are mined from binary-valued datasets. For real-valued datasets, we mainly consider scaling and shifting relationships, whereas for binary-valued datasets, we mine arbitrary boolean expressions (OR, AND, CNF, DNF). Boolean patterns are also used to mine redescriptions, i.e., to describe the same group of objects in multiple "orthogonal ways".

The subspace pattern mining has been tailored to gene microarray data clustering to find biclusters and triclusters. We present two novel deterministic clustering algorithms: MICROCLUSTER and TRICLUSTER (the first 3D microarray subspace clustering), which can mine arbitrarily positioned and overlapping biclusters/triclusters. Depending on different parameter values, they can mine different types of clusters, including those with constant or similar row/column values, as well as scaling and shifting expression patterns. Optionally, the two algorithms merge/prune some clusters having large overlaps. We also give a useful set of metrics to evaluate the clustering quality, and show their effectiveness on real microarray expression data.

We also present a comprehensive framework, BLOSOM, for mining boolean patterns from binary-valued datasets. This framework forms the algorithmic core of a redescription mining solution. We organize the space of boolean expressions into four categories; pure conjunctions, pure disjunctions, conjunction of disjunctions, and disjunction of conjunctions. For each category, we propose a *closure* operator that naturally leads to the concept of a *closed* boolean expression. Further, the closed generators form a lossless representation of all possible boolean expressions and, hence, all possible redescriptions. BLOSOM efficiently mines several forms of frequent boolean expressions by utilizing a number of methodical pruning techniques.

The future work is to extend subspace pattern mining to more general notions of linear relationships, combining scaling and shifting patterns. For boolean pattern mining, we plan to apply them to the gene ontology (GO) to discover cross relationships between different GO hierarchies.

CHAPTER 1
Introduction

We introduce the pattern mining problem in four sections: motivations and problem statements, background, challenges and our contributions. In each section, there are two parts related to subspace pattern mining and boolean pattern mining, respectively.

1.1 Motivations and Problem Statements

In this thesis, we are trying to find useful information from datasets. A typical dataset is a two-dimensional (2D) matrix, where each row represents a transaction, and each column represents an item. A dataset could be a real-valued one as shown in Figure 1.1(a), or a binary-valued one as shown in Figure 1.1(b), where 'T' means true and 'F' means false.

items

transactions

	A	B	C	D	E
i	5	3	8	4	2
ii	3	8	3	7	8
iii	9	1	1	2	3
iv	2	2	6	4	6
v	6	6	9	2	5
vi	8	3	2	6	9
vii	1	4	7	1	4

(a)

items

transactions

	A	B	C	D	E
i	T	F	F	T	F
ii	T	T	T	T	F
iii	F	T	F	T	F
iv	T	F	T	T	T
v	F	T	F	F	T
vi	F	T	F	F	T
vii	F	F	F	F	F

(b)

Figure 1.1: Example Datasets: (a) Real-valued, (b) Binary-valued

For real-valued datasets, we are interested in the subspace pattern, i.e, a considerable (both in rows and columns) group of data following some homogeneity pattern. For

example, in Figure 1.1(a), submatrix $\{iii, iv, vi\} \times \{B, D, E\}$ is such a data group with scaling pattern, i.e., the ratios between any two columns/rows are close. Other general patterns are shown in Figure 1.2 as follows: (a) constant column pattern whose values in the same column are close, (b) constant row pattern whose values in the same row are close, (c) constant pattern whose values in the submatrix are close, (d) scaling pattern whose values ratios between any two columns/rows are close, (e) shifting pattern whose value differences between any two columns/rows are close, (f) order preserving pattern with the values in each row following the column ranks given at the top.

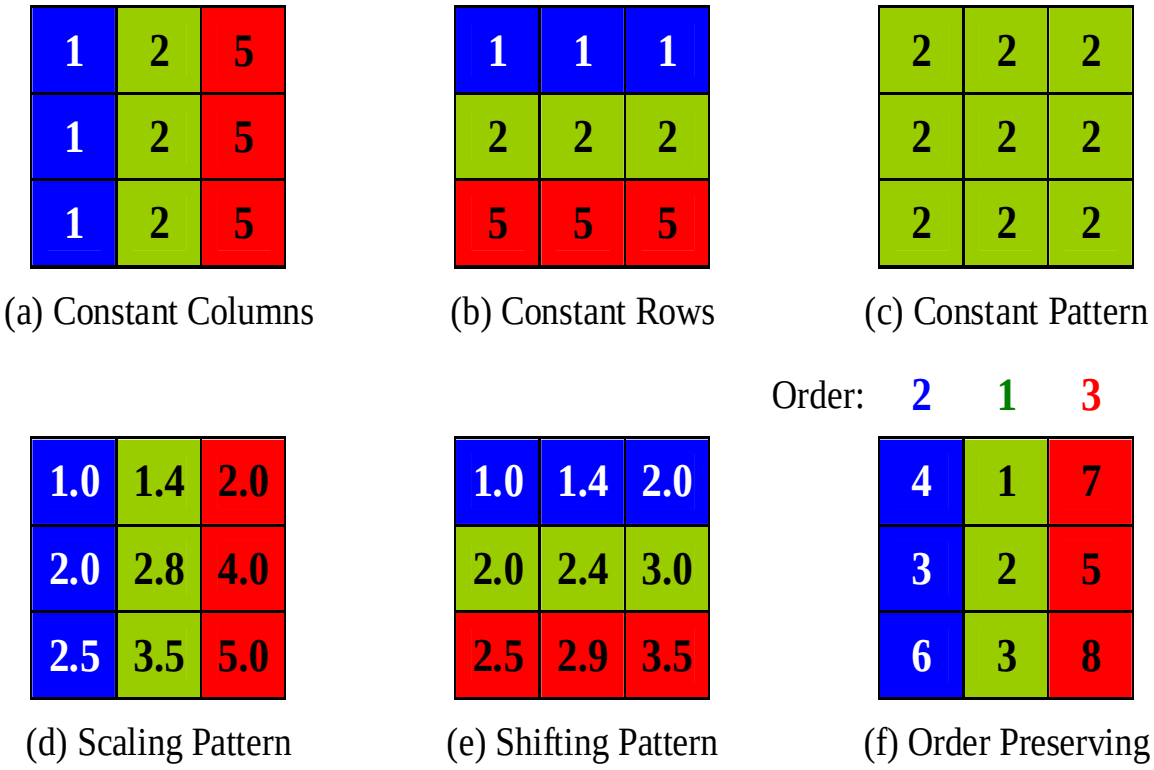


Figure 1.2: Example Datasets with Different Patterns

We use subspace clustering — biclustering (a clustering process of simultaneously mining column and row for a 2D real-valued data matrix) [13] to mine the subspace patterns (biclusters) from real-valued datasets. Traditional clustering algorithms work in the full dimensional space, which consider the value of each point in all the dimensions and try to group the similar points together. For example, in Figure 1.3 (a), if points p_2, p_4 and p_5 are similar in the whole space, they can be clustered together. Biclustering, however, does not have such a strict requirement. If some points are similar in several

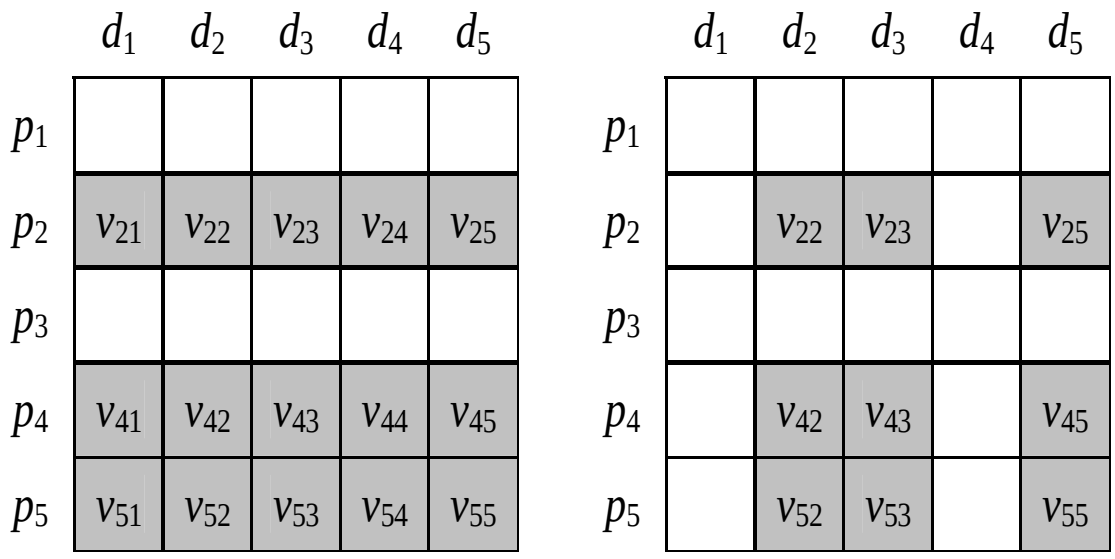


Figure 1.3: Difference between (a) Fullspace Clustering and (b) Biclustering

dimensions (a subspace), they will be clustered together in that subspace. For example, in Figure 1.3 (b), if points p_2, p_4 and p_5 are similar in the subspace composed of dimensions d_2, d_3 and d_5, they form a bicluster. Biclustering is very useful, especially for clustering in a high dimensional space where often only some dimensions are meaningful for some subset of points.

Furthermore we extended the bicluster concept for 2D datasets to handle three-dimensional (3D) datasets, which we call a *tricluster*. For example in a $4 \times 4 \times 3$ dataset, we show three biclusters in each of the three 4×4 slices, in Figure 1.4 (a). Their overlapping cells are shown as shaded. If we combine these biclusters together, we can get the final triclusters ($2 \times 2 \times 3$) as shown in Figure 1.4 (b). The detailed tricluster concept will be given later in Chapter 4.

For binary-valued datasets, we are interested in boolean patterns among columns/rows. We use the binary-valued dataset of Figure 1.5 to show some boolean pattern examples, where '**x**' means true and blank means false. From this dataset, we can see that $A \cup B = C \cup D$, which means both of the item expressions result in the same transaction set $\{1, 2, 3, 4\}$. In addition, the boolean expressions can be pure conjunction, pure disjunction, conjunctive normal form (CNF), disjunctive normal form (DNF), etc., and we are interested in those compact boolean expressions of the binary-valued data, which

have no information redundancy.

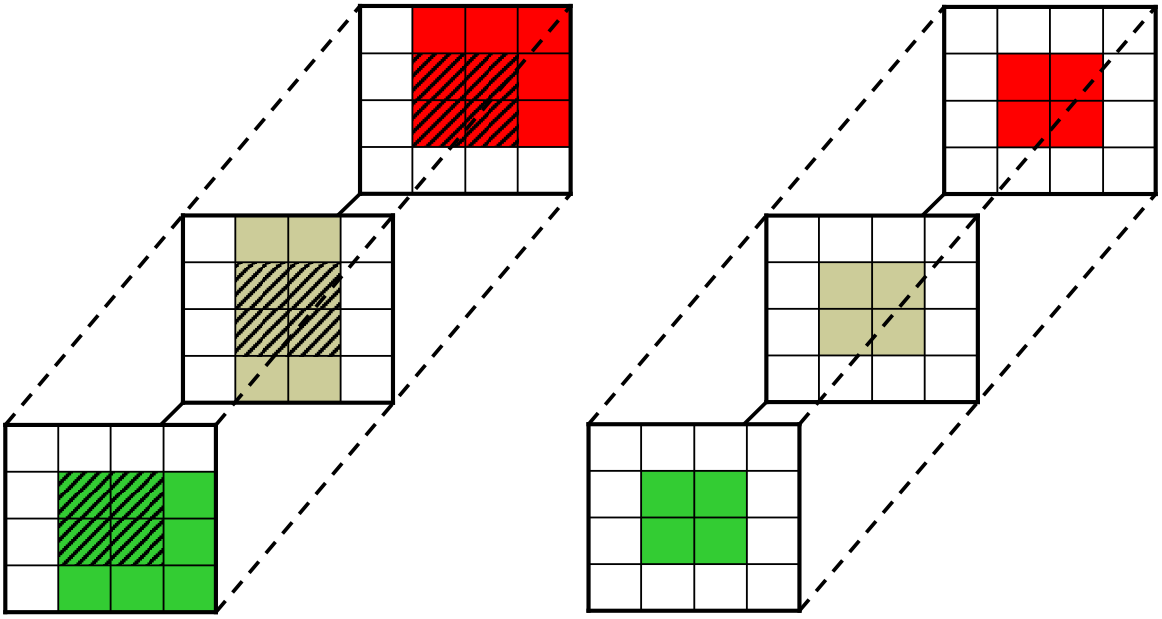


Figure 1.4: Tricluster Assembling: (a) Biclusters (b) Final Tricluster

items

transactions

	A	B	C	D	E
1	x		x	x	
2		x	x		
3	x	x	x	x	
4	x			x	x
5					x

Figure 1.5: A Binary-valued Dataset

1.2 Biological Background

1.2.1 Microarrays

A microarray is a two dimensional array, typically on a glass, filter, or silicon wafer, upon which genes or gene fragments are deposited or synthesized in a predetermined spatial order allowing them to be made available as probes in a high-throughput, parallel manner.

DNA microarrays (or chips) offer the latest technological advancement for multi-gene detection and diagnostics. They were conceived originally to examine gene expression for large numbers of genes. Every gene will be measured simultaneously. For example, one common use of DNA chips is to determine which genes are activated and which genes are repressed for some populations of cells. Furthermore, microarrays have also been applied to DNA sequence analysis, immunology, genotyping and diagnostics.

Given cells from different samples/tissues, we can measure their expression levels as follows [56]. First we extract the messenger RNA (mRNA) from cells. Genes which code for proteins are transcribed into mRNA in the cell nucleus. The mRNAs in turn are translated into proteins by ribosomes in the cytoplasm. The transcription level of a gene is taken to be the amount of its corresponding mRNA present in the cell. Comparative hybridization experiments compare the amounts of many different mRNAs in cell populations.

However, captured mRNAs are still difficult to work with because they are prone to being destroyed. The environment is full of RNA-digesting enzymes, so free RNA is quickly degraded. To prevent the experimental samples from being lost, each mRNA is converted (reverse-transcribed back) to a more stable labeled/colored (tagged with fluorescent molecules which emit detectable light when stimulated by a laser) DNA form which is called complementary DNA (cDNA) because its sequence is the complement of the original mRNA sequence.

Next, cDNAs are hybridized with the microarray/chip. The array holds hundreds or thousands of spots, each of which contains a different DNA sequence that can base pair with some kind of cDNA. So some labeled/colored cDNA binds to the spots, and the unbound cDNA is washed off.

Now it is time to detect the bound cDNA so it can be visualized. The microscope

slide containing the microarray is placed inside a darkened box, where it is scanned with a laser (with the same color as that of labeled cDNA) to detect the bound cDNA. The output image is stored on a computer for later analysis. The intensities provided by the array image can be quantified by measuring the average or integrated intensities of the spots.

So a typical microarray dataset is a 2D real-valued matrix, where rows represent genes, and columns represent samples got from different locations, under different conditions or at different times. In addition, each cell value in the microarray represents a gene expression level (or its logarithm).

1.2.2 Gene Ontology

The Gene Ontology (GO) [57] project was developed by the Gene Ontology Consortium. The ontologies (or controlled vocabularies) describe three aspects of gene products: molecular function, cellular component, and biological process. Molecular function describes activities, such as catalytic or binding activities, at the molecular level. GO molecular function terms represent activities rather than the entities (molecules or complexes) that perform the actions, and do not specify where or when, or in what context, the action takes place. Molecular functions generally correspond to activities that can be performed by individual gene products, but some activities are performed by assembled complexes of gene products. A biological process is a series of events accomplished by one or more ordered assemblies of molecular functions. A cellular component is just that, a component of a cell, but it could be part of some larger object, which may be an anatomical structure or a gene product group.

A gene product has one or more molecular functions and is used in one or more biological processes; it might be associated with one or more cellular components. Collaborating databases annotate their gene products (or genes) with GO terms, providing references and indicating what kind of evidence is available to support the annotations.

These controlled vocabularies are structured into directed acyclic graphs (DAGs) where each term represents a ‘node’ and the edges of each graph represent the relationship between the nodes.

1.3 Algorithm Background

In this thesis we consider the subspace pattern mining problem within the context of gene microarray analysis to mine subspace clusters. Biclustering has proved of great value for finding the interesting patterns in the microarray expression data [14], which records the expression levels of many genes (the rows/points), for different biological samples (the columns/dimensions). Biclustering is able to identify the co-expression patterns of a subset of genes that might be relevant to a subset of the samples of interest. Besides biclustering along the gene-sample dimensions, there has been a lot of interest in mining gene expression patterns across time [6]. The proposed approaches are also mainly two-dimensional, i.e., finding patterns along the gene-time dimensions.

Subspace patterns are extracted from real-valued datasets, whereas for binary-valued datasets, we are interested in their boolean patterns, which are also used to mine redescriptions, i.e., to describe the same group of objects in multiple "orthogonal ways".

In the past year, a new data mining problem called *redescription mining* has become popular [34, 52], both as a novel generalization of association rule mining and as a task of its own merit in bioinformatics and other applications. Redescription mining takes as input a binary-valued dataset just as in transaction-item modeling, but views it in a more general object-property modeling context, e.g., genes and the properties they exhibit, countries and their socio-economic indicators, political candidates and their positions on sensitive issues, and so on. The goal of redescription mining is to find object sets that can each be defined by at least two property sets (hence the name re-description). For instance, we might find that "*political candidates promoting gun control measures* are the same as *candidates with avowed support for the right to abortion*." More interestingly, since redescription encourages the view of properties as general boolean propositions, the induced property sets can be arbitrary boolean expressions. In particular, we are allowed to negate, intersect, and union the given properties, or create any arbitrary set theoretic expression for the purpose of obtaining a redescription. For instance, "*political candidates who support the death penalty* **but not** *campaign finance reform* are the same as *candidates who oppose amnesty for illegal immigrants* **and** *who oppose environment and labor standards in trade considerations*" is a more complicated redescription relating a difference of two properties to an intersection of two other properties.

Significant insight has already been obtained into the connections between association rule mining and redescription mining. Just as finding frequent sets is a precursor to mining association rules, finding frequent boolean expressions is a precursor to mining redescriptions. Just as we need only concentrate on closed itemsets for the purposes of rule mining, we need only focus on closed boolean expressions for the purposes of redescription mining. Just as a non-redundant association rule relates the minimum generators of two closed itemsets (one of which being a subset of the other), a non-redundant redescription relates two distinct minimum generators of the same closed boolean expression.

1.4 Challenges

Concerning the subspace/boolean pattern mining problems mentioned above, there are several challenges. For the challenges of mining microarray data, first, biclustering is an NP-hard problem [13], and thus many proposed algorithms of mining biclusters use heuristic methods or probabilistic approximations, which as a trade-off decrease the accuracy of the final clustering results. Second, microarray data is susceptible to noise, due to varying experimental conditions, thus methods should handle noise well. Third, given that we do not understand the complex gene regulation circuitry in the cell, it is important that clustering methods allow overlapping clusters that share subsets of genes or samples. Furthermore, the methods should be flexible enough to mine several (interesting) types of clusters. MICROCLUSTER and TRICLUSTER are two subspace clustering algorithms we present in this thesis, which meet the challenges above. In addtion, TRICLUSTER is an algorithm for 3D datasets; so it also needs to deal with these challenges for 3D data, which is even harder.

The above are the challenges to mine subspace patterns from real-valued datasets. For boolean pattern mining from binary-valued data, however, there are still major challenges to overcome. To date, for boolean pattern mining, none of the published algorithms can mine redescriptions between *arbitrary* boolean expressions. The CARTwheels algorithm [34] mines redescriptions only between length-limited boolean expressions in disjunctive normal form and CHARM-L [52] is restricted to redescriptions between conjunctions. Reference [31] presents the beginnings of a general framework but provides

only a conceptual description and does not involve any implementation or evaluation.

Aiming at these challenges, in this thesis we present three novel algorithms to address some hard problems faced by subspace/boolean pattern mining as follows.

1.5 Contributions

In this thesis, we present two efficient, deterministic, and complete subspace clustering methods called MICROCLUSTER [53] and TRICLUSTER [54] (which inherits all the features of MICROCLUSTER and expands them for 3D datasets), that address the above challenges of subspace pattern mining. The key features of our approach include:

1. We mine only the maximal biclusters/triclusters satisfying certain homogeneity criteria. The clusters can be arbitrarily positioned anywhere in the input data matrix and they can have arbitrary overlapping regions.

2. We use a flexible definition of a cluster which can mine several types of biclusters/triclusters, such as the clusters having identical or approximately identical values along each dimension, clusters with similar values along only one dimension or two dimensions, and clusters that exhibit scaling or shifting expression values (where one dimension is a approximately constant multiple of or is at an approximately constant offset from another dimension, respectively).

3. MICROCLUSTER and TRICLUSTER can optionally merge/prune biclusters/triclusters that have large overlaps, thus they can tolerate some noise in the dataset, and let the user focus on the most important clusters.

4. We also present a useful set of metrics to evaluate the clustering quality, and show the effectiveness of the two algorithms on several synthetic and real datasets.

5. In addition, specifically, TRICLUSTER is interested in mining triclusters, i.e., mining coherent clusters along the gene-sample-time (temporal) or gene-sample-region (spatial) dimensions. It utilizes the inherent unbalanced property of the 3D microarray dataset to achieve good performance.

6. To the best of our knowledge, TRICLUSTER is *the first 3D microarray subspace clustering method*.

Besides the contributions for subspace pattern mining, we also contribute considerably to the development of boolean pattern mining.

About the boolean pattern mining from binary-valued datasets, in this thesis we present the first algorithm [55], BLOSOM (an anagram of the bold letters in **BOOL**ean expression **M**ining over attribute **S**ets), to *simultaneously* mine closed boolean expressions over attribute sets *and* their minimal generators, yielding direct redescription mining solutions. Since every pair of minimal generators of a given closed attribute set yields a non-redundant redescription (see [52] for theoretical background) we do not explicitly mention redescriptions henceforth but instead focus on minimal generators. Furthermore, although redescription can be applied in a more general context, we utilize the nomenclature of item-transaction modeling, primarily for the ease with which the reader can map the presented concepts to prior literature. Our main contributions to boolean pattern mining are as follows:

1. We organize boolean expressions into four categories: (i) pure conjunctions (AND-clauses), (ii) pure disjunctions (OR-clauses), (iii) conjunctive normal form (conjunction of disjunctions), and (iv) disjunctive normal form (disjunction of conjunctions). The theory and algorithms developed for (i) and (ii) are then used as building blocks to develop solutions for (iii) and (iv). For each case, we propose a *closure* operator, which yields closed boolean expressions. The closed expressions and their minimal generators give the most specific and most general boolean expressions that are satisfied by their corresponding transaction set. Further, the closed/minimal generator expressions form a lossless representation of all possible boolean expressions in the respective category.

2. The BLOSOM framework is efficiently designed to determine minimal generators for each of the four classes of boolean expressions, namely BLOSOM-MA for conjunctions, BLOSOM-MO for disjunctions, BLOSOM-MC for CNF and BLOSOM-MD for DNF expressions. In addition, BLOSOM can mine closed expressions for conjunctions and disjunctions, namely BLOSOM-CA for conjunctions and BLOSOM-CO for disjunctions. The framework also allows mining of closed CNF/DNF expressions using BLOSOM-CO and BLOSOM-CA as building blocks, though these have yet to be implemented.

3. BLOSOM employs a number of effective pruning techniques for searching over the space of boolean expressions, yielding orders of magnitude in speedup. These include: dynamic sibling reordering, parent-child relationship pruning, sibling merging, threshold pruning, and fast subsumption checking.

4. BLOSOM utilizes a novel *extraset* data structure for fast frequency computations, and to identify the corresponding transaction set for a given arbitrary boolean expression.

We conducted several experiments on both real and synthetic datasets to study the behavior of BLOSOM with respect to different input settings and parameter thresholds. We also compare BLOSOM with a state-of-the-art algorithm for mining minimal conjunction generators. From these experiments we can see that BLOSOM is much faster (two to orders of times) than related algorithms, and the result on real datasets prove to be meaningful.

CHAPTER 2
Related Work

In this chapter, we discuss related work for both subspace and boolean pattern mining. Within subspace patterns, we focus on mining clusters of coherent genes in 2D and 3D microarray datasets. We then look at related work in boolean pattern mining. We review the microarray data clustering algorithms first, most of which are subspace clustering methods. They can be categorized as follows.

2.1 Clustering on 2D Datasets

There are many subspace and biclustering algorithms proposed in the literature; see [26] for an excellent review. We briefly review the most relevant ones among these methods.

2.1.1 Feature-based Clustering

CLIQUE [3] introduced the problem of subspace clustering; it is a complete algorithm that finds axis-aligned dense subspaces. PROCLUS [1] and ORCLUS [2] use projective clustering to partition the dataset into clusters occurring in possibly different subsets of dimensions in a high dimensional dataset. PROCLUS seeks to find axis-aligned subspaces by partitioning the set of points and then uses a hill-climbing technique to refine the partitions. ORCLUS finds arbitrarily oriented clusters by using ideas related to singular value decomposition.

CLIFF [46] iterates between feature (genes) filtering and sample partitioning. It first calculates k best features (genes) according to their intrinsic discriminability using current partitions. Then it partitions the samples with these features by keeping the minimum normalized weights. This process iterates until convergence. COSA [18] allows traditional clustering algorithms to cluster on a subset of attributes, rather than all of them. Principal Component Analysis for gene expression clustering has also been proposed [49].

2.1.2 Graph-based Clustering

HCS [22] is a fullspace clustering algorithm. It cuts a graph into subgraphs by removing some edges, and repeats until all the vertices in each subgraph are similar enough. MST [47] is also a fullspace clustering method. It uses greedy method to construct a minimum spanning tree, and splits the current tree(s) repeatedly, until the average edges length in each subtree is below some threshold. Then each tree is a cluster.

SAMBA [42] uses a bipartite graph to model and implement the clustering. It repeatedly finds the maximal highly-connected sub-graph in the bipartite graph. Then it performs local improvement by adding or deleting a single vertex until no further improvement is possible. Other graph-theoretic clustering algorithms include CLICK [38] and CAST [10].

There are some common drawbacks concerning the above algorithms applied to microarray datasets. First, some of them are randomized methods based on shrinking and expansion, which sometimes results in incomplete clusters. Second, none of them can deal with overlapped clusters properly. Third, the greedy methods will lead to a local optimum that may miss some important (part of) clusters. Moreover, fullspace clustering is even not biclustering and will compromise the important clusters by considering irrelevant dimensions. In general, none of them are deterministic, and they cannot guarantee that all valid (overlapped) clusters are found.

2.1.3 Pattern-based Clustering

For microarray analysis, δ-biclustering [13] introduced the notion of biclusters. It uses mean squared residue of a submatrix $(X \times Y)$ to find biclusters. If a submatrix with enough size has a mean squared residue less than threshold δ, it is a δ-bicluster. Initially, the algorithm starts with the whole data matrix, and repeatedly adds/deletes a row/column from the current matrix in a greedy way until convergence. After having found a cluster, it replaces the submatrix with random values, and continues to find the next best cluster. This process iterates until no further clusters can be found. One limitation of δ-biclustering is that it may converge to a local optimum, and cannot guarantee all clusters will be found. Also it can easily miss overlapping clusters due to the random value substitutions it does. Another move-based δ-biclustering method was proposed in [48].

However, it too is an iterative improvement based method.

Among the recent methods most similar to MICROCLUSTER is pCluster [44], which defines a cluster C as a submatrix of the original dataset, such that for any two by two submatrix $\begin{bmatrix} c_{xa} & c_{xb} \\ c_{ya} & c_{yb} \end{bmatrix}$ of C, $|(c_{xa} - c_{ya}) - (c_{xb} - c_{yb})| < \delta$, where δ is a threshold. The algorithm first scans the dataset to find all column-pair and row-pair maximal clusters called MDS. Then it does the pruning in turn using the row-pair MDS and the column-pair MDS. It then mines the final clusters based on a prefix tree. pCluster is symmetric, i.e., it treats rows and columns equally, and it is capable of finding similar clusters as TRICLUSTER, but it does not merge/prune clusters and is not robust to noise. Further, we show that it runs much slower than MICROCLUSTER on real microarray datasets.

MaPle [32] is an extension of pCluster. By skipping trivial sub-clusters and pruning non-promising cluster candidates earlier, it achieves faster performance than pCluster (around 2 times based on their experiments). Both of them run slower than MICROCLUSTER which is 1.4 to 20 times faster than pCluster as shown later in this chapter. Some common drawbacks exist concerning pCluster and it's extensions. First, these methods not merge/prune similar clusters, and thus they cannot handle highly overlapping clusters with noise properly, which can be very common in real datasets. In contrast, MICROCLUSTER uses cluster deletion and merging to handle these problems. Second, they are approximately exponentially scalable with respect to the number of genes which is larger than the number of samples, whereas MICROCLUSTER can transpose the datasets, and is exponential only with the number of samples, and not genes.

xMotif [29] uses Monte Carlo method to find biclusters, and it requires all genes expressions in a bicluster to be similar across all the samples. It randomly picks a seed sample s and a sample subset d (called discriminating set), and then finds all such genes that are conserved across all the samples in d. xMotif uses Monte Carlo method to find the clusters that cover all genes and samples. However it cannot guarantee to find all the clusters because of its random sampling process. DOC [33] is a simiar Monte Carlo algirthm to xMotif [29].

Stochastic algorithm OPSM [9] has similar drawback as xMotif, but uses a different cluster definition. It defines a cluster as a submatrix of the original data matrix after

row and column permutation, whose row values are in a non-decreasing pattern. Another method using this definition is OP-Cluster [25]. Gene clustering methods using Self Organizing Maps [41], and iterated two-way clustering [43] have also been proposed; a systematic comparison these and other biclustering methods can be found in [26]. After finishing the subspace clustering algorithms for 2D data, let us proceed to the 3D data methods.

2.2 Clustering on 3D Datasets

There are not too many algorithms in this field, and according to our knowledge, none of them can actually mine 3D clusters except TRICLUSTER. While there has been work on mining gene expression patterns across time, to the best of our knowledge there is no previous method that mines tri-clusters. On the other hand, there are many full-space and biclustering algorithms designed to work with microarray datasets, such as feature based clustering [1, 2, 46], graph based clustering [22, 47, 42], and pattern based clustering [13, 25, 44, 29, 9], which have been discussed above. Below, we focus on discussing time-series expression clustering methods, which are related to the third extra dimension of TRICLUSTER — time.

Jiang et al [23] gave a method to analyze the gene-sample-time microarray data. It treats the gene-sample-time microarray data as a gene$\times$sample matrix with each entry as a vector of the values along the time dimension. For any two such vectors, it uses their *Pearson's correlation coefficient* as the distance. Then for each gene, it groups similar time-vectors together to form a sample subset. After that, it enumerates the subset of all the genes to find those subsets of genes whose corresponding sample subsets result in a considerable intersection set. The paper discussed two methods: grouping samples first and grouping genes first. Although the paper dealt with three dimensional microarray data, it considers the time dimension in full space (i.e., all the values along the time dimension), and is thus unable to find temporal trends that are applicable to only a subset of the times, and as such it casts the 3D problem into a biclustering problem.

In general, most previous methods apply traditional full space clustering (with some improvements) to the gene time-series data. Thus these methods are not capable of mining coherent subspace clusters, i.e., these methods sometimes will miss important infor-

mation obscured by the data noise. For example, Erdal et al. [15] extract a 0/1 vector for each gene, such that there is a '1' whenever there is a big change in its expression from one time to the next. Using longest common subsequence length as similarity, they perform a full-dimensional clustering. The subspaces of time-points are not considered, and the sample space is ignored. Moller et al [28] present another time-series microarray clustering algorithm. For any two time vectors $[x_1(t_1), x_2(t_2), \cdots, x_k(t_k)]$ and $[y_1(t_1), y_2(t_2), \cdots, y_k(t_k)]$ they calculate $sim(x, y) = \sum_{k=1}^{n} \frac{(x_{k+1}-x_k)-(y_{k+1}-y_k)}{t_{k+1}-t_k}$. Then they use a full-space repeated fuzzy clustering algorithm to partition the time-series clusters. Ramoni et al [35] presents a Bayesian method for model-based gene expression clustering. It represents gene expression dynamics as autoregressive equations and uses an agglomerative method to search for the clusters. Feng et al [16] proposed a time-frequency based full-space algorithm using a measure of functional correlation set between time-course vectors of different genes. Filkov et al [17] addressed the analysis of short-term time-series gene microarray data, by detecting the period in a predominantly cycling dataset, and the phase between phase-shifted cyclic datasets. It too is a fullspace clustering method on gene time-series data. For a more comprehensive look at time-series gene expression analysis, see the recent paper by Bar-Joseph [6]. The paper divides the computational challenges into four analysis levels: experimental design, analysis, pattern recognition and gene networks. It discusses the computational and biological problems at each level and reviews some methods proposed to deal with these issues. It also highlights some open problems.

2.3 Boolean Pattern Mining

Since the introduction of the redescription mining problem in [34], followup works [31, 52] have aimed to systematically formalize and generalize it, with a view toward designing scalable algorithms for finding redescriptions. The work in [52] deserves particular mention: CHARM-L [52] modifies CHARM [51] to find the minimal generators for itemsets (MA). It first mines the closed sets, builds a lattice, and then uses the lattice to extract the minimal generators. In contrast, BLOSOM-MA directly mines the MAs and runs much faster than CHARM-L.

Inferring arbitrary boolean expressions from datasets is a mainstay of the machine

learning and computational learning theory communities. Mitchell [27] proposed the concept of version spaces (which are basically a partial order over expressions) to organize the search for expressions consistent with a given set of data. The theoretical machine learning (PAC) community has focused on learning boolean expressions [11] in the presence of membership queries and equivalence queries. All these works conform to the classical supervised learning scenario where both positive and negative examples of the unknown function are supplied. In contrast, redescription mining aims to find boolean expressions without explicit direction about the examples they cover. It is hence more related to (conceptual) clustering of data given constraints on the form of the data descriptors.

Mining frequent itemsets (i.e., pure conjunctions) has been extensively studied within the context of association rule mining [4]. The closure operator for AND-clauses was proposed in [19], and the notion of minimal generators was introduced in [7]. Many algorithms for mining closed itemsets (see [51, 20]), and a few to mine minimal generators [7, 52], have also been proposed in the past. Within the association rule context, there has been previous work on mining negative rules [36, 45, 5], as well as disjunctive rules [30]. Unlike these methods we are interested in characterizing, via the closure operation, such rules within the general framework of boolean expression mining. Other work relevant to ours appears in [39], which proposes mining closed and minimal monotone DNF expressions, and in [37], which mines set-valued disjunctive rules. However, the use of closure operators and minimal generators for the different classes of boolean expressions, and the efficient, extensible BLOSOM framework for mining arbitrary expressions, are the novel contributions of our work.

Also related is the mining of optimal rules according to some constraints [8], since the boolean expressions can be considered as constraints on the patterns. More general notions of itemsets (including negated items and disjunctions) have been considered in the context of concise representations [12, 24]. Another point of comparison is w.r.t. the work of Gunopulos et al. [21] where the authors aim to find frequent and (maximally) interesting sentences w.r.t. a variety of criteria. Many data mining tasks, including inferring boolean functions, are instantiations of this problem.

With the background knowledge from related work, we can introduce our original

work on subspace/boolean pattern mining in the following chapters step by step.

CHAPTER 3

MICROCLUSTER: Mining Subspace Patterns from 2D Real-valued Datasets

The content of this chapter was published in part in paper [53].

3.1 Definitions and Problem Statement

Let $G = \{g_0, g_1, \cdots, g_{n-1}\}$ be a set of n genes, and let $S = \{s_0, s_1, \cdots, s_{m-1}\}$ be a set of m biological samples (or conditions). A microarray dataset is a real-valued $n \times m$ matrix $D = (G \times S \rightarrow R) = \{d_{ij}\}$ (with $i \in [0, n-1], j \in [0, m-1]$), whose rows and columns correspond to genes and samples, respectively. Each entry d_{ij} records the (absolute or relative) expression level of gene g_i in sample s_j. For example, Table 3.1(a) shows a dataset with 10 genes and 7 samples. The cells that should be filled with some random expression values have been left blank for clarity.

A *bicluster* C is a submatrix of the dataset D, where $C = (X \times Y \rightarrow R) = \{c_{ij}\}$, with $X \subseteq G$ and $Y \subseteq S$, provided certain conditions of homogeneity are satisfied. For example, a simple condition might be that all values c_{ij} are identical or approximately equal (constant pattern). Other homogeneity conditions can also be defined, such as similar column/row pattern, scaling/shifting pattern [44], etc. Let $\mathcal{B}$ be the set of all biclusters that satisfy the given homogeneity conditions, then $C = X \times Y \in \mathcal{B}$ is called a *maximal bicluster* iff there doesn't exist $C' = X' \times Y' \in \mathcal{B}$ such that $C \subset C'$ (i.e., $X \subset X'$ and $Y \subset Y'$).

Let $C = X \times Y$ be a bicluster, and let $\begin{bmatrix} c_{ia} & c_{ib} \\ c_{ja} & c_{jb} \end{bmatrix}$ be an arbitrary 2×2 submatrix of C. We call C a **cluster** iff it is a maximal bicluster satisfying the following properties:

1. Let $r_i = |\frac{c_{ib}}{c_{ia}}|$ and $r_j = |\frac{c_{jb}}{c_{ja}}|$, then $\frac{\max(r_i, r_j)}{\min(r_i, r_j)} - 1 \leq \varepsilon$, where ε is a maximum ratio threshold. This threshold ensures that the ratio of column values across any two rows in the cluster are similar to each other.

2. If $c_{ia} \times c_{ib} < 0$ then sign(c_{ia})=sign(c_{ja}) and sign(c_{ib})=sign(c_{jb}), where sign(x) re-

	s_0	s_1	s_2	s_3	s_4	s_5	s_6
g_0	3.6	1.0	1.0		1.0	1.0	1.0
g_1	3.0	2.5			2.0		1.0
g_2		5.0			5.0		5.0
g_3	6.6	5.5					2.0
g_4	9.0	7.5			6.0		3.0
g_5	6.6				4.4		2.0
g_6		3.0			3.0		3.0
g_7		8.0	8.0		8.0	8.0	
g_8	6.0	5.0			4.0		2.0
g_9		4.0	4.0		4.0	4.0	4.0

(a)

	s_3	s_0	s_6	s_4	s_1	s_5	s_2
g_5		6.6	2.0	4.4			
g_2			5.0	5.0	5.0		
g_6			3.0	3.0	3.0		
g_0		3.6	1.0	1.0	1.0	1.0	1.0
g_9			4.0	4.0	4.0	4.0	4.0
g_7				8.0	8.0	8.0	8.0
g_3		6.6	2.0		5.5		
g_4		9.0	3.0	6.0	7.5		
g_8		6.0	2.0	4.0	5.0		
g_1		3.0	1.0	2.0	2.5		

(b)

Table 3.1: (a) Example Microarray Dataset. (b) Some Clusters

turns -1 (1) if x is negative (non-negative) (expression values of zero are replaced with a small random positive correction value, in the preprocessing step). This allows us to easily mine datasets having negative expression values. [1]

3. Let $r_x = \frac{\max(c_{xa}, c_{xb})}{\min(c_{xa}, c_{xb})}$, then $r_x - 1 \leq \delta^r$, where $x \in \{i, j\}$ and δ^r is a maximum row range threshold. Thus the cluster can allow at most δ^r variation within the rows.

[1] It also prevents us from reporting that, for example, expression ratio $\frac{-5}{5}$ is equal to $\frac{5}{-5}$.

4. Let $r_y = \frac{\max(c_{iy}, c_{jy})}{\min(c_{iy}, c_{jy})}$, then $r_y - 1 \leq \delta^c$, where $y \in \{a, b\}$ and δ^c is a maximum column range threshold. Thus the cluster can allow at most δ^r variation within the columns.

5. $|X| \geq mr$ and $|Y| \geq mc$, where mr and mc denote minimum rows and column cardinality thresholds, respectively, so that we find only meaningfully large clusters.

Lemma 1 *(Symmetry Property)*

Let $C = X \times Y$ be a bicluster, and let $\begin{bmatrix} c_{ia} & c_{ib} \\ c_{ja} & c_{jb} \end{bmatrix}$ *be an arbitrary 2×2 submatrix of C. Let $r_i = |\frac{c_{ib}}{c_{ia}}|$, $r_j = |\frac{c_{jb}}{c_{ja}}|$, $r_a = |\frac{c_{ja}}{c_{ia}}|$, and $r_b = |\frac{c_{jb}}{c_{ib}}|$ then $\frac{\max(r_i, r_j)}{\min(r_i, r_j)} - 1 \leq \varepsilon \iff \frac{\max(r_a, r_b)}{\min(r_a, r_b)} - 1 \leq \varepsilon$.*

Proof: Without loss of generality assume that $r_i \geq r_j$, then $|\frac{c_{ib}}{c_{ia}}| \geq |\frac{c_{jb}}{c_{ja}}| \iff |\frac{c_{ja}}{c_{ia}}| \geq |\frac{c_{jb}}{c_{ib}}| \iff r_a \geq r_b$. Also $\frac{\max(r_i, r_j)}{\min(r_i, r_j)} = \frac{r_i}{r_j} = \frac{|c_{ib}/c_{ia}|}{|c_{jb}/c_{ja}|} = \frac{|c_{ja}/c_{ia}|}{|c_{jb}/c_{ib}|} = \frac{r_a}{r_b} = \frac{\max(r_a, r_b)}{\min(r_a, r_b)}$. ■

Lemma 2 *(Shifting Cluster)*

Let $C = X \times Y = \{c_{xy}\}$ be a maximal bicluster, and let $C_{2,2} =$ $\begin{bmatrix} c_{ia} & c_{ib} \\ c_{ja} & c_{jb} \end{bmatrix}$ *be an arbitrary 2×2 submatrix of C. Let $e^C = \{e^{c_{xy}}\}$ be the bicluster obtained by applying the exponential function (base e) to each value in C. If e^C is a* **cluster***, then C is a shifting cluster. Specifically if $c_{xb} = c_{xa} + v_x$, with $v_x \leq \rho$, for all $x \in \{i, j\}$, for all $C_{2,2} \subseteq C$, then C is a sample shifting cluster. Also if $c_{iy} = c_{jy} + v_y$, with $v_y \leq \rho$, for all $y \in \{a, b\}$, then C is a gene shifting cluster.*

Proof: Let's consider the shifting pattern in the samples first. Let $\begin{bmatrix} e^{c_{ia}} & e^{c_{ib}} \\ e^{c_{ja}} & e^{c_{jb}} \end{bmatrix}$ be a 2×2 submatrix of e^C. Without loss of generality assume that $r_i \geq r_j$, then $\frac{r_i}{r_j} = \frac{|e^{c_{ib}}/e^{c_{ia}}|}{|e^{c_{jb}}/e^{c_{ja}}|} = \frac{|e^{v_i}|}{|e^{v_j}|}$. It follows that if we set $\varepsilon \geq \frac{|e^{v_i}|}{|e^{v_j}|} - 1$, then e^C is a **cluster**. The proof for a shifting pattern in the gene values is similar. ■

Table 3.1(b) shows some examples of different clusters that can be obtained by some row/column permutations in our example dataset. Let $mr = mc = 3$, and let $\varepsilon = 0.01$. If we let $\delta^r = \delta^c = \infty$ (i.e. unconstrained), then $C_1 = \{g_1, g_4, g_8\} \times \{s_0, s_1, s_4, s_6\}$ is a scaling cluster, i.e., each row/column is some scalar multiple of another row/column. We also discover two other maximal overlapping clusters, $C_2 = \{g_0, g_2, g_6, g_9\} \times \{s_1, s_4, s_6\}$,

$C_3 = \{g_0, g_7, g_9\} \times \{s_1, s_2, s_4, s_5\}$. Note that if we set $mc = 2$ we would find another maximal cluster $C_4 = \{g_0, g_2, g_6, g_7, g_9\} \times \{s_1, s_4\}$, which is subsumed by C_2 and C_3. We shall see later that MICROCLUSTER can optionally delete such a cluster in the final steps, and merge mined clusters if certain overlapping criteria are met.

Note that one of the features of MICROCLUSTER is that different choices of row and column range thresholds (δ^r and δ^c) allow MICROCLUSTER flexibility to produce different kinds of clusters. For example, if $\delta^r = \delta^c \approx 0$, then we obtain clusters with approximately identical values. We obtain row (column) constant clusters by constraining only $\delta^r \approx 0$ ($\delta^c \approx 0$). When $\delta^r \neq 0$ and $\delta^c \neq 0$, we obtain a scaling cluster, and by Lemma 2 we can mine shifting clusters as well.

3.2 The MICROCLUSTER Algorithm

As outlined above, MICROCLUSTER mines arbitrarily positioned and overlapping scaling and shifting patterns from a two dimensional dataset. Typically microarray datasets have more genes than samples, i.e. $|G| \geq |S|$. Due to the symmetric property, MICROCLUSTER always mines a dataset that has more rows than columns by transposing the input dataset (matrix) if necessary. MICROCLUSTER has three main steps:

1. Find the valid ratio-ranges for all pair of columns, and construct a range multigraph.
2. Mine the maximal clusters from the range multigraph.
3. Optionally, delete or merge clusters if certain overlapping criteria are met.

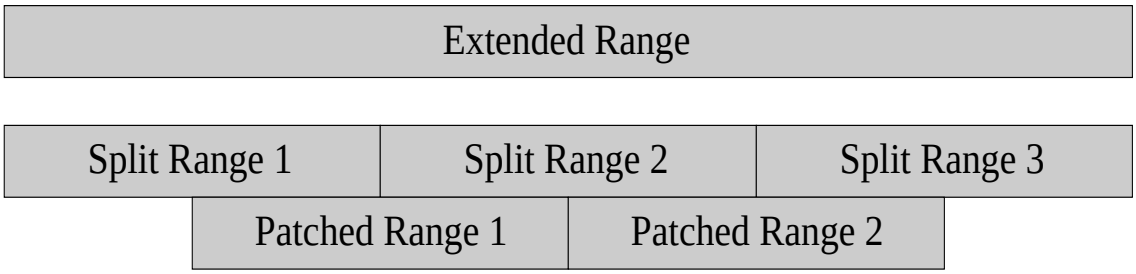
We look at each step below.

3.2.1 Construct Range Multigraph

Given a dataset D, the minimum row and column thresholds, mr and mc, and the maximum ratio threshold ε, let s_a and s_b be any two sample columns in D and let $r_x^{ab} = \frac{d_{xa}}{d_{xb}}$ be the ratio of the expression values of gene g_x in columns s_a and s_b, where $x \in [0, n-1]$. A *ratio range* is defined as an interval of ratio values, $[r_l, r_u]$, with $r_l \leq r_u$. Let $\mathcal{G}_{ab}([r_l, r_u]) = \{g_x : r_x^{ab} \in [r_l, r_u]\}$ be the set of genes, called the *gene-set*, whose ratios w.r.t. columns s_a and s_b lie in the given ratio range, and if $r_x^{ab} < 0$ all the values in the same column have same signs (negative/non-negative).

s0/s6	3.0	3.0	3.0	3.3	3.3	3.6
Row	g1	g4	g8	g3	g5	g0

(a)



(b)

Figure 3.1: (a) Sorted Ratios of Column s_0/s_6 in Table 3.1. (b) Split and Patched Ranges

In the first step MICROCLUSTER quickly tries to summarize the valid ratio ranges that can contribute to some bicluster. More formally, we call a ratio range *valid* iff:

1. $\frac{max(|r_u|,|r_l|)}{min(|r_u|,|r_l|)} - 1 \leq \varepsilon$, i.e., the range satisfies the maximum ratio threshold imposed in our cluster definition.

2. $|\mathcal{G}_{ab}([r_l, r_u])| \geq mr$, i.e., there are enough (at least mr) genes in the gene-set. This is imposed since our cluster definition requires any cluster to have at least mr genes.

3. If there exists a $r_x^{ab} < 0$, all the values in the same column ($\{d_{xa}\}$ or $\{d_{xb}\}$) have same signs (negative/non-negative).

4. $[r_l, r_u]$ is maximal w.r.t. ε, i.e., we cannot add another gene to $\mathcal{G}_{ab}([r_l, r_u])$ and yet preserve the ε bound.

Intuitively, we want to find all the maximal ratio ranges that satisfy the ε threshold, and span at least mx genes. Note that there can be multiple ranges between two columns and also that some genes may not belong to any range.

Figure 3.1 shows the ratio values for different genes using columns s_0/s_6 at time t_0 for our running example in Table 3.1. Assume $\varepsilon = 0.01$, and $mx = 3$, then there is

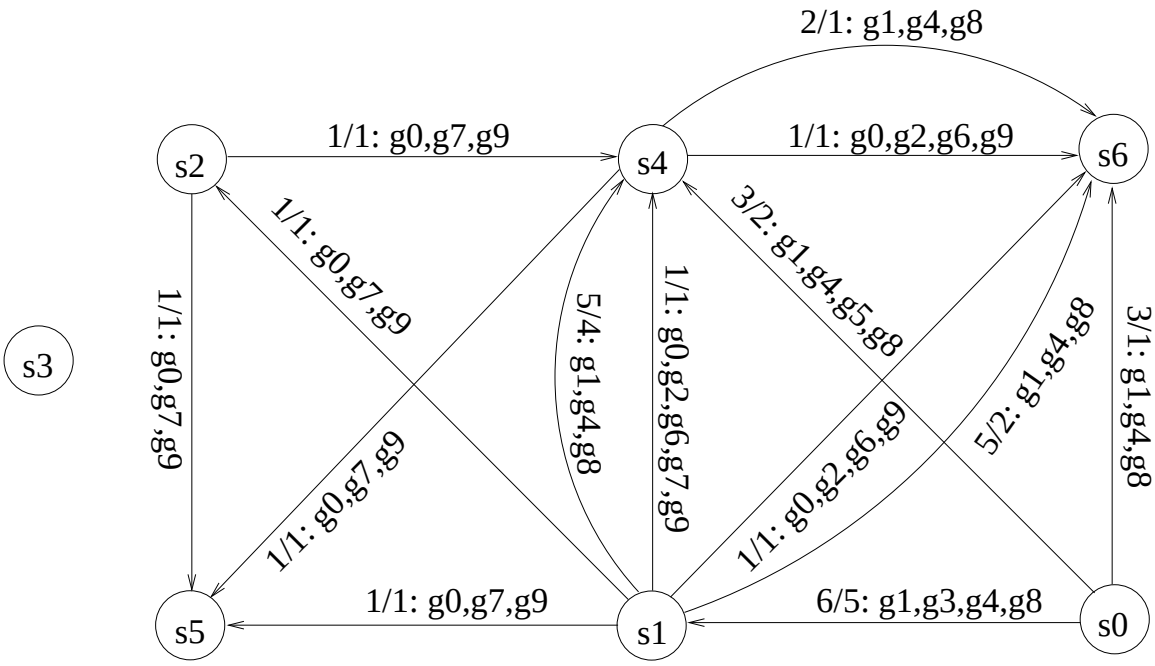


Figure 3.2: Weighted, Directed Range Multigraph

only one valid ratio range, $[3.0, 3.0]$ and the corresponding gene-set is $\mathcal{G}_{s_0 s_6}([3.0, 3.0]) = \{g_1, g_4, g_8\}$. Using a sliding window approach (with window size $r_x^{06} \times \varepsilon$ for each gene g_x) over the sorted ratio values, MICROCLUSTER finds all valid ratio ranges for all pairs of columns $s_a, s_b \in S$. If at any stage there are more than mx rows within the window, a range is generated. It is clear that different ranges may overlap. For instance, if we let $\varepsilon = 0.1$ we would obtain two valid ranges, $[3.0, 3.3]$ and $[3.3, 3.6]$, with overlapping gene-sets $\{g_1, g_4, g_8, g_3, g_5\}$ and $\{g_3, g_5, g_0\}$ respectively. If there are consecutive overlapping valid ranges, we merge them into an extended range, even though the maximum ratio threshold ε is exceeded. If the extended range is too wide, say more than 2ε, we split the extended range into several blocks of range at most 2ε (*split ranges*). To avoid missing any potential clusters, we also add some overlapping *patched ranges*. This process is illustrated in Figure 3.1(b). Note that an added advantage of allowing extended ranges is that it makes the method more robust to noise, since often the users may set a stringent ε condition, whereas the data might require a larger value.

Given the set of all valid, as well as the extended split or patched ranges, across any pairs of columns s_a, s_b with $a < b$, given as $\mathcal{R}^{ab} = \{R_i^{ab} = [r_{l_i}^{ab}, r_{u_i}^{ab}] \ : \ s_a, s_b \in S\}$, we construct a weighted, directed, range multigraph $M = (V, E)$, where $V = S$ (all

columns), and for each $R_i^{ab} \in \mathcal{R}^{ab}$ there exists a weighted, directed edge $(s_a, s_b) \in E$ with weight $w = \frac{r_{u_i}^{ab}}{r_{l_i}^{ab}}$. In addition, each edge in the range multigraph has associated with it the gene-set corresponding to the range on that edge. For example, suppose $mc = 3$, $mr = 3$, and $\varepsilon = 0.01$. Figure 3.2 shows the range multigraph constructed from Table 3.1.

3.2.2 Mine Biclusters from Range Multigraph

Input : parameters: $\varepsilon, mr, mc, \delta^r, \delta^c$, range graph M, set of genes G and samples S
Output : cluster set $\mathcal{C}$
Initialization: $\mathcal{C} = \emptyset$, call MICROCLUSTER($C = G \times \emptyset, S$)

MICROCLUSTER ($C = X \times Y, P$)::;

1 **if** C *satisfies* δ, δ^c **then**
2 **if** $|C.X| \geq mr$ **and** $|C.Y| \geq mc$ **then**
3 **if** $\nexists C' \in \mathcal{C}$, *such that* $C \subset C'$ **then**
4 Delete any $C'' \in \mathcal{C}$, if $C'' \subset C$
5 $\mathcal{C} \longleftarrow \mathcal{C} + C$
6 **foreach** $s_b \in P$ **do**
7 $C^{new} \longleftarrow C$
8 $C^{new}.Y \longleftarrow C^{new}.Y + s_b$
9 $P \longleftarrow P - s_b$
10 **if** $C.Y = \emptyset$ **then**
11 MICROCLUSTER(C^{new}, P)
  else
12 **forall** $s_a \in C.Y$ *and each* $R_i^{ab} \in \mathcal{R}^{ab}$ *satisfying* $|\mathcal{G}(R_i^{ab}) \cap C.X| \geq mr$ **do**
13 $C^{new}.X \longleftarrow (\bigcap_{all\ s_a \in C.Y} \mathcal{G}(R_i^{ab})) \cap C.X$
14 **if** $|C^{new}.X| \geq mr$ **then**
15 MICROCLUSTER(C^{new}, P)

Figure 3.3: MICROCLUSTER Algorithm

The process of constructing the range multigraph filters out most of the unrelated data, and the range multigraph represents in a compact way all the valid ranges that can be

used to mine potential biclusters. MICROCLUSTER uses a depth first search (DFS) on the range multigraph to mine all the clusters, as Figure 3.2.2 shows. It takes as input the set of parameter values $\varepsilon, mr, mc, \delta^r, \delta^c$, the range graph M, and genes G and samples S. It will output the final set of all clusters $\mathcal{C}$. MICROCLUSTER is a recursive algorithm, that at each call accepts a current *candidate* cluster $C = X \times Y$, and a set of not yet expanded samples P. The initial call is made with a cluster $C = G \times \emptyset$ with all genes G, but no samples, and with $P = S$, since we have not processed any samples yet. Before passing C to the recursive call we make sure that $|C.X| \geq mx$ (which is certainly true in the initial call, and also at line 14). Lines 1-5 check whether the current candidate C meets the minimum size thresholds mr and mc (line 1), and also row and column similarity thresholds δ^r and δ^c (line 2). If so, we next check if it is already contained by some maximal cluster $C' \in \mathcal{C}$ (line 3). If not, then we add C to the set of final clusters $\mathcal{C}$ (line 5), and remove any cluster $C'' \in \mathcal{C}$ already subsumed by C (line 4). Lines 6-15 generate a new candidate cluster by expanding the current candidate by one more sample, constructing the appropriate gene-set for the new candidate, before making a recursive call. MICROCLUSTER begins by adding to the current cluster C each new sample $s_b \in P$ (line 6), to obtain a new candidate C^{new} (line 7-8). Samples already processed are removed from P (line 9). Let s_a be all samples added to C until the previous recursive call. If no previous vertex s_a exists (which happens initially), then we simply call MICROCLUSTER with the new candidate. Otherwise, MICROCLUSTER tries all combinations of each qualified range edge R_i^{ab} between s_a and s_b for all $s_a \in C.Y$ (line 12), obtains their gene-set intersection $\bigcap_{all\ s_a \in C.Y} \mathcal{G}(R_i^{ab})$ and intersects with $C.X$ to obtain the valid genes in the new cluster C^{new} (line 13). If the new cluster has at least mr genes, then another recursive call to MICROCLUSTER is made (lines 14-15).

For example, let's consider how the clusters are mined from the range graph shown in Figure 3.2. Let $mr = 3, mc = 3, \varepsilon = 0.01$. Initially MICROCLUSTER starts at vertex s_0 with the candidate cluster $\{g_0, \cdots, g_9\} \times \{s_0\}$. We next process vertex s_1; since there is only one edge, we obtain a new candidate $\{g_1, g_3, g_4, g_8\} \times \{s_0, s_1\}$. From s_1 we process s_4 and consider both the edges: for $w = 5/4, \mathcal{G} = \{g_1, g_4, g_8\}$, we obtain the new candidate $\{g_1, g_4, g_8\} \times \{s_0, s_1, s_4\}$, but the other edge $w = 1/1, \mathcal{G} = \{g_0, g_2, g_6, g_7, g_9\}$ will not be extended since $\{g_0, g_2, g_6, g_7, g_9\} \cap \{g_1, g_3, g_4, g_8\} = \emptyset$. We then further process s_6. Out of

the two edges between s_4 and s_6, only one (with weight $2/1$) will yield a candidate cluster $\{g_1, g_4, g_8\} \times \{s_0, s_1, s_4, s_6\}$. Since this is maximal, and meets all parameters, at this point we have found one (C_1) of the three final clusters shown in Figure 3.1(b). Likewise, when we start from s_1, we will find the other two clusters $C_3 = \{g_0, g_7, g_9\} \times \{s_1, s_2, s_4, s_5\}$ and $C_2 = \{g_0, g_2, g_6, g_9\} \times \{s_1, s_4, s_6\}$. Intuitively, we are searching for maximal cliques (on samples), with cardinality at least mc, that also satisfy the minimum number of genes constraint mr.

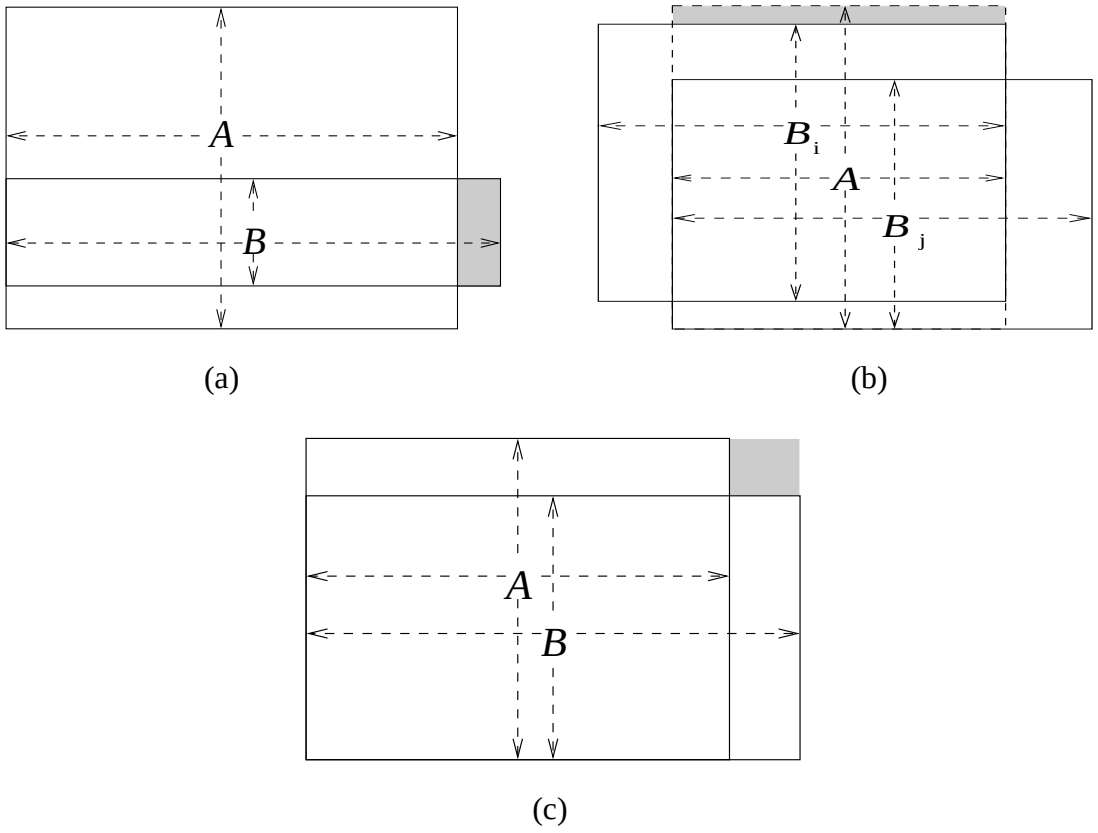


Figure 3.4: Three Pruning or Merging Cases

3.2.3 Merge and Prune Clusters

After mining the set of all clusters, MICROCLUSTER optionally merges or deletes certain clusters with large overlap. This is important, since real data can be noisy, and many clusters having large overlaps only make it harder for the users to select the important ones.

Let $A = X_A \times Y_A$, and $B = X_B \times Y_B$ be any two mined clusters. We define the

span of a cluster $C = X \times Y$, to be the set of gene-sample pairs that belong to the cluster, given as $L_C = \{(g_i, s_j) | g_i \in X, s_j \in Y\}$. Then we can define the following derived spans:

- $L_{A \cup B} = L_A \cup L_B$,
- $L_{A-B} = L_A - L_B$, and
- $L_{A+B} = L_{(X_A \cup X_B) \times (Y_A \cup Y_B)}$

If any of the following three overlap conditions are met, MICROCLUSTER either deletes or merges the clusters involved:

- **Delete B:** For any two clusters A and B, if $L_A > L_B$, and if $\frac{|L_{B-A}|}{|L_B|} < \eta$, then delete B. As illustrated in Figure 3.4(a), this means that if the cluster with the smaller span (B) has only a few extra elements, then delete the smaller cluster.
- **Delete A:** This is a generalization of case above. For a cluster A, if there exists a set of clusters $\{B_i\}$, such that $\frac{|L_A - L_{\bigcup_i B_i}|}{|L_A|} < \eta$, then delete cluster A. As shown in Figure 3.4(b), A is mostly covered by the $\{B_i\}$'s, then therefore can be deleted.
- **Merge A & B:** For two clusters A and B, if $\frac{|L_{A+B-A-B}|}{|L_{A+B}|} < \gamma$, merge A and B into one cluster $(X_A \cup X_B) \times (Y_A \cup Y_B)$. This case is shown in Figure 3.4(c). Here η, γ are user-defined thresholds.

3.2.4 Complexity Analysis

The range multigraph construction step takes time $O(|G||S|^2)$. The cluster mining step corresponds to constrained maximal clique enumeration, which is the most expensive. The precise number of clusters mined depends on the dataset and the input parameters. Nevertheless, for microarray datasets MICROCLUSTER is likely to be very efficient due to the following reasons: a) the range multigraph prunes away much of the noise and irrelevant information. b) MICROCLUSTER keeps intermediate gene-sets for all candidate clusters, so it can prune the search the moment input criteria are not met. The merging and pruning step apply only to those pairs of clusters that actually overlap, which can be determined in $O(|\mathcal{C}| \log |\mathcal{C}|)$ time.

3.2.5 Parameter Selection

The reader may notice that there are totally seven input parameters ($\varepsilon, mr, mc, \delta^r, \delta^c, \eta$ and γ), which are indispensable to generate different kinds of biclusters satisfying user requirements. Moreover it is relatively easy to set these parameters (we also show an experimental justification later). In the beginning, the user may not set η and γ, which help delete and merge clusters. These can be set later for removing highly overlapped clusters and/or for tolerating noise. δ^r and δ^c are used to restrict scaling or shifting patterns to be constant column/row patterns. These may be left unconstrained initially as well, to obtain more interesting clusters. However, if a user is interested in constant patterns only, then the two parameters can be set to gain further pruning and speedup.

For a specific dataset, ε decides the cluster pattern's accuracy, mr and mc decide the minimum size of a qualified cluster, and they together affect the running time for generating clusters. For MICROCLUSTER, ε and mr affect the number of edges between vertices in the multigraph, which affects the running speed and results. In contrast, mc affects the minimum search depth, and affects the pruning step. Since at first the user may not know the data distribution, we can set these three parameters to be highly constrained, i.e. small ε (e.g. 1% of the value range) and large mr and mc (e.g. 30% of original dataset size). Then according to concrete requirements and real running time, we can relax or strengthen these constraints. After having got the initial clusters, η and γ can be applied according to users' noise tolerance and overlapping allowance. In general, the users can control the input parameters for their specific needs to get interesting biclusters. MICROCLUSTER is by design capable of mining various kinds of patterns.

3.3 Experiments

Unless otherwise noted, all experiments were done on a Fedora virtual machine (448M memory, 1.4GHz, Pentium-M) over Windows XP through middleware VMware. We used both synthetic and real microarray datasets to evaluate MICROCLUSTER and to compare it with previous algorithms. The real datasets we used include the Yeast Elutriation (1536×14) (genome-www.stanford.edu/cellcycle/data/rawdata/individual.html), Yeast (2884×17) [13], CAMDA'04 (7091×46) (www.camda.duke.edu/camda04/), and Cancer (12625×13) (microarray.cnmcresearch.org/cancer_human.htm) datasets. Syn-

thetic datasets allow us to embed clusters, and then to test how MICROCLUSTER performs for varying input parameters. The input parameters to the generator are the total number of genes and samples, number of clusters to embed, percentage of overlapping clusters, row/column ranges for the cluster sizes, and the amount of noise for the expression values. Once all clusters are generated, the non-cluster regions are assigned random values.

3.3.1 Results on Synthetic Datasets

We first wanted to see how MICROCLUSTER behaves with varying input parameters in isolation. We generated synthetic data with the following default parameters: data matrix size 5000×40, number of clusters 10, cluster column size 8, cluster row size 200, percentage overlap 20%, noise level 3%. For each experiment, we kept all default parameters, except for the varying parameter. We also chose appropriate parameter values for MICROCLUSTER so that all embedded clusters were found. Figure 3.5(a)-(e) shows MICROCLUSTER's sensitivity to different parameters. We found that the time increases linearly with cluster row size (a), but exponentially with cluster column size (b). The time is also linear w.r.t. number of clusters (c), whereas the overlap percentage doesn't seem to have much impact on the time (d). Finally, as we add more noise, more time is required to mine the clusters, since there is a greater chance that a random gene or sample can belong to some cluster.

We next varied the total number of rows/columns in a dataset, but we kept the following default parameters: number of clusters 5, cluster column size range $[20\%, 27\%]$ (i.e., if there are $|S|$ total columns, a cluster will have column size between $0.2|S|$ and $0.27|S|$), cluster row size range $[10\% - 14\%]$, overlap percentage 20%, and noise 2%. Figure 3.5(f)-(g) show the running time. If we fix $|S| = 35$ and vary $|G|$, we find a linear increase in time (f), but if we fix $|G| = 800$ and vary $|S|$, we find that the time increases exponentially (g).

3.3.2 Effect of Parameter Changes on the Output

We define several metrics to analyze the output from different biclustering algorithms. If $\mathcal{C}$ is the set of all clusters output, then:

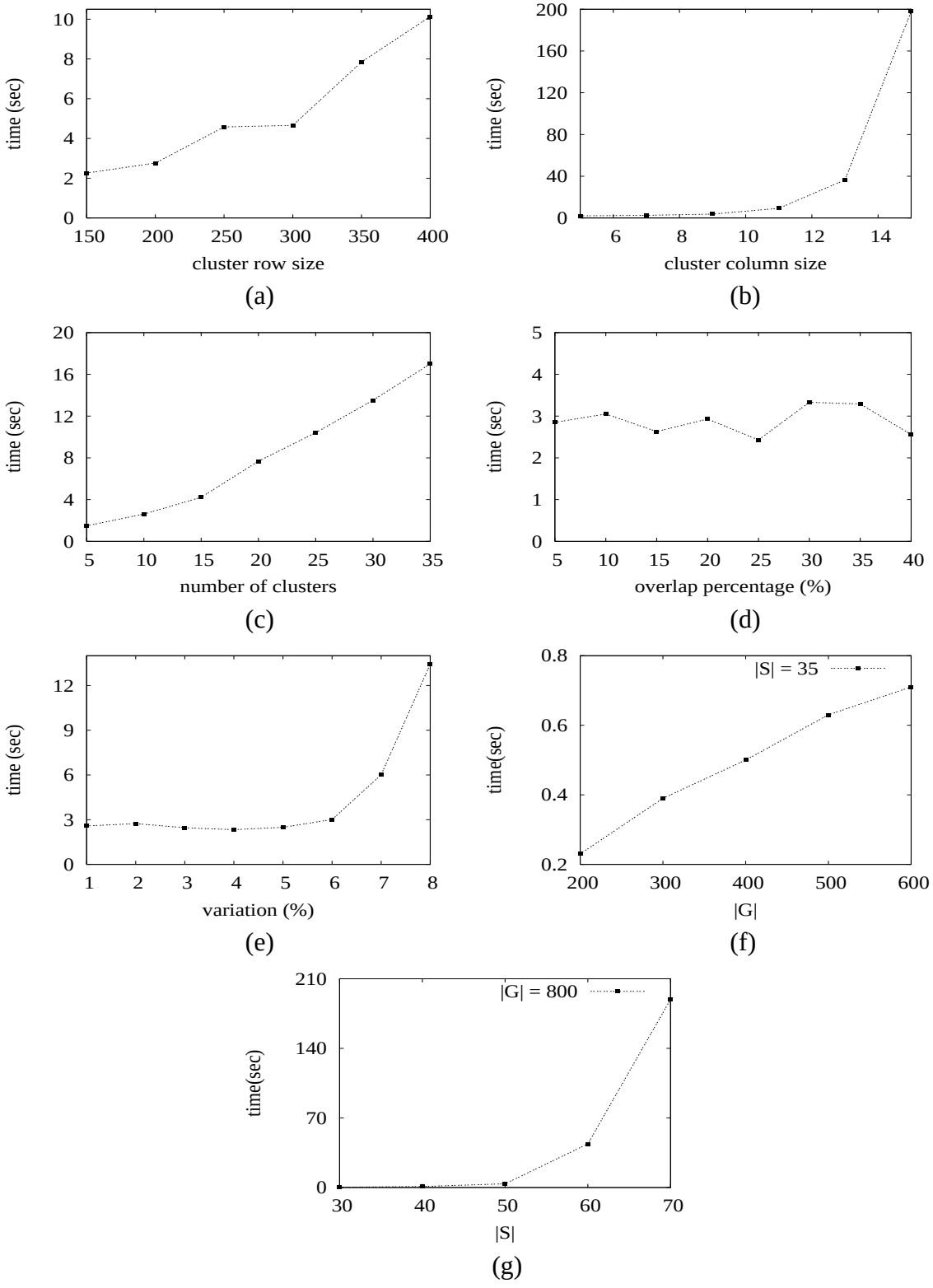
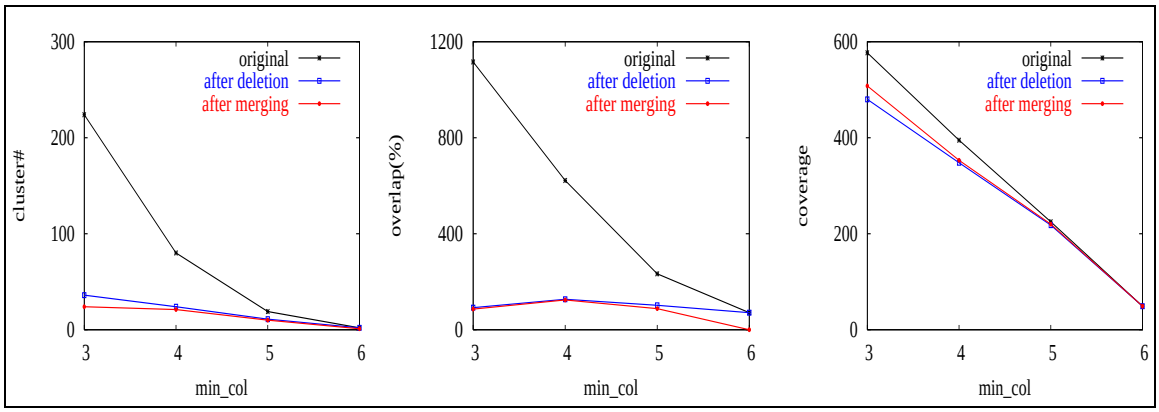
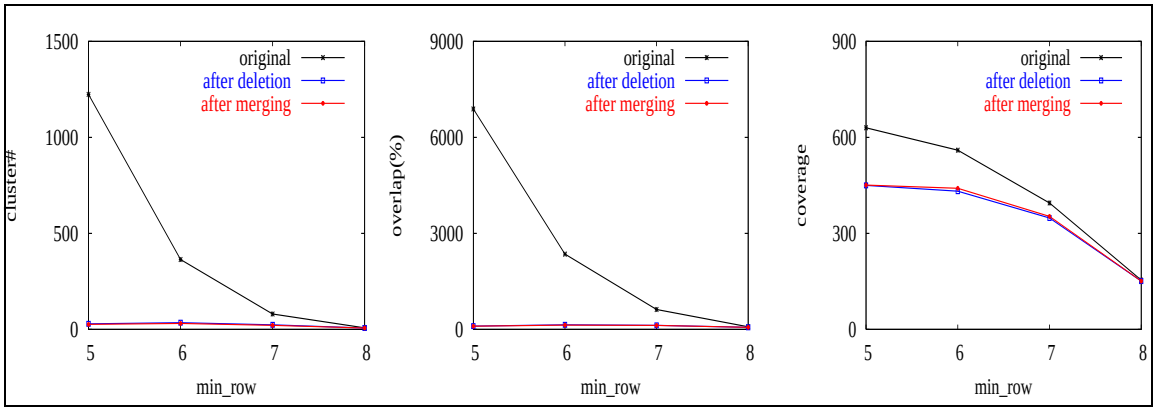


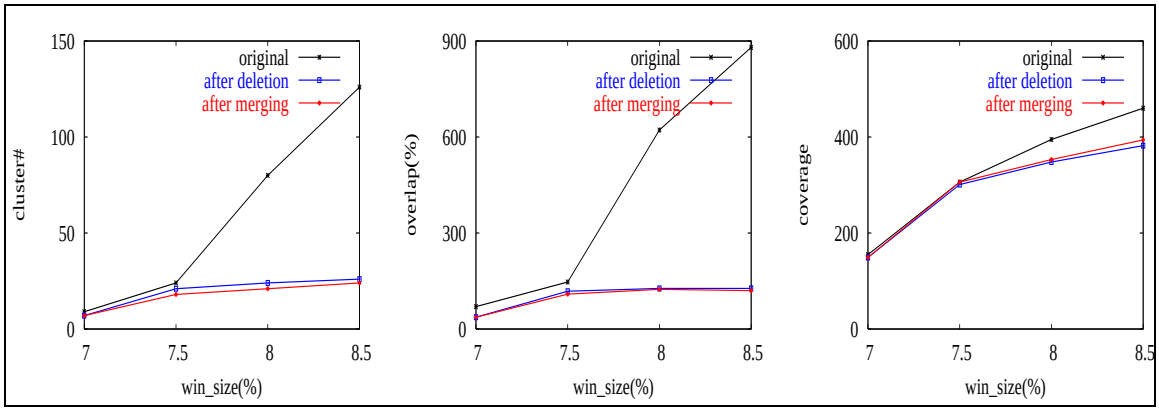
Figure 3.5: Evaluation of MICROCLUSTER **on Synthetic Data**



(a)

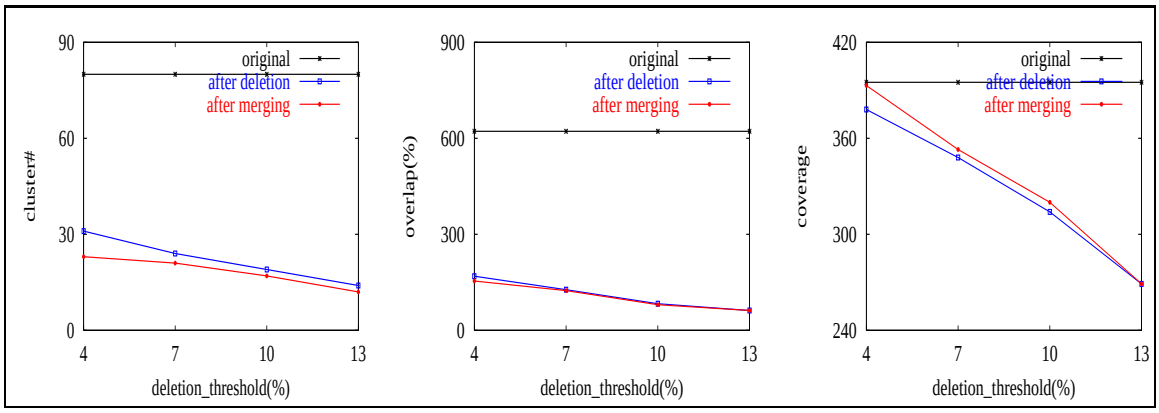


(b)

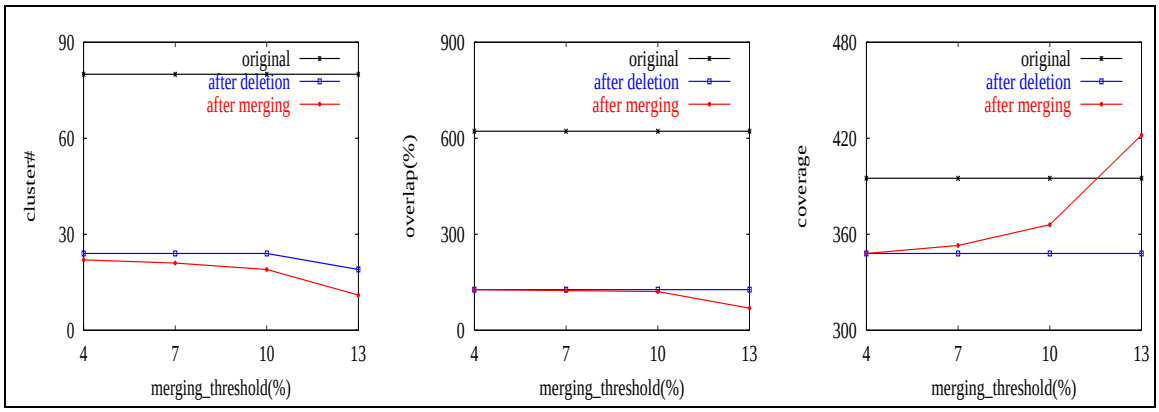


(c)

Figure 3.6: Effects of Parameter Changes on the Final Output



(a)



(b)

Figure 3.7: Effects of Merging and Deletion on the Final Output

- $Cluster\#$ is just $|\mathcal{C}|$,
- $Element\_Sum$ is the sum of the spans of all clusters, i.e., $Element\_Sum = \sum_{C \in \mathcal{C}} |L_C|$,
- $Coverage$ is the span of the union of all clusters, i.e., $Coverage = |L_{\bigcup_{C \in \mathcal{C}} C}|$, and finally
- $Overlap$ is given as $\frac{Element\_Sum - Coverage}{Coverage}$.

Note that $Element\_Sum$ will count an overlapping element as many times as the number of different clusters containing the element. On the other hand $Coverage$ counts each element only once, as long as it is part of some cluster. Thus $Overlap$ measures the degree of average overlap among the clusters. With these metrics, we can analyze the clustering results systematically.

We applied MICROCLUSTER to a real sub-dataset (50×14) of Yeast Elutriation dataset with different input parameters to analyze the effect of parameter changes on the final output, as Figure 3.6(a)-(c) shows. We also present the effects of merging and deletion as Figure 3.7(a)-(b) shows. The default parameter settings were $mc = 4, mr = 7, \varepsilon = 8\%, \eta = 7\%$ and $\gamma = 7\%$. For each experiment, we kept all default parameter settings, except for the varying parameter as marked by the x-axis.

From Figure 3.6(a)-(c), we can see that the original (before deletion and merging) number of clusters ($Cluster\#$) and $Overlap$ varies more than linearly with change in $mr/mc/\varepsilon$. However, after deletion and merging, $Cluster\#$ and $Overlap$ change gradually with the change of $mr/mc/\varepsilon$. The $Coverage$ actually reflects the real information conveyed by the clusters, which changes very little in the deletion/merging process as Figure 3.6(a)-(c) shows. In other words, the deletion and merging removes a lot of redundant information in the clusters, while retaining good coverage. This shows the usefulness of the deletion and merging steps. Figure 3.7(a)-(b) tell us that the change of η and γ affects the final $Cluster\#$, $Overlap$ and $Coverage$ (after deletion and merging) around linearly.

We also confirmed the robustness of MICROCLUSTER in these experiments. Between two sequential experiments (change one parameter only), the original clusters of one are always a subset of the other one, i.e. the cluster change generated by MICROCLUSTER is a consistent process. So users can always modify the input parameters to get the the best clustering (i.e., reach a trade-off point they are satisfied with).

3.3.3 Results from Real Microarray Datasets

Here we illustrate how MICROCLUSTER finds relevant clusters in real datasets. Figure 3.8(a) is an example bicluster mined on dataset CAMDA'04. Figure 3.8(c) and (d) are constant column and constant row cluster examples respectively mined on subset of Yeast dataset, where color-change (green-black-red) represents value range from 60 (light green) to 140 (light red). We also tested MICROCLUSTER on gene expression data of human cancer from 13 pediatric astrocytomas (6 high-grade and 7 low-grade) with 12625 genes. Our experiments show MICROCLUSTER can separate these two kinds of samples very well. It outputs one cluster with 254 genes and 6 samples ($\varepsilon = 0.02$), all belonging to the low-grade class. Figure 3.8(b) shows the gene expression pattern.

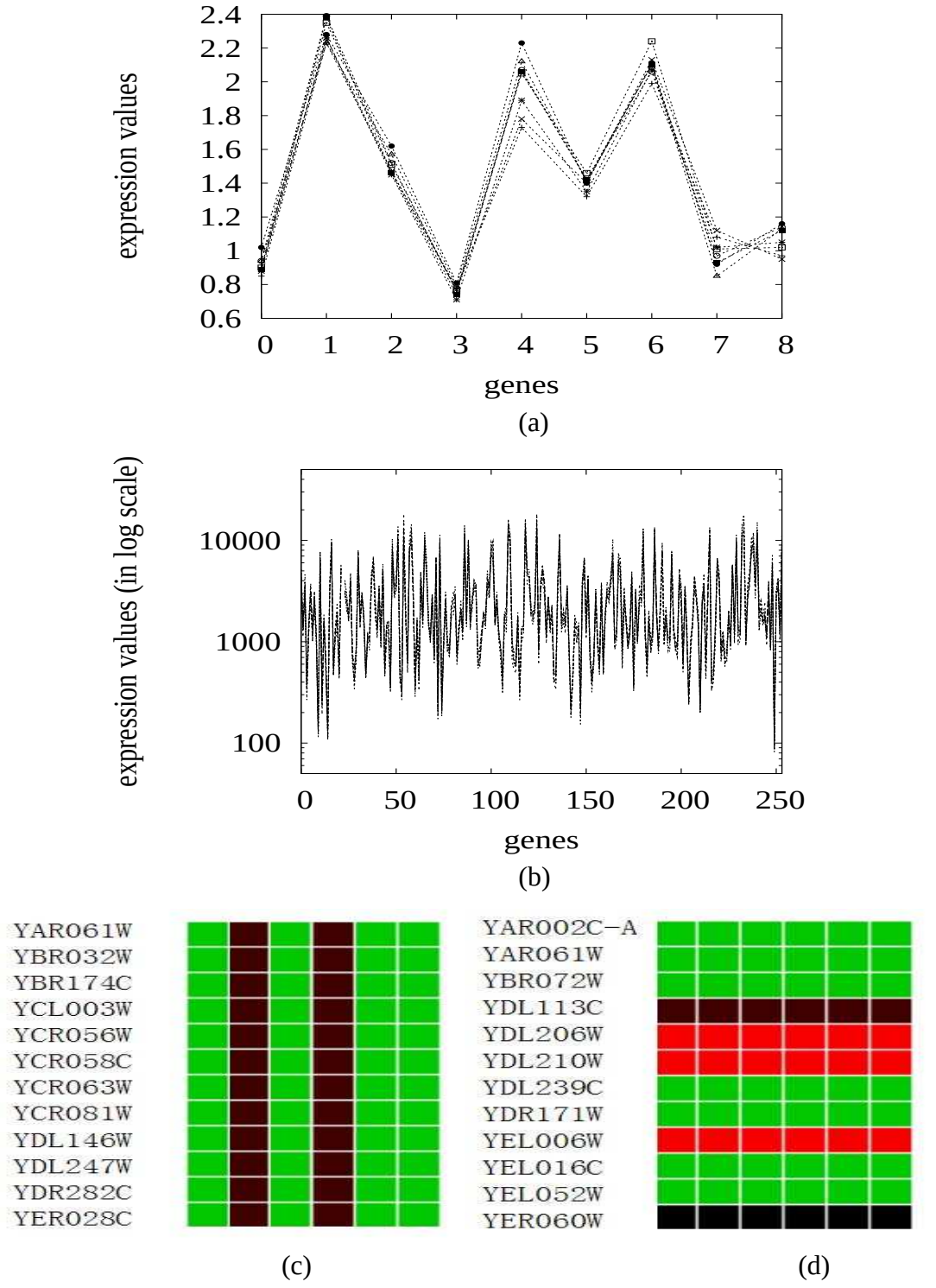


Figure 3.8: Example Clusters in the CAMDA'04 (a) and Human Cancer (b) Datasets, and a Constant Column (c), and Constant Row (d) Cluster

mr	mc	6			7			8		
		mClus.	pClus.	MaPle	mClus.	pClus.	MaPle	mClus.	pClus.	MaPle
40	Run Time(sec)	639	883	228	18	150	80	13	27	34
	$Cluster\#$	1195			106			5		
	$Element\_Sum$	333,502			32,782			1,688		
	$Coverage$	6,456			2,234			682		
45	Run Time(sec)	162	876	165	12	148	67	11	23	27
	$Cluster\#$	554			35			1		
	$Element\_Sum$	169,534			12,082			392		
	$Coverage$	5,349			1,483			392		
50	Run Time(sec)	43	871	138	11	146	58	10	22	20
	$Cluster\#$	243			11			0		
	$Element\_Sum$	81,456			4,200			0		
	$Coverage$	4,178			961			0		

Table 3.2: Comparison between MICROCLUSTER **and pCluster/MaPle on the Yeast Dataset**

We compared MICROCLUSTER with pCluster [44] and MaPle [32], which are also deterministic algorithms, however, they capture a subset of bicluster patterns that MICROCLUSTER does (i.e. they can get the same clusters if we let $\delta^r = \delta^c = \infty$ and $\eta = \gamma = 0$). We used a Cygwin/WindowsXP PC with 768MB memory and 1.4GHz Pentium-M processor. The pCluster and MaPle executable codes were provided by their authors. We used small thresholds (for cluster tolerance δ, mentioned in Chapter 2, we set for pCluster δ=1, and for MaPle δ=0. [2] For MICROCLUSTER we set ε=0.1%) to get the same outputs from all methods. Table 3.2 shows that whereas all of them found exactly the same clusters, MICROCLUSTER is 1.4 to 20 times faster than pCluster, and also faster than MaPle in most cases for the unbalanced microarray dataset.

We also checked if any of the clusters discovered by MICROCLUSTER share common gene process, function, or cellular location using the gene ontology (GO; www.gene ontology.org) project data, which aims at developing three structured, controlled vocabularies (ontologies) that describe gene products in terms of their associated biological processes, cellular components and molecular functions in a species-independent manner. We used the yeast genome gene ontology term finder (www.yeastgenome.org) to verify the biological significance of MICROCLUSTER's result (four clusters got from the Yeast dataset (2884×17) [13] with parameters $\varepsilon = 0.01, mr = 60, mc = 5$).

Table 3.3 shows the significant (with p-value less than 0.01) shared GO terms used to describe the set of genes in each cluster. Also, when there are multiple hierarchical

[2] When $\delta \geq 1$ MaPle runs very slowly (45 sec for 8×50, 615 sec for 8×45, and more than 2000 seconds for other combinations in Table 3.2) for unknown reasons.

Cluster	GO term	Gene Names	p-value
$C_1(62 \times 5)$	process: meiotic gene conversion	SAE3, REC114	0.00740
	process: cell cycle checkpoint	KCC4, RFX1, CSM3	0.00768
$C_2(62 \times 5)$	process: meiotic DNA recombinase assembly	RAD57, SAE3	0.00086
	process: carbohydrate transport	MAL31, MPH2, YEA4	0.00265
	function: solute:cation symporter activity	MAL31, UGA4, FCY21	0.00012
	function: transporter activity	MAL31, UGA4, MPH2, YEA4, YEL006W, FCY21, YFL054C, PMC1, VHT1, QDR1, YMR034C	0.00085
	function: carbohydrate transporter activity	MAL31, MPH2, YEA4	0.00201
$C_3(60 \times 5)$	process: transport, establishment of localization	ERP1, MAL31, ATG20, UGA4, MPH2, YEA4, YEL006W, YFL054C, PMC1, VHT1, MIP6, QDR1, PEP8, SAL1, LAS17, SYT1	0.0018
	process: response to drug	SNG1, SLI1, QDR1	0.00220
	function: solute:cation symporter activity	MAL31, UGA4, FCY21	0.00011
	function: transporter activity	MAL31, UGA4, MPH2, YEA4, YEL006W, FCY21, YFL054C, PMC1, VHT1, QDR1, YMR034C, SAL1	0.00015
	function: metal ion binding	SAL1, IZH4	0.00695
	location: membrane	ATG20, UGA4, ADY3, MPH2, COQ4, YEL006W, FCY21, YFL054C, PMC1, VHT1, SNG1, SLI1, MIP6, QDR1, PEP8, SAL1, IZH4	0.00203
$C_4(61 \times 5)$	process: carbohydrate transport	MAL31, MPH2, YEA4	0.00253
	process: polyamine transport	UGA4, TPO2	0.00327
	process: transport, establishment of localization	ERP1, SSA3, MAL31, ATG20, UGA4, MPH2, YEA4, YEL006W, YFL054C, PMC1, VHT1, TPO2, VMR1, QDR1, PEP5	0.00570
	function: transporter activity	MAL31, UGA4, MPH2, YEA4, YEL006W, FCY21, YFL054C, PMC1, VHT1, TPO2, VMR1, QDR1, YMR034C	4.26e-05
	function: solute:cation symporter activity	MAL31, UGA4, FCY21	0.00011
	function: carbohydrate transporter activity	MAL31, MPH2, YEA4	0.00191
	function: polyamine transporter activity	UGA4, TPO2	0.00912
	location: vacuolar membrane	UGA4, PMC1, TPO2, PEP5	0.00871

Table 3.3: Significant Shared GO Terms (Process, Function, Component) for Genes in Different Clusters

terms involving the same group of genes, we only report the most significant terms. For example for cluster C_1, we find significant genes involved in meiotic gene conversion and cell cycle checkpoint. We can see that there exists differences in the clusters. There are some similarities also. These clusters are related to mainly meosis, transport, and localization. Some genes are localized in membranes.

For comparison, we also applied SAMBA [42] and xMotif [29] to the same Yeast dataset (2884×17) with the same minimum cluster size requirements (i.e. $mr = 60$ and $mc = 5$). However, SAMBA found four highly similar clusters (each containing around 140 genes), any two of which overlap more than 97%. So in fact, SAMBA got only one cluster containing 140 genes. xMotif found three clusters (containing 36, 86 and 120 genes respectively), which had some similarities to our clusters, however, many genes reported in a cluster had "unknown biological process/function" as a significant GO term. Further it gave different results each time even with the same input parameters because of its randomized nature. The above results indicate that MICROCLUSTER can find potentially biologically significant clusters, where other approaches might not be so effective.

3.4 Conclusions

In this chapter we introduced a novel deterministic biclustering algorithm called MICROCLUSTER, which can mine arbitrarily positioned and overlapping clusters. Depending on different parameter values, MICROCLUSTER can mine different types of clusters, including those with constant or similar row/column values, similar row values, similar column values, as well as scaling and shifting expression patterns. MICROCLUSTER first constructs a range multigraph, which is a compact representation of all similar value ranges in the dataset between any two columns. It then searches for constrained maximal cliques in this multigraph to yield the final set of biclusters. Optionally, MICROCLUSTER merges/deletes some clusters having large overlaps. We present a useful set of metrics to evaluate the clustering quality, and we show that MICROCLUSTER can outperform a related biclustering method by a factor of 20.

CHAPTER 4

TRICLUSTER: Mining Subspace Patterns from 3D Real-valued Datasets

The content of this chapter was published in part in paper [54].

4.1 Definitions and Problem Statement

The concepts and lemmas in this section are extensions from 2D (given in Chapter 3) case to the 3D case. Let $G = \{g_0, g_1, \cdots, g_{n-1}\}$ be a set of n genes, let $S = \{s_0, s_1, \cdots, s_{m-1}\}$ be a set of m biological samples (e.g., different tissues or experiments), and let $T = \{t_0, t_1, \cdots, t_{l-1}\}$ be a set of l experimental time points. A three dimensional microarray dataset is a real-valued $n \times m \times l$ matrix $D = (G \times S \times T \rightarrow R) = \{d_{ijk}\}$ (with $i \in [0, n-1], j \in [0, m-1], k \in [0, l-1]$), whose three dimensions correspond to genes, samples and times respectively (note that the 3rd dimension can also be a spatial region of interest, but without loss of generality (w.l.o.g.), we will consider time as the 3rd dimension). Each entry d_{ijk} records the (absolute or relative) expression level of gene g_i in sample s_j at time t_k. For example, Table 4.1(a) shows a dataset with 10 genes, 7 samples and 2 time points. For clarity certain cells have been left blank; we assume that these are filled by some random expression values.

A *tricluster* C is a submatrix of the dataset D, where $C = X \times Y \times Z = \{c_{ijk}\}$, with $X \subseteq G$, $Y \subseteq S$ and $Z \subseteq T$ provided certain conditions of homogeneity are satisfied. The definition is an extension of *bicluster*'s definition in Chapter 3 for the 3D case. For example, a simple condition might be that all values $\{c_{ijk}\}$ are identical or approximately equal. If we are interested in finding common gene co-expression patterns across different samples and times, we can find clusters that have similar values in the G dimension, but can have different values in the S and T dimensions. Other homogeneity conditions can also be defined, such as similar values in S dimension, order preserving submatrix, and so on [26].

Let $C = X \times Y \times Z = \{c_{ijk}\}$ be a tricluster and let $C_{2,2} = \begin{bmatrix} c_{ia} & c_{ib} \\ c_{ja} & c_{jb} \end{bmatrix}$ be any

Time t_0

	s_0	s_1	s_2	s_3	s_4	s_5	s_6
g_0	3.6	1.0	1.0		1.0	1.0	1.0
g_1	3.0	2.5			2.0		1.0
g_2		5.0			5.0		5.0
g_3	6.6	5.5					2.0
g_4	9.0	7.5			6.0		3.0
g_5	6.6				4.4		2.0
g_6		3.0			3.0		3.0
g_7		8.0	8.0		8.0	8.0	
g_8	6.0	5.0			4.0		2.0
g_9		4.0	4.0		4.0	4.0	4.0

	s_3	s_0	s_6	s_4	s_1	s_5	s_2
g_5		6.6	2.0	4.4			
g_2			5.0	5.0	5.0		
g_6			3.0	3.0	3.0		
g_0		3.6	1.0	1.0	1.0	1.0	1.0
g_9			4.0	4.0	4.0	4.0	4.0
g_7				8.0	8.0	8.0	8.0
g_3		6.6	2.0		5.5		
g_4		9.0	3.0	6.0	7.5		
g_8		6.0	2.0	4.0	5.0		
g_1		3.0	1.0	2.0	2.5		

Time t_1

	s_0	s_1	s_2	s_3	s_4	s_5	s_6
g_0		0.5	0.5		0.5	0.5	0.5
g_1		3.0			2.4		1.2
g_2		2.5			2.5		2.5
g_3		5.5					2.0
g_4		9.0			7.2		3.6
g_5					4.4		2.0
g_6		1.5			1.5		1.5
g_7		4.0	4.0		4.0	4.0	
g_8		6.0			4.8		2.4
g_9		2.0	2.0		2.0	2.0	2.0

(a)

	s_3	s_0	s_6	s_4	s_1	s_5	s_2
g_5							
g_2			2.5	2.5	2.5		
g_6			1.5	1.5	1.5		
g_0			0.5	0.5	0.5	0.5	0.5
g_9			2.0	2.0	2.0	2.0	2.0
g_7				4.0	4.0	4.0	4.0
g_3							
g_4			3.6	7.2	9.0		
g_8			2.4	4.8	6.0		
g_1			1.2	2.4	3.0		

(b)

Table 4.1: (a) Example Microarray Dataset. (b) Some Clusters

arbitrary 2×2 submatrix of C, i.e., $C_{2,2} \subseteq X \times Y$ (for some $z \in Z$) or $C_{2,2} \subseteq X \times Z$(for some $y \in Y$) or $C_{2,2} \subseteq Y \times Z$(for some $x \in X$). We call C a *scaling cluster* if we have $c_{ib} = \alpha_i c_{ia}$ and $c_{jb} = \alpha_j c_{ja}$, and further $|\alpha_i - \alpha_j| \leq \varepsilon$, i.e., the expression values differ by an approximately (within ε) constant *multiplicative factor* α. We call C a *shifting cluster* iff we have $c_{ib} = \beta_i + c_{ia}$ and $c_{jb} = \beta_j + c_{ja}$, and further $|\beta_i - \beta_j| \leq \varepsilon$, i.e., the expression values differ by a approximately (within ε) constant *additive factor* β.

We say that cluster $C = X \times Y \times Z$ is a subset of $C' = X' \times Y' \times Z'$, iff $X \subseteq X'$, $Y \subseteq Y'$ and $Z \subseteq Z'$. Let $\mathcal{B}$ be the set of all triclusters that satisfy the given homogeneity conditions, then $C \in \mathcal{B}$ is called a *maximal tricluster* iff there doesn't exist another cluster

$C' \in \mathcal{B}$ such that $C \subset C'$ Let $C = X \times Y \times Z$ be a tricluster, and let $C_{2,2} = \begin{bmatrix} c_{ia} & c_{ib} \\ c_{ja} & c_{jb} \end{bmatrix}$ an arbitrary 2×2 submatrix of C, i.e., $C_{2,2} \subseteq X \times Y$ (for some $z \in Z$) or $C_{2,2} \subseteq X \times Z$ (for some $y \in Y$) or $C_{2,2} \subseteq Y \times Z$ (for some $x \in X$). Now let us extend *microcluster*'s definition in Chapter 3 for the 3D case. We call C a **valid cluster** iff it is a maximal tricluster satisfying the following properties:

1. Let $r_i = |\frac{c_{ib}}{c_{ia}}|$ and $r_j = |\frac{c_{jb}}{c_{ja}}|$ be the ratio of two column values for a given row (i or j). We require that $\frac{\max(r_i,r_j)}{\min(r_i,r_j)} - 1 \leq \varepsilon$, where ε is a maximum ratio value.

2. If $c_{ia} \times c_{ib} < 0$ then sign(c_{ia}) = sign(c_{ja}) and sign(c_{ib}) = sign(c_{jb}), where sign(x) returns -1/1 if x is negative/non-negative (expression values of zero are replaced with a small random positive correction value, in the preprocessing step). This allows us to easily mine datasets having negative expression values.

3. We require that the cluster satisfy maximum range thresholds along each dimension. For any $c_{i_1j_1k_1} \in C$ and $c_{i_2j_2k_2} \in C$, let $\delta = |c_{i_1j_1k_1} - c_{i_2j_2k_2}|$. We require the following conditions: a) If $j_1 = j_2$ and $k_1 = k_2$, then $\delta \leq \delta^x$, b) if $i_1 = i_2$ and $k_1 = k_2$, then $\delta \leq \delta^y$, and c) if $i_1 = i_2$ and $j_1 = j_2$, then $\delta \leq \delta^z$. where δ^x, δ^y and δ^z represent the maximum range of expression values allowed along the gene, sample, and time dimensions.

4. We require that $|X| \geq mx$, $|Y| \geq my$ and $|Z| \geq mz$, where mx, my and mz denote minimum cardinality thresholds for each dimension.

Lemma 3 *(Symmetry Property for 3D Case - an extension of Lemma 1)*
Let $C = X \times Y \times Z$ be a tricluster, and let $\begin{bmatrix} c_{ia} & c_{ib} \\ c_{ja} & c_{jb} \end{bmatrix}$ an arbitrary 2×2 submatrix of $X \times Y$ (for some $z \in Z$) or $X \times Z$(for some $y \in Y$) or $Y \times Z$(for some $x \in X$). Let $r_i = |\frac{c_{ib}}{c_{ia}}|$, $r_j = |\frac{c_{jb}}{c_{ja}}|$, $r_a = |\frac{c_{ja}}{c_{ia}}|$, and $r_b = |\frac{c_{jb}}{c_{ib}}|$ then $\frac{\max(r_i,r_j)}{\min(r_i,r_j)} - 1 \leq \varepsilon \iff \frac{\max(r_a,r_b)}{\min(r_a,r_b)} - 1 \leq \varepsilon$.

Proof: Without loss of generality assume that $r_i \geq r_j$, then $|\frac{c_{ib}}{c_{ia}}| \geq |\frac{c_{jb}}{c_{ja}}| \iff |\frac{c_{ja}}{c_{ia}}| \geq |\frac{c_{jb}}{c_{ib}}| \iff r_a \geq r_b$. We now have $\frac{\max(r_i,r_j)}{\min(r_i,r_j)} = \frac{r_i}{r_j} = \frac{|c_{ib}/c_{ia}|}{|c_{jb}/c_{ja}|} = \frac{|c_{ja}/c_{ia}|}{|c_{jb}/c_{ib}|} = \frac{r_a}{r_b} = \frac{\max(r_a,r_b)}{\min(r_a,r_b)}$. Thus $\frac{\max(r_i,r_j)}{\min(r_i,r_j)} - 1 \leq \varepsilon \iff \frac{\max(r_a,r_b)}{\min(r_a,r_b)} - 1 \leq \varepsilon$. ■

The symmetric property of our cluster definition allows for very efficient cluster mining. The reason is that we are now free to mine clusters searching over the dimensions with the least cardinality. For example, instead of searching for subspace clusters over subsets of the genes (which can be large), we can search over subsets of samples (which are typically very few) or over subsets of time-courses (which are also not large).

Note that by definition, a cluster represents a scaling cluster (if the ratio is 1.0, then it is a uniform cluster). However, our our definition allows for the mining of shifting clusters as well, as indicated by the lemma below.

Lemma 4 *(Shifting Cluster for 3D Case - an extension of Lemma 2)*
Let $C = X \times Y \times Z = \{c_{xyz}\}$ be a maximal tricluster. Let $e^C = \{e^{c_{xyz}}\}$ be the tricluster obtained by applying the exponential function (base e) to each value in C. If e^C is a (scaling) cluster, then C is a shifting cluster.

Proof: Let $C_{2,2} = \begin{bmatrix} c_{ia} & c_{ib} \\ c_{ja} & c_{jb} \end{bmatrix}$ an arbitrary 2×2 submatrix of C. Assume that e^C is a valid scaling cluster. Then by definition, $e^{c_{ib}} = \alpha_i e^{c_{ia}}$. But this immediately implies that $\ln(e^{c_{ib}}) = \ln(\alpha_i e^{c_{ia}})$, which gives us $c_{ib} = \ln(\alpha_i) + c_{ia}$. Likewise, we have $c_{jb} = \ln(\alpha_j) + c_{ja}$. Setting $\beta_i = \ln(\alpha_i)$ and $\beta_j = \ln(\alpha_j)$, we have that C is a shifting cluster. ■

Note that the clusters can have arbitrary positions anywhere in the data matrix, and they can have arbitrary overlapping regions (though, TRICLUSTER can optionally merge or delete overlapping clusters under certain scenarios). We impose the minimum size constraints i.e. mx, my and mz to mine large enough clusters. Typically $\varepsilon \approx 0$, so that the ratios of the values along one dimension in the cluster are similar (by symmetry Lemma 3, this property is also applicable for the other two dimensions), i.e., the ratios can differ by at most ε. Further, different choices of dimensional range thresholds (δ^x, δ^y and δ^z) produce different kinds of clusters:

1. If $\delta^x = \delta^y = \delta^z = 0$, then we obtain a cluster that has identical values along all dimensions.

2. If $\delta^x = \delta^y = \delta^z \approx 0$, then we obtain clusters with approximately identical values.

3. If $\delta^x \approx 0$, $\delta^y \neq 0$ and $\delta^z \neq 0$, then we obtain a cluster ($X \times Y \times Z$), where each gene $g_i \in X$ has similar expression values across the different samples Y and the

different times Z, and different genes' expression values cannot differ by more than the threshold δ^x. Similarly we can obtain other cases by setting i) $\delta^x \neq 0$, $\delta^y \approx 0$ and $\delta^z \neq 0$ or ii) $\delta^x \neq 0$, $\delta^y \neq 0$ and $\delta^z \approx 0$.

4. $\delta^x \approx 0$, $\delta^y \approx 0$ and $\delta^z \neq 0$, we obtain a cluster with similar values for genes and samples, but the time-courses are allowed to differ by some arbitrary scaling factor. Similar cases are obtained by setting i) $\delta^x \approx 0$, $\delta^y \neq 0$ and $\delta^z \approx 0$, and ii) $\delta^x \neq 0$, $\delta^y \approx 0$ and $\delta^z \approx 0$.

5. If $\delta^x \neq 0$, $\delta^y \neq 0$ and $\delta^z \neq 0$, then we obtain a cluster that exhibits scaling behavior on genes, samples and times, and the expression values are bounded by δ^x, δ^y and δ^z respectively.

Note also that TRICLUSTER also allows different ε values for different pairs of dimensions. For example, we may use one value of ε to constrain the expression values for, say, the gene-sample slice, but we may then relax the maximum ratio threshold for the temporal dimension to capture more interesting (and big) changes in expression as time progresses.

For example, Table 4.1(b) shows some examples of different clusters that can be obtained by permuting some dimensions. Let $mx = my = 3$, $mz = 2$ and let $\varepsilon = 0.01$. If we let $\delta^x = \delta^y = \delta^z = \infty$, i.e., if they are unconstrained, then $C_1 = \{g_1, g_4, g_8\} \times \{s_1, s_4, s_6\} \times \{t_0, t_1\}$ is an example of a scaling cluster, i.e., each point values along one dimension is some scalar multiple of another point values along the same dimension. We also discover two other maximal overlapping clusters, $C_2 = \{g_0, g_2, g_6, g_9\} \times \{s_1, s_4, s_6\} \times \{t_0, t_1\}$, $C_3 = \{g_0, g_7, g_9\} \times \{s_1, s_2, s_4, s_5\} \times \{t_0, t_1\}$. Note that if we set $my = 2$ we would find another maximal cluster $C_4 = \{g_0, g_2, g_6, g_7, g_9\} \times \{s_1, s_4\} \times \{t_0, t_1\}$, which is subsumed by C_2 and C_3. We shall see later that TRICLUSTER can optionally delete such a cluster in the final steps. If we set $\delta^x = 0$, and let δ^y and δ^z be unconstrained, then we will not find cluster C_1, whereas all other clusters will remain valid. This is because if $\delta^x = 0$, then the values for each gene in the cluster must be identical, however since δ^y and δ^z are unconstrained the cluster can have different coherent values along the samples and times. Since ε is symmetric for each dimension, TRICLUSTER first discovers all unconstrained clusters rapidly, and then prunes unwanted clusters

if δ^x, δ^y or δ^z are constrained. Finally, it optionally deletes or merges mined clusters if certain overlapping criteria are met.

4.2 The TRICLUSTER Algorithm

TRICLUSTER can mine arbitrarily positioned and overlapping scaling and shifting patterns from a three dimensional dataset, as well as several specializations. Typically 3D microarray datasets have more genes than samples, and perhaps an equal number of time points and samples, i.e. $|G| \geq |T| \approx |S|$. Due to the symmetric property, TRICLUSTER always transposes the input 3D matrix such that the dimension with the largest cardinality (say G) is the first dimension; we then make S as the second and T as the third dimension. TRICLUSTER has the following main steps:

1. For each $G \times S$ time slice matrix, get the maximal biclusters corresponding to each time slice using MICROCLUSTER.
2. Construct a graph based on the mined biclusters of each time slice (as vertices) and get the maximal triclusters.
3. Optionally, delete or merge clusters if certain overlapping criteria are met.

The first step has been stated clearly in Chapter 3. Next, we look at the other two steps below.

4.2.1 Get Triclusters from Bicluster Graph

After having got the maximal bicluster set $\mathcal{C}^t$ for each time slice t, we use them to mine the maximal triclusters. This is accomplished by enumerating the subsets of the time slices as shown in Figure 4.1, using a process similar to the MICROCLUSTER clique mining (Figure 3.2.2). For example, from Table 4.1, we can get the biclusters from the two time points t_0 and t_1 as shown in Figure 4.2. Since the clusters are identical, to adequately illustrate our tricluster mining method, let's assume that we also obtain other biclusters at times points t_3 and t_8. Assume the minimum size threshold is $mx \times my \times mz = 3 \times 3 \times 3$. TRICLUSTER starts from time t_0, which contains three biclusters. Let's begin with cluster C_1 at time t_0, given as $C_1^{t_0}$. For each bicluster $\mathcal{C}^{t_1}$, only $C_1^{t_1}$ can be used for extension

Input : parameters: $\varepsilon, mx, my, mz, \delta^x, \delta^y, \delta^z$, bicluster sets $\{\mathcal{C}^t\}$ of all times, set of genes G, samples S and times T
Output : cluster set $\mathcal{C}$
Initialization: $\mathcal{C} = \emptyset$, call TRICLUSTER($C = G \times S \times \emptyset, T$)

```
1  TRICLUSTER (C = X×Y×Z, P);
2  if C satisfies δ^x, δ^y, δ^z then
3      if |C.Z| ≥ mz then
4          if ∄C' ∈ C, such that C ⊂ C' then
5              Delete any C'' ∈ C, if C'' ⊂ C
6              C ⟵ C + C
7  foreach t_b ∈ P do
8      C^new.Z ⟵ C.Z + t_b
9      P ⟵ P − t_b
10     forall t_a ∈ C.Z and each bicluster c_i^{t_a} ∈ C^{t_a} satisfying
       |c_i^{t_a}.X ∩ C.X| ≥ mx and |c_i^{t_a}.Y ∩ C.Y| ≥ my do
11         C^new.X ⟵ (⋂_{all t_a∈C.Z} c_i^{t_a}.X) ∩ C.X
12         C^new.Y ⟵ (⋂_{all t_a∈C.Z} c_i^{t_a}.Y) ∩ C.Y
13         if |C^new.X| ≥ mx and |C^new.Y| ≥ my and the ratios at
           time t_b, t_a are coherent then
14             TRICLUSTER(C^new, P)
```

Figure 4.1: TRICLUSTER Algorithm

since $C_1^{t_0} \cap C_1^{t_1} = \{g_1, g_4, g_8\} \times \{s_0, s_1, s_4, s_6\}$, which can satisfy the cardinality constraints (Figure 4.1, line 15). We continue by processing time t_3, but the cluster cannot be extended. So we try t_8, and we may find that we can extend it via $C_1^{t_8}$. The final result of this path is $\{g_1, g_4, g_8\} \times \{s_0, s_1, s_4, \} \times \{t_0, t_1, t_8\}$. Similarly we try all such paths and keep maximal triclusters only. During this process we also need to check the coherent property along the time dimension, as the tricluster definition requires, between the new time slice and the previous one. For example, for the three biclusters in Table 4.1, the ratios between t_1 and t_0 are 1.2 (for C_1) and 0.5 (for C_2 and C_3) respectively. If the

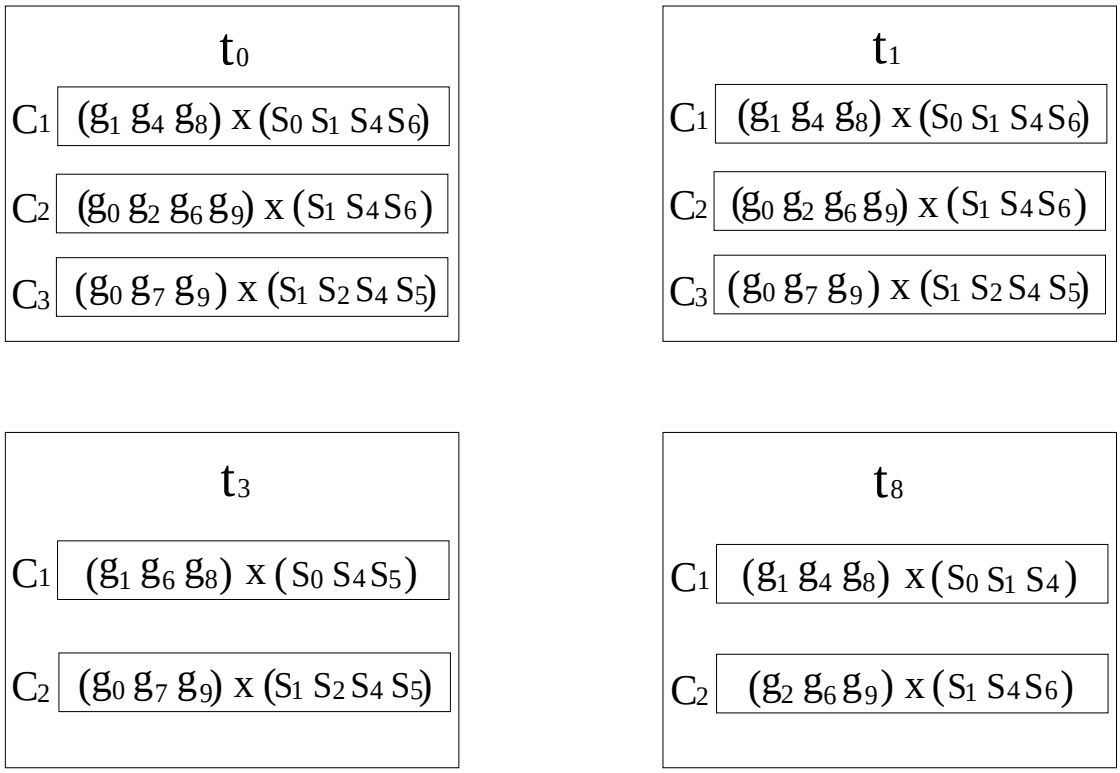


Figure 4.2: Tricluster Example

extended bicluster has no such coherent values in the intersection region, TRICLUSTER will prune it.

The complexity of this part (along the time dimension) is the same as that of biclusters generation (MICROCLUSTER) for one time slice. But since the MICROCLUSTER need to run $|T|$ times, the total running time is: $|T| \times (time(multigraph) + time(Micro\ Cluster)) + time(triCluster)$.

4.2.2 Merge and Prune Clusters

After mining the set of all clusters, TRICLUSTER optionally merges or deletes certain clusters with large overlap to deal with the dataset noise and reduce the information redundancy conveyed to the users. The merging and pruning are extensions of the merging and pruning processes of MICROCLUSTER from 2D datasets to 3D datasets. Now lets extend the metrics from 2D to 3D first. For clarity, we still use two dimensional figures here to show the triclusters relationship as Figure 3.4.

Let $A = X_A \times Y_A \times Z_A$, and $B = X_B \times Y_B \times Z_B$ be any two mined triclusters. We

define the *span* of a tricluster $C = X \times Y \times Z$, to be the set of gene-sample-time tuples that belong to the tricluster, given as $L_C = \{(g_i, s_j, t_k) | g_i \in X, s_j \in Y, t_k \in Z\}$. Then we can define the following derived spans:

- $L_{A \cup B} = L_A \cup L_B$,
- $L_{A-B} = L_A - L_B$, and
- $L_{A+B} = L_{(X_A \cup X_B) \times (Y_A \cup Y_B) \times (Z_A \cup Z_B)}$

If any of the following three overlap conditions are met, TRICLUSTER either deletes or merges the clusters involved:

1. For any two clusters A and B, if $L_A > L_B$, and if $\frac{|L_{B-A}|}{|L_B|} < \eta$, then delete B. As illustrated in Figure 3.4(a), this means that if the cluster with the smaller span (B) has only a few extra elements, then delete the smaller cluster.

2. This is a generalization of case 1). For a cluster A, if there exists a set of clusters $\{B_i\}$, such that $\frac{|L_A - L_{\cup_i B_i}|}{|L_A|} < \eta$, then delete cluster A. As shown in Figure 3.4(b), A is mostly covered by the $\{B_i\}$'s. Therefore it can be deleted.

3. For two clusters A and B, if $\frac{|L_{(A+B)-A-B}|}{|L_{A+B}|} < \gamma$, merge A and B into one cluster $(X_A \cup X_B) \times (Y_A \cup Y_B) \times (Z_A \cup Z_B)$. This case is shown in Figure 3.4(c). Here η, γ are user-defined thresholds.

4.2.3 Complexity Analysis

Since we have to evaluate all pairs of samples, compute their ratios, and find the valid ranges over all the genes, the range multigraph construction takes time $O(|G||S|^2|T|)$. The bicluster mining step and tricluster mining step correspond to constrained maximal clique enumeration (i.e., cliques satisfying the $mx, my, mz, \delta^x, \delta^y, \delta^z$ parameters) from the range multigraph and bicluster graph. Since in the worst case there can be an exponential number of clusters, these two steps are the most expensive. The precise number of clusters mined depends on the dataset and the input parameters. Nevertheless, for microarray datasets TRICLUSTER is likely to be very efficient due to the similar reasons as MICROCLUSTER.

4.3 Experiments

All experiments were done on a Linux/Fedora virtual machine (Pentium-M, 1.4GHz, 448M memory) over Windows XP through middleware VMware unless otherwise noted. We used both synthetic and real microarray datasets to evaluate TRICLUSTER algorithm. For the real dataset we used the yeast cell-cycle regulated genes [40] (`http://genome-www.stanford.edu/cellcycle`). The goal is the study was to identify all genes whose mRNA levels are regulated by the cell cycle.

Synthetic datasets allow us to embed clusters, and then to test how TRICLUSTER performs for varying input parameters. We generate synthetic data using the following steps: The input parameters to the generator are the total number of genes, samples and times; number of clusters to embed; percentage of overlapping clusters; dimensional ranges for the cluster sizes; and the amount of noise for the expression values. The program randomly picks cluster positions in the data matrix, ensuring that no more than the required number of clusters overlap. The cluster sizes are generated uniformly between each dimensional ranges. For generating the expression values within a cluster, we generate at random, base values (v_i, v_j and v_k) for each dimension in the cluster. Then the expression value is set as $d_{ijk} = v_i \cdot v_j \cdot v_k \cdot (1 + \rho)$, where ρ doesn't exceed the random noise level. Once all clusters are generated, the non-cluster regions are assigned random values.

4.3.1 Results on Synthetic Datasets

TRICLUSTER's behaviors with varying input parameters in isolation are shown first. The synthetic data are generated with the following default parameters: data matrix size $4000 \times 30 \times 20 (G \times S \times T)$, number of clusters 10, cluster size $150 \times 6 \times 4 (X \times Y \times Z)$, percentage overlap 20%, noise level 3%. For each experiment, we keep all default parameters, except for the varying parameter. We also choose appropriate parameter values for TRICLUSTER so that all embedded clusters were found. Figure4.3(a)-(f) shows TRICLUSTER's sensitivity to different parameters. We found that the time increases approximately linearly with the number of genes in a cluster (a). This is because, the range multi-graph is constructed on the samples, and not on the genes; more genes lead to longer gene-sets (per edge), but the intersection time is essentially linear in gene-set size. The time is ex-

ponential with the number of samples (b), since we search over the sample subset space. Finally, the time for increasing time-slices is also linear for the range shown (c), but in general the dependence will be exponential, since TRICLUSTER searches over subsets of time points after mining the biclusters for each time-slice. The time is linear w.r.t. number of clusters (d), whereas the overlap percentage doesn't seem to have much impact on the time (e). Finally, as we add more noise, the more the time to mine the clusters (f), since there is more chance that a random gene or sample can belong to some cluster.

4.3.2 Results from Real Microarray Datasets

For the yeast cell cycle data we looked at the time slices for the Elutritration experiments. There are a total of 7679 genes whose expression value is measured from time 0 to 390 minutes at 30 minute intervals. Thus there are 14 total time points. Finally, we use 13 of the attributes of the raw data as the samples (e.g., the raw values for the average and normalized signal for Cy5 & Cy3 dyes, the ratio of those values, etc.). Thus we obtain a 3D expression matrix of size: $T \times S \times G = 14 \times 13 \times 7679$. We mined this data looking for triclusters with minimum size at least $mx = 50$ (genes), $my = 4$ (samples), $mz = 5$ (time points), and we set $\varepsilon = 0.003$ (however, we relax the ε threshold along the time dimension). The per dimension thresholds $\delta^x, \delta^y, \delta^z$ were left unconstrained. TRICLUSTER output 5 clusters in 17.8s, with the following metrics (defined in Chapter 1) as Table 4.2 shows.

Clusters#	5
Elements#	6520
Coverage	6520
Overlap	0.00%
Fluctuation	T:626.53, S:163.05, G:407.3

Table 4.2: An Output Metrics Example

We can see that none of the 5 clusters was overlapping. The values for the total span across the clusters was 6520 cells, and the variances along each dimension are also shown. To visually see a mined tricluster, we plot various 2D views of one of the clusters (C_0) in Figure 4.4, Figure 4.5, and Figure 4.6. Figure 4.4 shows how the expression values for the genes (X axis) changes across the samples (Y axis), for different time points (the

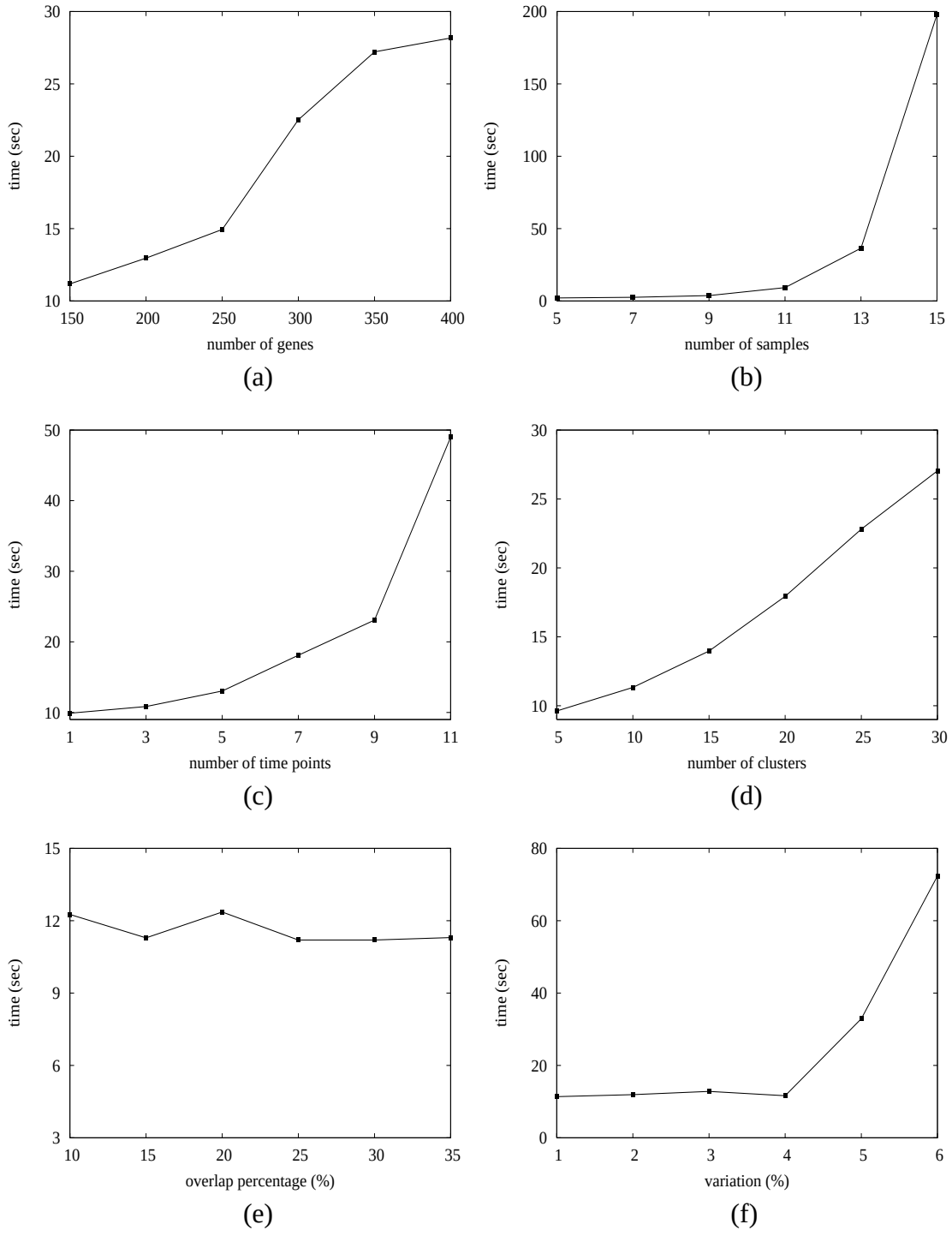


Figure 4.3: Evaluation of TRICLUSTER on Synthetic Datasets

different sun-plots). Figure 4.5 shows how the gene expression (X axis) changes across the different time slices (Y axis), for different samples (the different sub-plots). Finally Figure 4.5 shows what happens at different times (X axis) for different genes (Y axis), across different samples (the different sub-plots). These figures show that TRICLUSTER is able to mine coherent clusters across any combination of the gene-sample-time dimensions.

The gene ontology (GO) project (www.geneontology.org) aims at developing three structured, controlled vocabularies (ontologies) that describe gene products in terms of their associated biological processes, cellular components and molecular functions in a species-independent manner. We used the yeast genome gene ontology term finder (www.yeastgenome.org) to verify the biological significance of TRICLUSTER's result. We obtained a hierarchy of GO terms for each gene within each cluster, for each of the three categories: processes, cellular components and gene functions. Table 4.3 shows the significant shared GO terms (or parents of GO terms) used to describe the set of genes in each cluster. The table shows the number of genes in each cluster and the significant GO terms for the process, function and component ontologies. Only the most significant common terms are shown. For example for cluster C_0, we find that the genes are mainly involved in ubiquitin cycle. The tuple $(n = 3, p = 0.00346)$ means that out of the 51 genes, 3 belong to this process, and the statistical significance is given by the p-value of 0.00346. Within each category, the terms are given in descending order of significance (i.e., increasing p-values). Further, only p-values lower than 0.01 are shown; the other genes in the cluster share other terms, but at a lower significance. From the table it is clear that the clusters are distinct along each category. For example, the most significant process for C_0 is ubiquitin cycle, for C_1 it is G1/S transition of mitotic cell cycle, for C_2 it is lipid transport, for C_3 it is physiological process/organelle organization and biogenesis, and for C_4 it is pantothenate biosynthesis. Looking at function we find the most significant terms to be protein phosphatase regulator activity for C_1, oxidoreductase activity for C_2, MAP kinase activity for C_3, and ubiquitin conjugating enzyme activity for C_4. Finally, the clusters also differ in terms of cellular component: C_2 genes belong to cytoplasm, C_3 genes to membrane, and C_4 to Golgi vesicle.

These results indicate that TRICLUSTER can find potentially biologically signif-

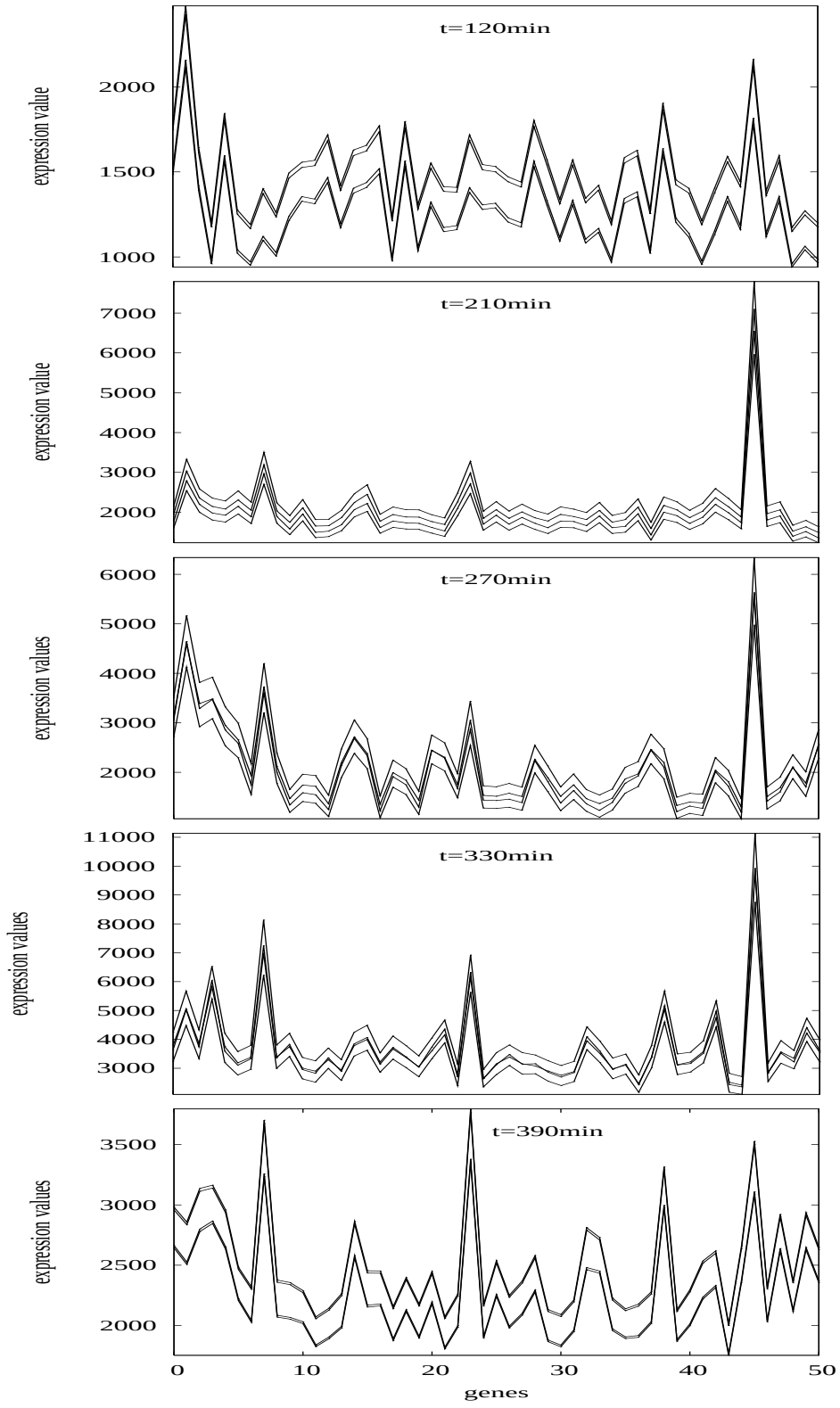


Figure 4.4: Sample Curves

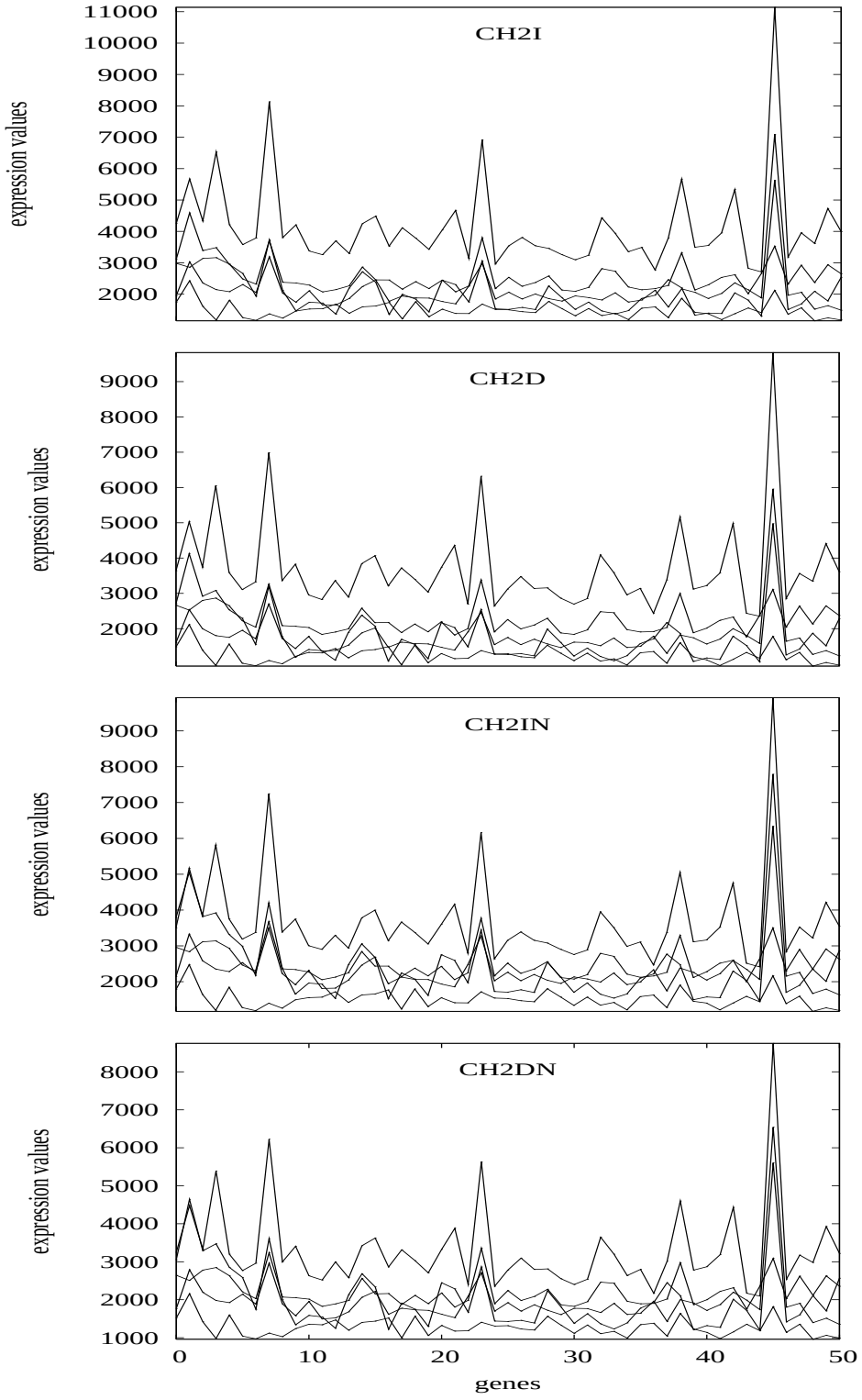


Figure 4.5: Time Curves

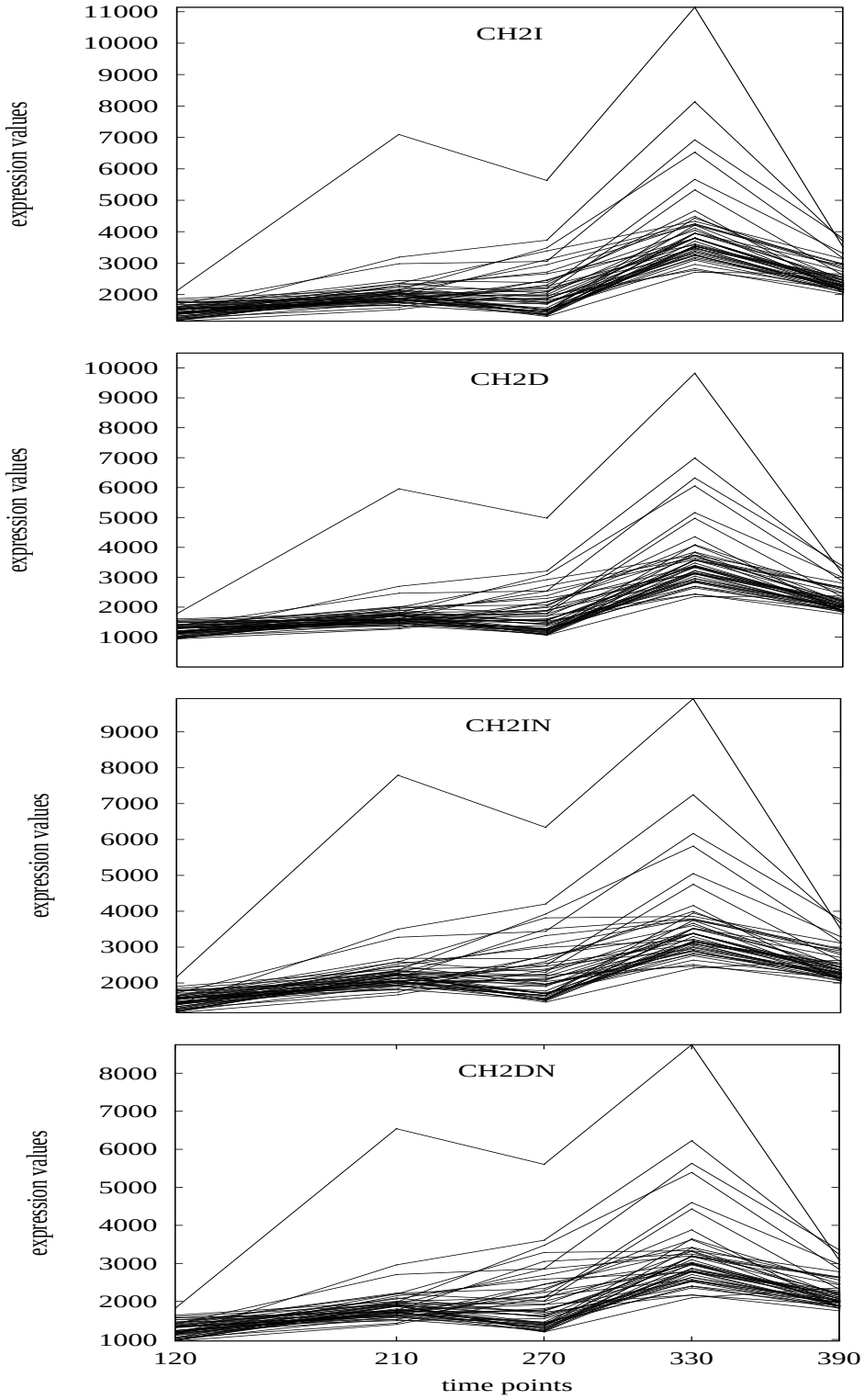


Figure 4.6: Gene Curves

Cluster	#Genes	Process	Function	Cellular Component
C_0	51	ubiquitin cycle (n=3, p=0.00346), protein polyubiquitination (n=2, p=0.00796), carbohydrate biosynthesis (n=3, p=0.00946)		
C_1	52	G1/S transition of mitotic cell cycle (n=3, p=0.00468), mRNA polyadenylylation (n=2, p=0.00826)	protein phosphatase regulator activity (n=2,p=0.00397) , phosphatase regulator activity (n=2, p=0.00397)	
C_2	57	lipid transport (n=2, p=0.0089)	oxidoreductase activity (n=7, p=0.00239), lipid transporter activity (n=2, p=0.00627), antioxidant activity (n=2, p=0.00797)	cytoplasm (n=41, p=0.00052), microsome (n=2, p=0.00627), vesicular fraction (n=2, 0.00627), microbody (n=3, p=0.00929), peroxisome (n=3, p=0.00929)
C_3	97	physiological process (n=76, p=0.0017), organelle organization and biogenesis (n=15, p=0.00173), localization (n=21, p=0.00537)	MAP kinase activity (n=2, p=0.00209), deaminase activity (n=2, p=0.00804), hydrolase activity, acting on carbon-nitrogen, but not peptide, bonds (n=4, p=0.00918), receptor signaling protein serine/threonine kinase activity (n=2, p=0.00964)	membrane (n=29, p=9.36e-06), cell (n=86, p=0.0003), endoplasmic reticulum (n=13, p=0.00112), vacuolar membrane (n=6, p=0.0015), cytoplasm (n=63, p=0.00169) intracellular (n=79, p=0.00209), endoplasmic reticulum membrane (n=6, p=0.00289), integral to endoplasmic reticulum membrane (n=3, p=0.00328), nuclear envelope-endoplasmic reticulum network (n=6, p=0.00488)
C_4	66	pantothenate biosynthesis (n=2, p=0.00246), pantothenate metabolism (n=2, p=0.00245), transport (n=16, p=0.00332), localization (n=16, p=0.00453)	ubiquitin conjugating enzyme activity (n=2, p=0.00833), lipid transporter activity (n=2, p=0.00833)	Golgi vesicle (n=2, p=0.00729)

Table 4.3: Significant Shared GO Terms (Process, Function, Component) for Genes in Different Clusters

icant clusters either in genes or samples or times, or any combinations of these three dimensions. Since the method can mine coherent subspace clusters in any 3D dataset, TRICLUSTER will also prove to be useful in mining temporal and/or spatial dimensions. For example, if one of the dimensions represents genes, another the spatial region of interest, and the third dimension the time, then TRICLUSTER can find interesting expression patterns in different regions at different times.

4.4 Conclusions

In this chapter, an original clustering problem $triclustering$ and a corresponding deterministic triclustering algorithm (based on MICROCLUSTER) called TRICLUSTER are introduced, which can mine arbitrarily positioned and overlapping clusters. Depending on different parameter values, TRICLUSTER can mine different types of clusters, including those with constant or similar values along each dimension, as well as scaling and shifting expression patterns. TRICLUSTER first uses MICROCLUSTER to yield the set of biclusters for each time slice of the input 3D dataset. Then TRICLUSTER constructs another multi-graph using the highly overlapping biclusters (as edges) corresponding to each time slice (as vertices). The clique mining of the multi-graph will give the final set of triclusters. Optionally, TRICLUSTER merges/deletes some clusters having large overlaps. A useful set of metrics similar to that of MICROCLUSTER is presented to evaluate the clustering quality. In addition, we also evaluate the sensitivity of TRICLUSTER to different parameters. The experimental results show that TRICLUSTER can find meaningful clusters in real data.

CHAPTER 5

BLOSOM: Mining Arbitrary Boolean Patterns from Binary-valued Datasets

The content of this chapter was published in part in paper [55].

5.1 Definitions and Problem Statement

Dataset Given a set of items $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$, let $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ be a set of transaction identifiers (tids). A dataset $\mathcal{D}$ is then a subset of $\mathcal{T} \times 2^{\mathcal{I}}$ (note that $2^{\mathcal{I}}$ denotes the power-set of $\mathcal{I}$, i.e., the set of all subsets of $\mathcal{I}$); in other words, the dataset $\mathcal{D}$ is a set of tuples of the form $(t, t.X)$, where $t \in \mathcal{T}$ is the tid of the transaction containing the set of items $t.X \subseteq \mathcal{I}$. Note that any categorical dataset can be transformed into this transactional form, by assigning a unique item for each attribute-value pair.

Given dataset $\mathcal{D}$, we denote by $\mathcal{D}^T$ the transposed dataset that consists of a set of tuples of the form $(i, i.Y)$, where $i \in \mathcal{I}$ and $i.Y$ is the set of tids of transactions containing i. Figure 5.1 shows a dataset and its transpose, which we will use as a running example in this chapter. It has five items $I = \{A, B, C, D, E\}$ and five tids $\mathcal{T} = \{1, 2, 3, 4, 5\}$. Note that in $\mathcal{D}$, transaction t_1 contains the set of items $\{A, C, D\}$ (for convenience, we write it as ACD), and in $\mathcal{D}^T$, the set of tids of transactions that contain item A is $\{1, 3, 4\}$ (again, for convenience we write it as 134).

$\mathcal{D}$		$\mathcal{D}^T$	
tid	set of items	item	tidset
1	ACD	A	134
2	BC	B	23
3	ABCD	C	123
4	ADE	D	134
5	E	E	45

Figure 5.1: Dataset $\mathcal{D}$ and its Transpose $\mathcal{D}^T$

Boolean Expressions Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of binary-valued attributes or *items*. Let $\wedge$, $\vee$ denote the binary operators standing for the logical AND and OR, respv., and let $\neg$ denote the unary operator for logical negation. A *literal* is either an item i or its negation $\neg i$. A *clause* is either the logical AND or logical OR of one or more literals. An AND-clause is a clause that has only the $\wedge$ operator over all its literals, and an OR-clause is one that has only the $\vee$ operator over all its literals, e.g., $i_2 \wedge i_4$ is an AND-clause, while $i_3 \vee i_5$ is an OR-clause. We assume without loss of generality that a clause has all distinct literals (since a clause is either an AND- or an OR-clause, repeated literals are logically redundant). A *boolean expression* is the logical AND or OR of one or more clauses. Parentheses are used to demarcate operator grouping and precedence, when necessary. A boolean expression is said to be in *conjunctive normal form* (CNF) if it is an AND of OR-clauses. An expression is said to be in *disjunctive normal form* (DNF) if it is an OR of AND-clauses. For example, $(i_3 \wedge i_4) \vee (i_1 \wedge i_5 \wedge i_7)$ is in DNF, whereas $(i_2 \vee i_3) \wedge (i_0 \vee i_1 \vee (\neg i_3))$ is in CNF. For notational convenience we denote a negated item $\neg i$ as $\bar{i}$, we use the symbol $|$ instead of $\vee$ to denote OR, and simply omit the operator $\wedge$ whenever there is no ambiguity; thus two adjacent clauses without any operator will always denote an AND. For example, $(i_2 \vee i_3) \wedge (i_0 \vee i_1 \vee (\neg i_3))$ is rewritten as $(i_2|i_3)(i_0|i_1|\bar{i_3})$, and $(i_3 \wedge i_4) \vee (i_1 \wedge i_5 \wedge i_7)$ is rewritten as $(i_3 i_4)|(i_1 i_5 i_7)$. The length $|e|$ of an expression e is the number of literals it has.

Tidset and Support Given a transaction $(t, t.X) \in \mathcal{D}$, with $t \in \mathcal{T}$ and $t.X \subseteq \mathcal{I}$, we say that tid t *satisfies* an item/literal $i \in \mathcal{I}$ if $i \in t.X$, and t satisfies the literal $\bar{i}$ if $i \notin t.X$. For a literal l, the *truth value* of l in transaction t, denoted $V_t(l)$ is given as follows:

$$V_t(l) = \begin{cases} 1 & \text{if } t \text{ satisfies } l \\ 0 & \text{if } t \text{ does not satisfy } l \end{cases}$$

We say that a transaction $t \in \mathcal{T}$ *satisfies* a boolean expression E if the truth-value of E, denoted $V_t(E)$, evaluates to true when we replace every literal l in E with $V_t(l)$. For any boolean expression E, $\mathbf{t}(E) = \{t \in \mathcal{T} : V_t(E) = 1\}$ denotes the set of tids (also called a *tidset*) that satisfy E. For example, for the dataset of Figure 5.1, $\mathbf{t}((A \cap E) \cup (B \cap D)) = \{3, 4\}$ because $V_3((A \cap E) \cup (B \cap D)) = 1$ and $V_4((A \cap E) \cup (B \cap D)) = 1$; since

$V_2((A \cap E) \cup (B \cap D)) = 0$, so $2 \notin \mathbf{t}((A \cap E) \cup (B \cap D))$.

The *support* of a boolean expression E in dataset $\mathcal{D}$ is the number of transactions which satisfy E, i.e., $|\mathbf{t}(E)|$. An expression is *frequent* if its support is more than or equal to a user-specified *minimum support* (min_sup) value, i.e., if $|\mathbf{t}(E)| \geq min\_sup$. For disjunctive expressions, we also impose a maximum support threshold max_sup to disallow any expression with too high a support. Setting $min\_sup = 1$ and $max\_sup = \infty$ allows mining all possible expressions.

Boolean Expression Mining Tasks Given a dataset $\mathcal{D}$ and support thresholds min_sup and/or max_sup, we are interested in mining various types of frequent boolean expressions over the item space, such as AND-clauses (i.e., pure conjunctions), OR-clauses (i.e., pure disjunctions), CNF and DNF. Moreover, instead of mining all such frequent boolean expressions, we focus our attention on mining only (frequent) closed boolean expressions and their minimal generators, as described next.

5.2 Closed Boolean Expressions

In this section we describe the concept of closed boolean expressions, and show that this set is necessary and sufficient to capture all the information about boolean expressions, and has smaller cardinality than the set of all boolean expressions.

Let the set $\mathcal{E}$ denote the set of all possible boolean expressions over the set of items $\mathcal{I}$. Let $(\mathcal{E}, \theta)$ be a partially ordered set with the binary relation θ (e.g., $\subseteq, \supseteq$, etc.), and let $S \subseteq \mathcal{E}$ be a set of boolean expressions. An element $U \in \mathcal{E}$ ($L \in \mathcal{E}$) is an *upper bound* (*lower bound*) of S if $X\ \theta\ U$ ($L\ \theta\ X$) for all $X \in S$. The least upper bound is called the *join* of S, and is denoted as $\bigvee S$, and the greatest lower bound is called the *meet* of S, and is denoted as $\bigwedge S$. If $S = \{X, Y\}$, we also write $X \vee Y$ for the join, and $X \wedge Y$ for the meet. A partially ordered set (L, θ) is a *lattice* if for any two elements X and Y in L, the join $X \vee Y$ and meet $X \wedge Y$ always exist. L is a *complete lattice* if $\bigvee S$ and $\bigwedge S$ exist for all $S \subseteq L$. Any finite lattice is complete [19]. Given a partially ordered set $(\mathcal{E}, \theta)$ and some property P, we say that $X \in \mathcal{E}$ is *maximal* w.r.t. P if X satisfies P, and there does not exist $X' \in \mathcal{E}$, such that $X\ \theta\ X'$ and X' also satisfies P, unless $X = X'$. We say that $X \in \mathcal{E}$ is *minimal* w.r.t. P if X satisfies P, and there does not exist $X' \in \mathcal{E}$, such that

$X' \; \theta \; X$ and X' also satisfies P, unless $X = X'$.

Given the partially ordered set $(\mathcal{E}, \theta)$ with binary operator θ, a *closure* operator $\mathbb{C}$ on $\mathcal{E}$ satisfies the following conditions for all $X, Y \in \mathcal{E}$.

1. *idempotent*: $\mathbb{C}(\mathbb{C}(X)) = \mathbb{C}(X)$
2. *extensive*: $X \; \theta \; \mathbb{C}(X)$
3. *monotone*: if $X \; \theta \; Y$, then $\mathbb{C}(X) \; \theta \; \mathbb{C}(Y)$

An expression $X \in \mathcal{E}$ is called $\mathbb{C}$-*closed* (or simply closed, if $\mathbb{C}$ is obvious) if $\mathbb{C}(X) = X$. In other words, the expression X is a fixed point of the closure operator $\mathbb{C}$. Equivalently, X is closed if there exists no $Y \; \theta \; X$ with $\mathbf{t}(X) = \mathbf{t}(Y)$. For instance, in the dataset from Figure 5.1, let $\mathcal{E}^{\wedge}$ be the set of all AND-clauses over $\mathcal{I}$, then $(\mathcal{E}^{\wedge}, \supseteq)$ is a partially ordered set with binary relation $\supseteq$. Assume $X = \{B\}$, then $\mathbb{C}(X) = \{B, C\}$ since $\mathbf{t}(\{B\}) = \mathbf{t}(\{B, C\}) = \{2, 3\}$.

Let X be a $\mathbb{C}$-closed expression. We say that a boolean expression $Y \; \theta \; X$ is a $\mathbb{C}$-*generator* (or simply generator, if $\mathbb{C}$ is obvious) of X if $\mathbb{C}(Y) = X$. Equivalently, Y is a generator of X if $\mathbf{t}(Y) = \mathbf{t}(X)$. Y is called a *proper* generator if $Y \neq X$. By definition, a proper generator cannot be closed. Let $\mathcal{G}(X)$ be the set of all generators of X, including X. Then the minimal elements of $\mathcal{G}(X)$ are called the *minimal generators* of X. We use the notation $\mathcal{G}^{\min}(X)$ to denote the set of all the minimal generators of a closed expression X. The unique maximal element of $\mathcal{G}(X)$ is X. Since $\mathbf{t}(Y) = \mathbf{t}(X) = T$ for all $Y \in \mathcal{G}(X)$, we observe that a $\mathbb{C}$-closed expression X *maximally describes* its corresponding tidset $T = \mathbf{t}(X)$, i.e., X represents the maximal expression, w.r.t. the binary operator θ, satisfied by the maximal set of tids T. On the other hand, a minimal generator Y of X *minimally describes* the same tidset T, since there does not exist $Y' \; \theta \; Y$, such that Y' describes the same tidset T (unless $Y' = Y$). Put another way, the closed expressions are the most specific expressions that describe the tidset T, whereas the minimal generators are the most general expressions that describe T. We use the notation $\mathbb{C}(\mathcal{E})$ to denote the set of all $\mathbb{C}$-closed expressions in $\mathcal{E}$, and the notation $\mathcal{G}^{\min}{}_{\mathbb{C}}(\mathcal{E})$ to denote the set of all $\mathbb{C}$-generators of closed expressions in $\mathcal{E}$, i.e., $\mathcal{G}^{\min}{}_{\mathbb{C}}(\mathcal{E}) = \bigcup_{X \in \mathbb{C}(\mathcal{E})} \mathcal{G}^{\min}(X)$.

Given the partially ordered set $(\mathcal{E}, \theta)$ of all boolean expressions, and the partially ordered set $(2^{\mathcal{T}}, \subseteq)$ of tidsets, let $\rho : \mathcal{E} \mapsto 2^{\mathcal{T}}$ and $\eta : 2^{\mathcal{T}} \mapsto \mathcal{E}$ be two mapping be-

tween them, then these mappings form a *Galois connection* if they satisfy the following properties [19] for any $X, X_1, X_2 \in \mathcal{E}$ and $Y, Y_1, Y_2 \in 2^{\mathcal{T}}$:

- $X_1 \; \theta \; X_2 \Rightarrow \rho(X_2) \subseteq \rho(X_1)$
- $Y_1 \subseteq Y_2 \Rightarrow \eta(Y_2) \; \theta \; \eta(Y_1)$
- $X \; \theta \; \eta(\rho(X))$ and $Y \subseteq \rho(\eta(Y))$

It is known that the composition of these two mapping produces a closure operator [19], i.e., $\mathbb{C}(X) = (\eta \circ \rho)(X) = \eta(\rho(X))$ is a closure operator on $\mathcal{E}$.

5.2.1 Pure Conjunctions: AND-clauses

We first consider AND-clauses or pure conjunctive boolean expressions. These have been well studied in the data mining community in the form of itemset mining [4], since an itemset $X \subseteq \mathcal{I}$ is simply an AND-clause over $\mathcal{I}$. The closure operator for AND-clauses was studied in [19], and the notion of minimal generators was introduced in [7]. Many algorithms for mining closed sets (e.g., Charm [51] and others [20]), and a few to mine minimal generators [7, 52], have been proposed.

We briefly review these previous results, especially w.r.t. the closure operator. Let $\mathcal{E}^\wedge$ be the set of all AND-clauses over $\mathcal{I}$, let $(\mathcal{E}^\wedge, \subseteq)$ be the partial order over $\mathcal{E}^\wedge$, and let $(2^{\mathcal{T}}, \subseteq)$ be the partial order over $2^{\mathcal{T}}$. Let $X \in \mathcal{E}^\wedge$ be some AND-clause, then the following two mappings form a Galois connection over $\mathcal{E}^\wedge = 2^{\mathcal{I}}$ and $2^{\mathcal{T}}$ [19]:

- $\rho = \mathbf{t} : \mathcal{E}^\wedge \mapsto 2^{\mathcal{T}}, \quad \mathbf{t}(X) = \{t \in \mathcal{T} \mid \;\; t \text{ satisfies } X\}$
- $\eta = \mathbf{i}^\wedge : 2^{\mathcal{T}} \mapsto \mathcal{E}^\wedge, \quad \mathbf{i}^\wedge(Y) = \{i \in \mathcal{I} \mid \;\; Y \subseteq \mathbf{t}(i)\}$

Thus $\mathbb{C}^\wedge = \mathbf{i}^\wedge \circ \mathbf{t}$ forms a closure operator for AND-clauses [19, 50]. Note also that $t \in \mathcal{T}$ satisfies $X \in \mathcal{E}^\wedge$ iff t satisfies all literals $l \in X$, i.e., $t \in \mathbf{t}(X) \iff t \in \mathbf{t}(l), \; \forall l \in X$. This implies that $\mathbf{t}(X) = \bigcap_{l \in X} \mathbf{t}(l)$. Consider our example dataset from Figure 5.1; $\mathbb{C}^\wedge(AC) = \mathbf{i}^\wedge(\mathbf{t}(AC)) = \mathbf{i}^\wedge(\mathbf{t}(A) \cap \mathbf{t}(C)) = \mathbf{i}^\wedge(134 \cap 123) = \mathbf{i}^\wedge(13) = ACD$. Thus ACD is a closed AND-clause, whereas AC is one of its (minimal) generators. Table 5.1 shows the set of all closed AND-clauses, as well as their minimal generators.

While one can mine all possible AND-clauses (i.e., all frequent itemsets), it is better to mine just the closed clauses and their minimal generators which are the most compact

Tidset	Closed	Min Generators
3	ABCD	AB, BD
4	ADE	AE, DE
13	ACD	AC, CD
23	BC	B
45	E	E
123	C	C
134	AD	A, D

Table 5.1: Closed (CA) and Minimal Generators (MA) for AND-clauses

representations of the boolean patterns without losing any information. Thus we define two boolean expression mining tasks related to AND-clauses: 1) Mine all the closed AND-clauses (**CA**), and 2) Mine all the minimal generators of AND-clauses (**MA**).

5.2.2 Pure Disjunctions: OR-clauses

While AND-clause mining has been previously studied in the form of itemset mining, OR-clause mining is a new task we propose. We will define a novel closure operator on the OR-clauses, which will enable us to extract all the closed OR-clauses and their minimal generators.

Here we restrict $\mathcal{E}^{\vee}$ to denote the set of all OR-clauses over $\mathcal{I}$; $\mathcal{T}$ is the set of all tids as before. For an OR-clause X, let $\mathcal{L}(X) = \{l | l \text{ is a literal in } X\}$ denote the set of its literals. Given $X, Y \in \mathcal{E}^{\vee}$, we define the relation $\preceq$ between OR-clauses as follows: $X \preceq Y$ iff $\mathcal{L}(X) \subseteq \mathcal{L}(Y)$. Then the relation $\preceq$ induces a partial order over $\mathcal{E}^{\vee}$. For example, $A|C \preceq A|B|C|D$, since $\mathcal{L}(A|C) = AC \subseteq ABCD = \mathcal{L}(A|B|C|D)$. Below for convenience we also write an OR-clause $l_1|l_2|\cdots|l_k$ as $\bigvee(l_1 l_2 \cdots l_k)$.

Let $X \in \mathcal{E}^{\vee}$ be an OR-clause; we define $\mathbf{t}(X) = \bigcup_{l \in X} \mathbf{t}(l)$, where l is a literal.

Theorem 1 *Let $(\mathcal{E}^{\vee}, \preceq)$ be the partial order over OR-clauses, and let $(2^{\mathcal{T}}, \supseteq)$ be the partial order over $2^{\mathcal{T}}$. Let $X \in \mathcal{E}^{\vee}$ be some OR-clause, then the following two mappings form a Galois connection over $\mathcal{E}^{\vee}$ and $2^{\mathcal{T}}$:*

- $\rho = \mathbf{t} : \mathcal{E}^{\vee} \mapsto 2^{\mathcal{T}}, \quad \mathbf{t}(X) = \{t \in \mathcal{T} \mid \ t \textit{ satisfies } X\}$
- $\eta = \mathbf{i}^{\vee} : 2^{\mathcal{T}} \mapsto \mathcal{E}^{\vee}, \quad \mathbf{i}^{\vee}(Y) = \bigvee\{i \in \mathcal{I} \mid \ \mathbf{t}(i) \subseteq Y\}$

PROOF: *We prove the conditions of Galios connection as follows. Let* $X, X_1, X_2 \in \mathcal{E}^{\vee}$ *and* $Y, Y_1, Y_2 \in 2^{\mathcal{T}}$.

1) Show that $X_1 \preceq X_2 \Longrightarrow \rho(X_2) \supseteq \rho(X_1)$*:*

$X_1 \preceq X_2 \Longrightarrow \mathcal{L}(X_1) \subseteq \mathcal{L}(X_2) \Longrightarrow \bigcup_{l \in \mathcal{L}(X_1)} \mathbf{t}(l) \subseteq \bigcup_{l \in \mathcal{L}(X_2)} \mathbf{t}(l) \Longrightarrow \mathbf{t}(X_1) \subseteq \mathbf{t}(X_2) \Longrightarrow \rho(X_1) \subseteq \rho(X_2)$.

2) Show that $Y_1 \supseteq Y_2 \Longrightarrow \eta(Y_2) \preceq \eta(Y_1)$*:*

Let $Y_1 \supseteq Y_2$*. Then* $i \in \mathcal{L}(\eta(Y_2)) \Longrightarrow \mathbf{t}(i) \subseteq Y_2 \subseteq Y_1 \Longrightarrow i \in \mathcal{L}(\eta(Y_1))$*, so* $\mathcal{L}(\eta(Y_2)) \subseteq \mathcal{L}(\eta(Y_1)) \Longrightarrow \eta(Y_2) \preceq \eta(Y_1)$.

3) Show that $X \preceq \eta(\rho(X))$*:*

$i \in \mathcal{L}(X) \Longrightarrow \mathbf{t}(i) \subseteq \mathbf{t}(X) \Longrightarrow i \in \mathcal{L}(\eta(\rho(X))) \Longrightarrow \mathcal{L}(X) \subseteq \mathcal{L}(\eta(\rho(X))) \Longrightarrow X \preceq \eta(\rho(X))$.

4) Show that $Y \supseteq \rho(\eta(Y))$*:*

Let $t \in \mathbf{t}(\eta(Y))$*, then* t *satisfies* $\eta(Y)$*. Assume that* $t \notin Y$*, then* t *does not satisfy* $\eta(Y)$*, which is a contradiction. So* $t \in Y$*, which implies* $\mathbf{t}(\eta(Y)) \subseteq Y$*, i.e.,* $\rho(\eta(Y)) \subseteq Y$. ■

Thus $\mathbb{C}^{\vee} = \mathbf{i}^{\vee} \circ \mathbf{t}$ forms a closure operator for OR-clauses. For example, in our example dataset from Figure 5.1, $\mathbb{C}^{\vee}(A|C) = \mathbf{i}^{\vee}(\mathbf{t}(A|C)) = \mathbf{i}^{\vee}(1234) = A|B|C|D$. Thus $A|B|C|D$ is a closed OR-clause, whereas $A|C$ is one of its (minimal) generators. Table 5.2 shows the set of all closed OR-clauses, as well as their minimal generators.

Tidset	Closed	Min Generators
23	B	B
45	E	E
123	$B\|C$	C
134	$A\|D$	A, D
1234	$A\|B\|C\|D$	$A\|B, A\|C, B\|D, C\|D$
1345	$A\|D\|E$	$A\|E, D\|E$
2345	$B\|E$	$B\|E$
12345	$A\|B\|C\|D\|E$	$A\|B\|E, B\|D\|E, C\|E$

Table 5.2: Closed (CO) and Minimal Generators (MO) for OR-clauses

We define two new mining tasks related to OR-clauses: 1) Mine all the closed OR-clauses (**CO**), and 2) Mine all the minimal generators of OR-clauses (**MO**).

5.2.3 CNF-expressions

Let $\mathcal{E}^{\text{cnf}}$ denote the set of all boolean expressions in CNF, i.e., each $X \in \mathcal{E}^{\text{cnf}}$ is an AND of OR-clauses. By definition $\mathcal{E}^{\vee} \subseteq \mathcal{E}^{\text{cnf}}$. Also $\mathcal{E}^{\wedge} \subseteq \mathcal{E}^{\text{cnf}}$, since an AND-clause is a CNF-expression over (OR) clauses consisting of a single literal. From Section 5.2.2 we know that $\mathbb{C}^{\vee}$-closed expressions completely characterize all possible tidsets that can be obtained by pure disjunctions over $\mathcal{I}$. Moreover, the minimal $\mathbb{C}^{\vee}$-generators are the minimal expressions that characterize those tidsets. Thus we assume that we have already available the set of all closed $\mathbb{C}^{\vee}(\mathcal{E}^{\vee})$ and minimal $\mathbb{C}^{\vee}$-generators $\mathcal{G}^{\min}(\mathcal{E}^{\vee})$, as well as their corresponding tidsets.

For convenience we denote a CNF-expression X as $X = \bigwedge X_i$, where each X_i is an OR-clause. We say that X is a *min-CNF-expression* if for each OR-clause X_i there does not exist another X_j ($i \neq j$) such that $X_i \preceq X_j$. Note that any CNF-expression can easily be made a min-CNF-expression by simply deleting the offending clauses. For example in the CNF-expression $(B|C)(A|B|C|D)$, we have $B|C \preceq A|B|C|D$; thus the expression is logically equivalent to its min-CNF form $(B|C)$. For any CNF-expression X, we use the notation $\min^{\text{cnf}}(X)$ to denote its min-CNF form.

Let $\mathcal{E}_m^{\text{cnf}}$ denote the set of all min-CNF-expressions. Given $X, Y \in \mathcal{E}_m^{\text{cnf}}$, with $X = \bigwedge X_i$ and $Y = \bigwedge Y_i$, we say that X is more general than Y, denoted $X \sqsubseteq_c Y$, if there exists a 1-1 mapping ϕ that maps each $X_i \in X$ to $\phi(X_i) = Y_j \in Y$, such that $X_i \preceq Y_j$. For example, in the dataset from Figure 5.1, assume $X = (C \cup D) \cap (B \cup E)$ and $Y = (A \cup C \cup D) \cap (B \cup E)$, then $X \sqsubseteq_c Y$ if we define ϕ: $\phi(X_i) \subseteq X_i$ and $\mathbf{t}(\phi(X_i)) = \mathbf{t}(X_i)$ ($\phi(C \cup D) = (A \cup C \cup D)$).

Theorem 2 *Let $(\mathcal{E}_m^{cnf}, \sqsubseteq_c)$ be the partial order over min-CNF-expressions, and let $(2^{\mathcal{T}}, \subseteq)$ be the partial order over $2^{\mathcal{T}}$. The following two mappings form a Galois connection over $\mathcal{E}_m^{cnf}$ and $2^{\mathcal{T}}$:*

- $\rho = \mathbf{t} : \mathcal{E}_m^{cnf} \mapsto 2^{\mathcal{T}}$, $\mathbf{t}(X) = \{t \in \mathcal{T} \mid t \text{ satisfies } X\}$
- $\eta = \mathbf{i}^c : 2^{\mathcal{T}} \mapsto \mathcal{E}_m^{cnf}$, $\mathbf{i}^c(Y) = \bigwedge\{\min^{cnf}(X) \mid X \in \mathbb{C}(\mathcal{E}^{\vee}) \wedge Y \subseteq \mathbf{t}(X)\}$

PROOF: *If we let each minimal $\mathbb{C}^{\vee}$ generator be a new item, then the proof becomes the same Galois connection proof as section 5.2.1.* ■

Thus $\mathbb{C}^c = \mathbf{i}^c \circ \mathbf{t}$ forms a closure operator for min-CNF-expressions. Also note that we can directly compute the minimal generators of a closed CNF-expression $X = \bigwedge X_i$, by constructing a CNF-expression by taking an AND of minimal generators of each X_i. For example, in our example dataset from Figure 5.1, $\mathbb{C}^c(A(B|E)) = \mathbf{i}^c(\mathbf{t}(A(B|E)) = \mathbf{i}^c(34) = (A|D)(B|E)$. Thus $(A|D)(B|E)$ is a closed min-CNF-expression, whereas $A(B|E)$ is one of its (minimal) generators. Table 5.3 shows the closed CNF expressions and their minimal generators in addition to those already shown in Tables 5.2 and 5.1.

Tidset	Closed	Min Generators
34	$(A\|D)(B\|E)$	$A(B\|E), D(B\|E)$
234	$(A\|B\|C\|D)(B\|E)$	$(A\|B)(B\|E)$, $(A\|C)(B\|E)$, $(B\|D)(B\|E)$, $(C\|D)(B\|E)$
345	$(A\|D\|E)(B\|E)$	$(A\|E)(B\|E)$, $(D\|E)(B\|E)$

Table 5.3: Additional Closed (CC) and Minimal Generators (MC) for CNF

We define two new mining tasks, namely mining all the closed min-CNF-expressions (**CC**), and their minimal generators (**MC**).

5.2.4 DNF-expressions

Let $\mathcal{E}^{\text{dnf}}$ denote the set of all boolean expression in DNF, i.e., each $X \in \mathcal{E}^{\text{dnf}}$ is an OR of AND-clauses. By definition $\mathcal{E}^{\wedge} \subseteq \mathcal{E}^{\text{dnf}}$. Also $\mathcal{E}^{\vee} \subseteq \mathcal{E}^{\text{dnf}}$, since an OR-clause is a DNF-expression over single literal (AND) clauses. From Section 5.2.1 we know that $\mathbb{C}^{\wedge}$-closed expressions completely characterize all possible tidsets that can be obtained by pure conjunctions over $\mathcal{I}$. Moreover, the minimal $\mathbb{C}^{\wedge}$-generators are the minimal expressions that characterize those tidsets. Thus we assume that we have already available the set of all closed $\mathbb{C}^{\wedge}(\mathcal{E}^{\wedge})$ and minimal $\mathbb{C}^{\wedge}$-generators $\mathcal{G}^{\min}(\mathcal{E}^{\wedge})$, as well as their corresponding tidsets.

For convenience we denote a DNF-expression X as $X = \bigvee X_i$, where each X_i is an AND-clause. We say that X is a *min-DNF-expression* if for each AND-clause X_i there does not exist another X_j ($i \neq j$) such that $X_i \subseteq X_j$. Note that any DNF-expression can easily be made a min-DNF-expression by simply deleting the offending clauses. For example in the DNF-expression $(AD)|(ADE)$, we have $AD \subseteq ADE$;

thus the expression is logically equivalent to its min-DNF form (AD). For any DNF-expression X, we use the notation $\min^{\text{dnf}}(X)$ to denote its min-DNF form.

Let $\mathcal{E}_m^{\text{dnf}}$ denote the set of all min-DNF-expressions. Given $X, Y \in \mathcal{E}_m^{\text{dnf}}$, with $X = \bigvee X_i$ and $Y = \bigvee Y_i$, we say that X is more general than Y, denoted $X \sqsubseteq_d Y$, if there exists a 1-1 mapping ϕ that maps each $X_i \in X$ to $\phi(X_i) = Y_j \in Y$, such that $X_i \subseteq Y_j$.

Theorem 3 *Let $(\mathcal{E}_m^{dnf}, \sqsubseteq_d)$ be the partial order over min-DNF-expressions, and let $(2^{\mathcal{T}}, \supseteq)$ be the partial order over $2^{\mathcal{T}}$. The following two mappings form a Galois connection over $\mathcal{E}_m^{dnf}$ and $2^{\mathcal{T}}$:*

- $\rho = \mathbf{t} : \mathcal{E}_m^{dnf} \mapsto 2^{\mathcal{T}}$,
  $\mathbf{t}(X) = \{t \in \mathcal{T} \mid t \textit{ satisfies } X\}$
- $\eta = \mathbf{i}^d : 2^{\mathcal{T}} \mapsto \mathcal{E}_m^{dnf}$,
  $\mathbf{i}^d(Y) = \bigvee\{\min^{dnf}(X) \mid X \in \mathbb{C}(\mathcal{E}^{\wedge}) \wedge \mathbf{t}(X) \subseteq Y\}$

PROOF: *If we let each minimal $\mathbb{C}^{\wedge}$ generator be a new item, then the proof becomes the same Galois connection proof as section 5.2.2.* ■

Thus $\mathbb{C}^d = \mathbf{i}^d \circ \mathbf{t}$ forms a closure operator for min-DNF-expressions. Also note that we can directly compute the minimal generators of a closed DNF-expression $X = \bigvee X_i$, by constructing a DNF-expression by taking an OR of minimal generators of each X_i. For example, in our example dataset from Figure 5.1, $\mathbb{C}^d(B|(AE)) = \mathbf{i}^d(\mathbf{t}(B|(AE)) = \mathbf{i}^d(234) = B|(AE)|(DE)$. Thus $B|(AE)|(DE)$ is a closed min-DNF-expression, whereas $B|(AE)$ is one of its (minimal) generators. Table 5.4 shows the closed DNF expressions and their minimal generators in addition to those already shown in Tables 5.2 and 5.1.

Tidset	Closed	Min Generators
34	$(ABCD)\|(ADE)$	$(AB)\|(AE)$, $(AB)\|(DE)$, $(BD)\|(AE)$, $(BD)\|(DE)$
234	$(BC)\|(ADE)$	$B\|(AE)$, $B\|(DE)$
345	$(ABCD)\|E$	$(AB)\|E$, $(BD)\|E$

Table 5.4: Additional Closed (CD) and Minimal Generators (MD) for DNF

We define two new mining tasks, namely mining all the closed min-DNF-expressions (**CD**), and their minimal generators (**MD**).

5.2.5 A Note on Negated Literals

The closure operators proposed above are equally valid for any literal (i or $\bar{i}$, $i \in \mathcal{I}$). However, when including negative literals, we have to disallow simple tautologies like the OR-clause $i|\bar{i}$ which is always true (i.e., its tidset is $\mathcal{T}$). Similarly, we must disallow any AND-clause containing $i \wedge \bar{i}$ since this is always false (its tidset is $\emptyset$). This problem gets exacerbated with CNF/DNF expressions: for example, $A(\bar{A}|B) = (A\bar{A})|(AB) = AB$. In our current BLOSOM implementation, we treat each literal (both positive and negative) as unique. Thus our algorithms would find logically redundant patterns. As part of future work we are working on designing tautological constraints to eliminate such redundancies.

5.3 The BLOSOM Framework

The BLOSOM framework for mining arbitrary boolean expressions supports several different algorithms, namely BLOSOM-CA and BLOSOM-MA for mining closed and minimal generators of AND-clauses, BLOSOM-CO and BLOSOM-MO for mining closed and minimal generators of OR-clauses, BLOSOM-MC for mining minimal generators of CNF expressions and finally BLOSOM-MD for mining minimal generators of DNF expressions. Although not implemented, we show how these basic algorithms can be used as building blocks for the mining of closed CNF/DNF expressions.

BLOSOM consists of two main algorithms that are further specialized. The first one deals with mining the minimal generators for all four types of expressions (MA, MO, MC, MD), and the second one deals with closed expressions (CA, CO); we are currently extending the latter to mine CC and CD as well. Another distinction is to be made for mining pure conjunctions (CA, MA) and pure disjunctions (CO, MO) versus mining CNF/DNF expressions (MC, MD), since the latter require that we first mine (MO, MA), and then we have to reapply (MA, MO) over the OR-/AND-clauses. For example for mining MC, we first mine MO, and then mine MA over the minimal OR-clauses. Thus MC and MD mining are two-phased, whereas (CA, CO, MA, MO) are all single-phased.

Note also that by DeMorgan's law, to get MA we can mine MO over the complemented tidsets. For example in our example dataset in Figure 5.1, with $\mathcal{T} = 12345$, we have $\mathbf{t}(AB) = \mathbf{t}(A) \cap \mathbf{t}(B) = 134 \cap 23 = 3$. We can obtain the same results if we

take the OR of $\bar{A}$ and $\bar{B}$ and complement the final results. For example $\overline{\mathbf{t}(\bar{A})|\mathbf{t}(\bar{B})} = \overline{\mathbf{t}(\bar{A}) \cup \mathbf{t}(\bar{B})} = \overline{25 \cup 145} = \overline{1245} = 3$. In other words, just as a NAND or NOR gate is sufficient to realize all boolean expressions, a single algorithm for mining minimal generators, namely BLOSOM-MO is sufficient to mine all other minimal generators. That is, BLOSOM-MO directly mines all OR-clauses; to mine MA, we mine MO over complemented tidsets; to mine MC, we first mine MO for minimal generators, and then apply MO over their complemented tidsets to obtain MA of OR-clauses; to mine MD, we first mine MO over complemented tidsets to obtain minimal AND-clauses, and then apply MO over them a second time to get DNF-expressions. Likewise for mining closed sets we develop only one basic method BLOSOM-CO; this can be used to mine all other closed expressions, using a single phase for CO and CA, and using two phases for CC and CD (the latter two are currently being implemented).

BLOSOM assumes that the input dataset is $\mathcal{D}$, and it then transforms it to work with the transposed dataset $\mathcal{D}^T$. Starting with the single items (literals) and their tidset, BLOSOM performs a depth-first search (DFS) extending an existing expression by one more "item". BLOSOM employs a number of effective pruning techniques for searching over the space of boolean expressions, yielding orders of magnitude in speedup. These include: dynamic sibling reordering, parent-child relationship pruning, sibling merging, threshold pruning, and fast subsumption checking. Further BLOSOM utilizes a novel *extraset* data structure for fast frequency computations, and to identify the corresponding transaction set for a given arbitrary boolean expression.

5.3.1 BLOSOM-MO **Algorithm**

Figure 5.2 shows the outline of the BLOSOM-MO algorithm for mining minimal generators. It conceptually utilizes a DFS tree to search over the OR-clauses. Each OR-clause is stored as a set of items (the OR is implicitly assumed). Thus each node of the search tree is a pair of ($T \times I$), where T is a transaction set and I is an item set denoted an OR-clause. In the description below, we use MO to denote a minimal generator of OR-clauses.

Input : min_sup, max_sup, max_item and dataset $\mathcal{D}$
Output : a hash table $\mathcal{M}$ containing all MO of $\mathcal{D}$
Initialization: $NL = \emptyset$; for each item i of $\mathcal{D}$, add set pair $(\mathbf{t}(i) \times \{i\})$ to NL; call BLOSOM($\emptyset,\ NL,\ \emptyset$)

BLOSOM-MO($\mathcal{P}_0,\ NL_0,\ T_0$)
1 **sort** $NL_0 = \{n_i\}$ **in decreasing order of** $|n_i.T|$**.**
2 **while** $\exists\, n_1$ ***and*** $n_2 \in NL_0$ ***such that*** $n_1.T = n_2.T$ **do**
3 $n_1.I \leftarrow n_1.I \cup n_2.I$
4 $NL_0 \leftarrow NL_0 - n_2$
5 **foreach** $n_1 \in NL_0$ **do**
6 $\mathcal{P}_1 \leftarrow \mathcal{P}_0 + n_1.I$
7 $T_1 \leftarrow T_0 \cup n_1.T$
8 **if** $|T_1| > max\_sup$ **then**
9 **continue /* goto line 5 */**
10 **if** $|T_1| \geq min\_sup$ **then**
11 **foreach** ***combination*** I_1 ***of*** $\mathcal{P}_1$ **do**
 /* I_1 **is generated by picking one and only one item from each** $P \in \mathcal{P}_1$**. */**
12 **if** $\mathcal{M}$ ***contains entry*** T_1 **then**
13 **delete all supersets of** I_1 **in** $\mathcal{M}$**[**T_1**].**
14 $\mathcal{M}[T_1] \leftarrow \mathcal{M}[T_1] + I_1$
15 **if** $|\mathcal{P}_1| \geq max\_item$ **then**
16 **continue /* goto line 5 */**
17 $NL_1 \leftarrow \emptyset$
18 **foreach** $n_2 \in NL_0$ ***with rank after*** n_1 **do**
19 $n_3.T \leftarrow n_2.T - n_1.T$
20 **if** $|n_3.T| \neq \emptyset$ **then**
21 $n_3.I \leftarrow n_2.I$
22 $NL_1 \leftarrow NL_1 + n_3$
23 **if** $NL_1 \neq \emptyset$ **then**
24 BLOSOM($\mathcal{P}_1,\ NL_1,\ T_1$)

Figure 5.2: BLOSOM-MO **Algorithm**

5.3.2 Pruning and Optimization Techniques

Before describing the pseudo-code we briefly describe each of the optimizations used in the BLOSOM-MO for fast expression mining.

Dynamic Sibling Reordering: Before expanding a group of sibling nodes in the search tree, BLOSOM-MO dynamically reorders them by their tidset size dynamically. Since smaller tidsets are more likely to be contained in longer previous tidsets, this can prune out many branches, in combination with several optimizations below.

Relationship Pruning: BLOSOM-MO makes use of two kinds of relationship pruning: parent-child and sibling based. During the DFS expansion, if the tidset of a node is the same as any of its parents', the node and all of its descendants are pruned (based on the definition of minimal generators). If some sibling nodes of the same parent node have the same tidset, they are merged together by unioning their itemsets into a "composite node". A prefix item set data structure is utilized to deal with siblings merging. All sibling nodes of the same father share a common prefix, containing the item sets on the path from the root to the current node. The prefix also saves memory, since all the siblings do not need to keep their separate prefix copies.

Threshold Pruning: BLOSOM uses three thresholds to prune the trivial generators, based on min_sup, max_sup and max_item. Furthermore, if the item set I of the current node is the whole set of items $\mathcal{I}$, then we stop its expansion immediately, since any descendant of it will be pruned.

Fast Subsumption Checking in One Direction: BLOSOM-MO maintains a hash table for storing the current MO, whose hash keys are integers, representing the summation of the tids of a transaction set. An element of the hash table is a pair of ($T \times MS$), where T is a transaction set and MS is a set of T's MOs. Subsumption checking on MOs guarantees that the current generators are minimal, i.e. for some transaction set T, any two of its generators do not contain each other. Before adding a new generator $G(T \times I)$ to a hash-cell, BLOSOM first checks if there is such a hash table entry of T. If not, we create one. Otherwise, remove supersets of I in MS, and add I. By our enumeration process, it is not necessary to check if G is minimal, i.e., G cannot be a superset of any MO of entry T, otherwise G would have been pruned previously by one of its ancestor nodes using the parent-child pruning. So we only need to do the subsumption checking in one direction.

Extraset Technique: BLOSOM-MO utilizes a novel $extraset$ technique to save set operation time and memory for tidsets. The DFS expansion involves a lot of tidset unions, and each node has to maintain such an intermediate set, which is a superset of the parent node's tidset. So for each node, if we just retain the extra-part of its parent's tidset in a data structure we call $extraset$, we will lose no information, yet we will save a lot of memory while storing the intermediate tidsets. Thus an $extraset$ is the opposite of the diffsets used in Charm [51]. In addition, the corresponding set union operations on the original transaction set will be changed to the operations on the $extraset$ in the following subtree enumerations, which also saves lots of time for the size shrinkage of the operand sets. It is simple to prove that the $extraset$ of the next level can be generated using the $extraset$ of the current level in the tree expansion process. Additional costs for the $extraset$ application are to calculate the whole transaction set and do the set complementation when applied to MA generation. In spite of these costs, the utilization of the $extraset$ still speeds up the algorithm and saves memory and time for set operations, especially for dense datasets.

5.3.3 Algorithmic Description

BLOSOM-MO takes as input the set of parameter values min_sup, max_sup, max_item and a dataset $\mathcal{D}$ (we implicitly convert it to $\mathcal{D}^T$). The max_item constraint is used to limit the maximum size of any boolean expression, if desired. BLOSOM-MO is a recursive algorithm that, at each call accepts a current prefix queue $\mathcal{P}_0$ containing the I sets on the path from the root to the current node, a node list NL_0 containing a group of sibling nodes of the same parent, and the current transaction set T_0. The initial call is made with an empty $\mathcal{P}_0$, the NL containing all the original single item generators and an empty T_0. Line 1 sorts the current sibling nodes in NL_0 in decreasing order according to their T sizes, which will speed up the convergence of the algorithm (dynamic sibling reordering). Lines 2-4 merge the sibling nodes with the same transaction set T by unioning their item sets together (sibling merging). Lines 5-24 are the main loop to process each of the sibling nodes one by one. Line 6 updates the current prefix queue for further recursive calling. Line 7 generates the current transaction set T_1 by unioning the $extraset$ $n_1.T$ and the parent's transaction set T_0. Lines 8-9 check the max_sup threshold, which is set because

too frequent item sets may be out of the user's interest (for disjunctions). Lines 10-14 try to add the current sibling node satisfying min_sup and max_sup to the hash table $\mathcal{M}$. If the node satisfies max_sup (line 8) and min_sup (line 10), then the algorithm will try to add every possible queue $\mathcal{P}_1$ items combination to the current MO hash table (lines 11-14). Each combination is generated by picking one and only one item from each $P \in \mathcal{P}_1$. At the same time, we also need to do subsumption checking and to delete any supersets of the new MO I_1 to be added (line 13). The new MO I_1 must be minimal, i.e. it is not subsumed by any current MO in the hash table. The reason is that if it were not minimal it would have been pruned by one of its ancestor nodes, so the subsumption checking in reverse direction is unnecessary (fast subsumption checking in one direction). Lines 15-16 check the max_item threshold. Since $|n_1.I|$ may be greater than 1 because of sibling merging, we can not jump out (line 16) of the function until we have tried every sibling node. Lines 17-22 produce node n_1's valid child nodes generated with other sibling nodes ranking behind n_1. First, line 17 initializes the valid children node list as empty. Then it tries every node pair (line 18) and generates child nodes (line 19). Here BLOSOM-MO only saves the $extraset$ ($n_1.T \cup n_2.T - n_1.T = n_2.T - n_1.T$) to save memory and set operation time which proves very efficient for dense datasets. If a child node is not subsumed by its parents (line 20), then it is added to the valid children node list (lines 21-22) (note: line 20 constitutes the parent-child relationship pruning). Lines 23-24 make a recursive call with all the node n_1's valid children.

5.3.4 An Example

Here we show an example of how BLOSOM-MO works on our example dataset from Figure 5.1. Initially the prefix queue $\mathcal{P}_0$ is empty, and node list NL contains five single item generators as shown in $\mathcal{D}^T$ in Figure 5.1. After sibling merging and reordering, they become $AD{\times}134,\ C{\times}123,\ B{\times}23$ and $E{\times}45$ as Figure 5.3 shows. Node A and node D are merged together, since they have the same transaction set. From $\mathcal{P}_1{=}\{A, D\}$, we get two item combinations: A and D. In addition $T_0{=}\emptyset$, so we add A and D to the entry $T_1{=}\emptyset \cup 134$ of the MOs hash table $\mathcal{M}$ as Table 5.5 shows, where the subscript numbers represent the order in which MOs are added to the table. Then we expand node $\{A, D\}{\times}134$ by combining with node $C{\times}123,\ B{\times}23$ and $E{\times}45$, and save the $extraset$

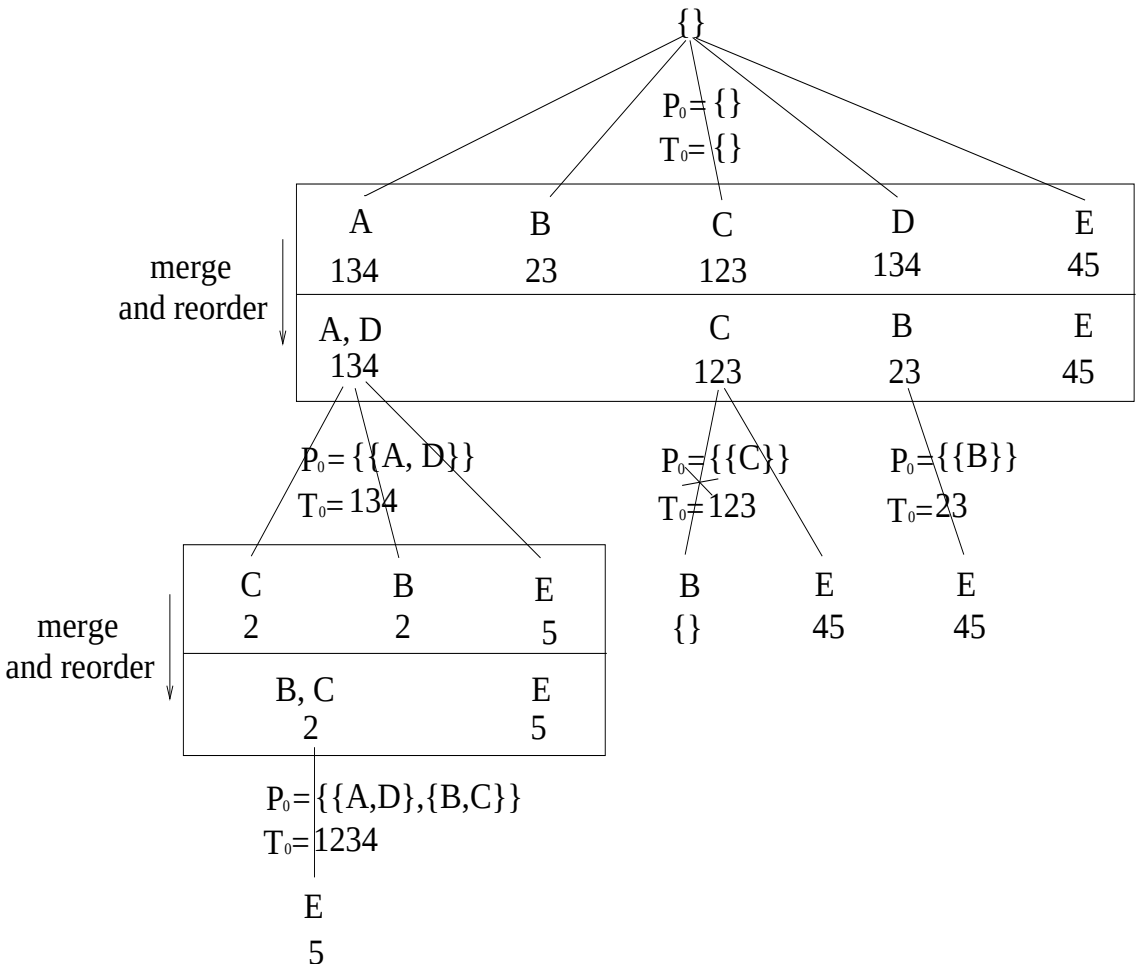


Figure 5.3: DFS Search Tree

of the transaction set union for each combination pair as follows. Since $X \cup Y - X = Y - X$, the expansion results are $123-134=2$; $23-134=2$; and $45-134=5$. After reordering and merging, $NL_1=\{\{B,C\}\times 2,\ E\times 5\}$, then with $\mathcal{P}_0=\{\{A,D\}\}$ and $T_1=134$, BLOSOM is called recursively. Next step from $\mathcal{P}_1=\{\{A,D\},\{B,C\}\}$, we will get four combinations: $\{A,B\},\ \{A,C\},\ \{B,D\}$ and $\{C,D\}$, and add them to the entry $T_1=134 \cup 2=1234$ of $\mathcal{M}$. Readers can continue this process until all the MOs are generated as Table 5.5 shows with the subscript order. Please note in path $Root \rightarrow C \rightarrow B$, node $B\times\emptyset$ is pruned using parent-child relationship because of its empty $extraset$ (i.e. generator BC is not minimal and generates the same transaction set as one of its parents). In addition, when generator CE is added to entry 12345 of $\mathcal{M}$, previously added MOs ACE and CDE of the en-

try should be deleted (as marked by underlines) for they are subsumed by (superset of) generator CE.

T Set	Minimal OR Generators
134	$\{A\}_1, \{D\}_1$
1234	$\{AB\}_2, \{AC\}_2, \{BD\}_2, \{CD\}_2$
12345	$\{ABE\}_3, \underline{\{ACE\}_3}, \{BDE\}_3, \underline{\{CDE\}_3}$ $\{CE\}_6$
1345	$\{AE\}_4, \{DE\}_4$
123	$\{C\}_5$
23	$\{B\}_7$
2345	$\{BE\}_8$
45	$\{E\}_9$

Table 5.5: Addition of Minimal Generators (MOs) to Hash Table $\mathcal{M}$

5.3.5 The BLOSOM-CO Algorithm

Whereas BLOSOM-MO is designed for mining minimal generators, BLOSOM-CO is specific to mining closed expressions (both CA and CO, and it can also be extended to mine closed CNF/DNF expressions: CC and CD). While the overall framework is the same, there are essential differences in the processes that a separate algorithm is required. Due to space limitations we do not describe the algorithm in detail. We simply mention two additional pruning optimizations utilized in BLOSOM-CO and leave the full details for later expansions. The two new pruning techniques are as follows: 1) **Sibling Containment Pruning:** After sibling merging, another relationship among siblings can be utilized for CO generation, i.e. containment relationship. When expanding a node N, any sibling node ranking after N whose transaction set is a subset of that of N can be pruned in the next level of N's expansion tree, and its item set is merged (union together) to the item set of current node. 2) **Hash Pruning:** Assume $\mathcal{M}$ is the hash table of COs. If we find the item set of the current tree node ($T \times I$) is subsumed by the entry of $\mathcal{M}$, i.e. $I \subset \mathcal{M}(T)$, then all the descendant nodes of the current node will be pruned, whose item sets will be subsumed by that of the descendant nodes of $T \times \mathcal{M}(T)$. This pruning is formed because after sibling containment pruning, there is no one direction property for CO mining when doing subsumption check as in MO mining.

5.4 Experiments

We used both synthetic and real datasets to evaluate BLOSOM and to compare it with previous algorithms. For the real datasets we used *chess* and *connect* to test the input parameters (min_sup, max_sup and max_item) effect on BLOSOM. The chess and connect datasets are derived from their respective game steps. These datasets are taken from the UCI machine learning repository, and converted into transactional form; they are all obtained from IBM Almaden Research Center (`http://www.almaden.ibm.com/`). Chess has 3196 transactions, 76 items and its average transaction size is 36, whereas Connect has 67557 transactions, 130 items, and its average transaction size is 43.

We use synthetic datasets to evaluate the effect of other parameters like number of items, number of transactions, and dataset density, on our algorithm. The synthetic datasets are generated with three parameters: $item\#, transaction\#$ and dataset $density$. The size of the dataset is $item\# \times transaction\#$. For each item i, the average size of its transaction set $t(i)$ is $density \times transaction\#$, distributed from 0 to $transaction\#$; and the transaction IDs are distributed in each $t(i)$ with equal probability. Unless otherwise noted, all experiments were done on a Fedora virtual machine (448M memory, 1.4GHz, Pentium-M) over Windows XP through middleware VMware.

We evaluate six algorithms within the BLOSOM framework, BLOSOM-CA and BLOSOM-MA for mining closed and minimal generators of AND-clauses, BLOSOM-CO and BLOSOM-MO for mining closed minimal generators of OR-clauses, BLOSOM-MC for mining minimal generators of CNF expressions and finally BLOSOM-MD for mining minimal generators of DNF expressions. For convenience, we will use the two letter suffixes to refer to different algorithms. For example, we use CA to refer to BLOSOM-CA.

5.4.1 Effect of Optimizations

We first study the effect of various optimizations proposed in Section 5.3.2 on the performance of BLOSOM-MO, as shown in Figure 5.4. The x-axis shows the number of items in the synthetic dataset; the dataset size has the ratio, transaction#:items# = 2:1. Each curve shows the running time after applying the optimization specified in succes-

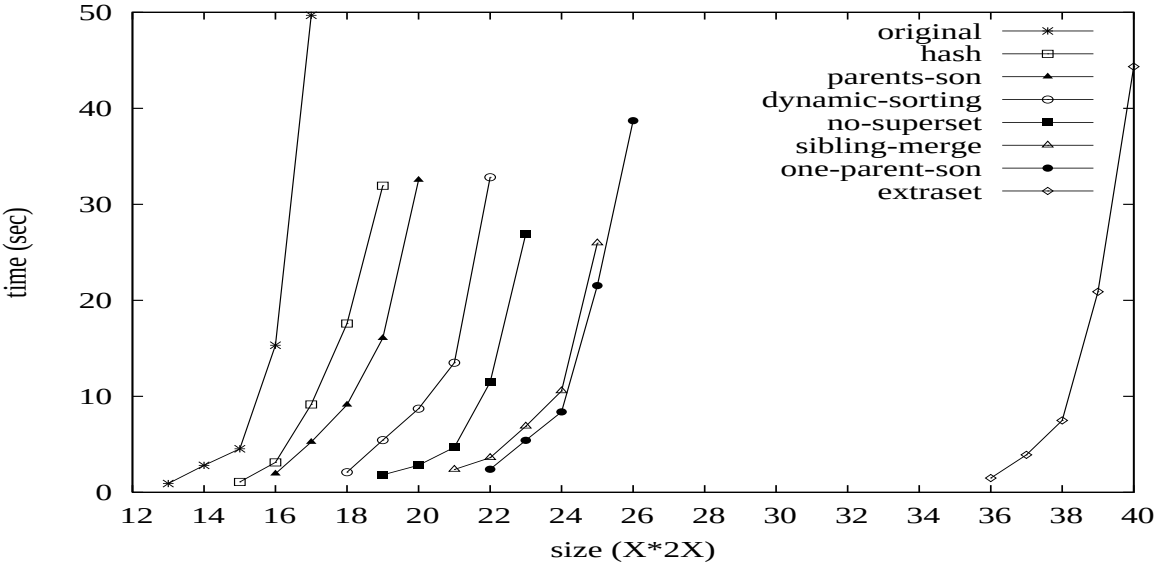


Figure 5.4: The Effects of the Speedup Optimizations

sion. Thus the final graph for $extraset$ includes all previous prunings. In the legends, `original` stands for an unoptimized version, `hash` means using a hash table for subsumption, `parent-son` means doing parent-child pruning, `dynamic-reordering` is the reordering of nodes, `no-superset` stands for one directional subsumption checking, `sibling-merging` is self explanatory, `one-parent-son` means we check only the left parent when making use of the parent-child relationship to do the pruning on the basis of the application of dynamic-sorting. Finally, we apply `extrasets`. We can see that the cumulative effect of the optimizations is substantial; BLOSOM-MO can process a nine times ($(36/12)^2$) larger dataset than the base algorithm in around the same running time. The curves suggest a total speedup of orders of magnitude over the base version.

5.4.2 Synthetic Datasets

For synthetic data we use common parameters shown in Table 5.6, and we vary one parameter at a time, as shown in Figure 5.5. The first row shows how the mining time for various expressions varies when we increase the number of items. The second row shows the effect of increasing the number of transactions. The final row shows the effect of changing the density. The columns of the figures are arranged in groups of two: first column compares closed expression mining (CA and CO), second column compares the minimal generator mining methods (MA and MO), and the last column compares minimal

Parameters	MO/MA	MC/MD
$item\#$	50	15
$trans\#$	300	30
$density$	0.5	0.35
min_sup	0	0
max_sup	infi nity	infi nity
max_item	4	3

Table 5.6: Common Experimental Parameters

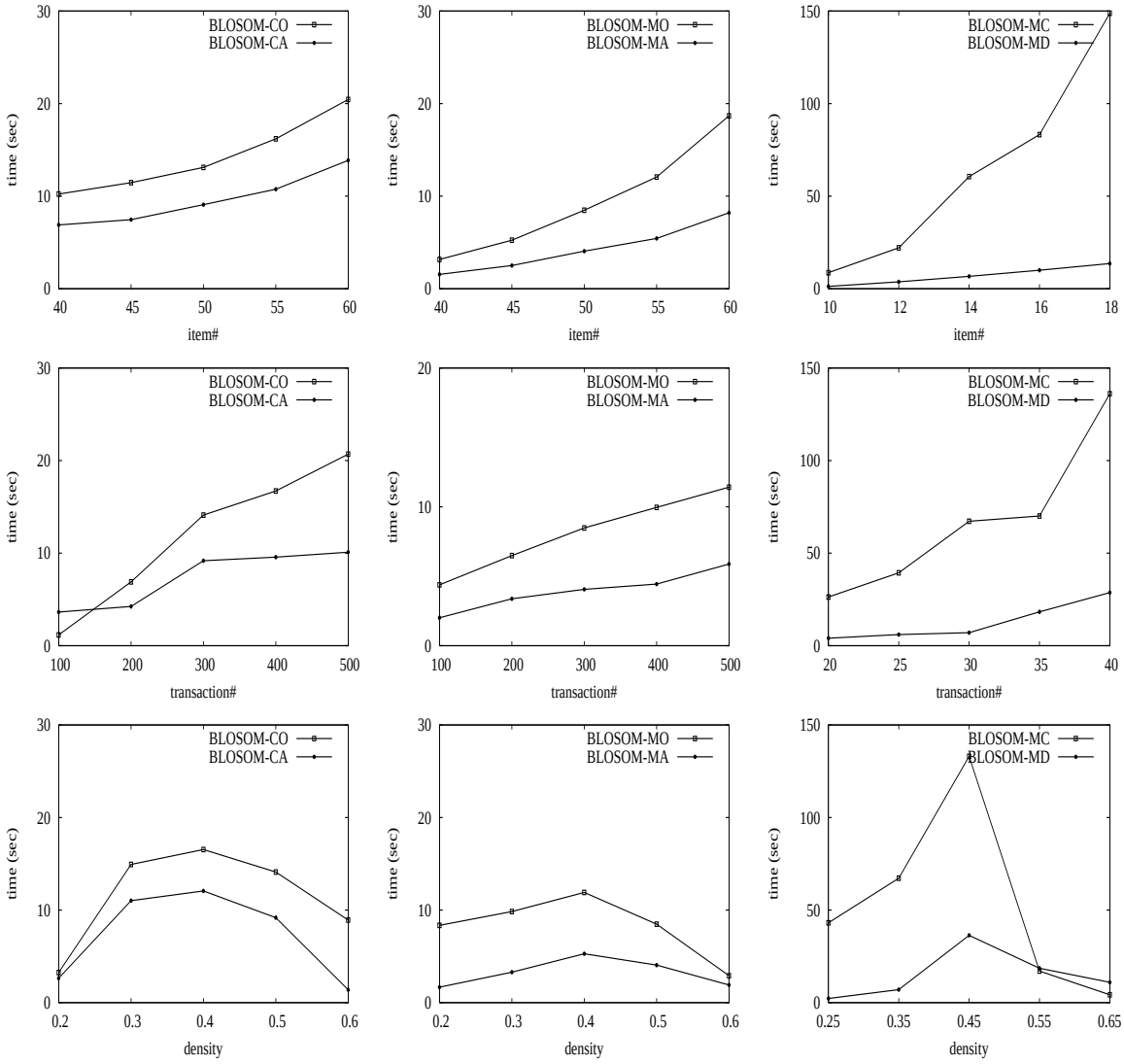


Figure 5.5: Synthetic Data: Effect of Number of Items and Transactions, and Density

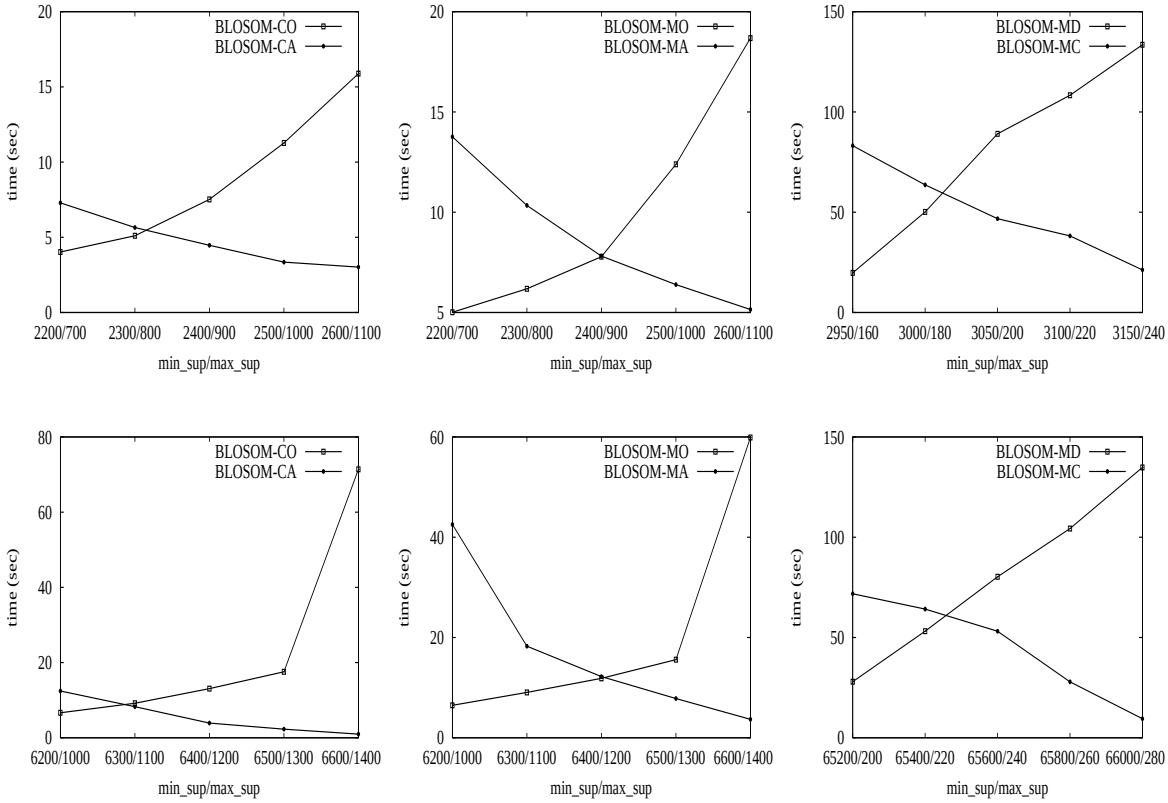


Figure 5.6: Real Data: Effect of min_sup/max_sup

CNF/DNF methods (MC/MD).

From these graphs we observe several trends. Notice that as we increase number of items/transactions and density, the disjunction times are always higher than conjunction times. That is CO is higher than CA, MO higher than MA, and MD higher than MC.

As we increase the number of items, the time increases gradually for all algorithms except MC. The reason is that first MO (for disjunctions) takes higher time to run than MA, and then MC has to compute conjunctions of these OR-clauses. The same trend is observed for increasing transactions.

We observe that as we increase density, when density is around 0.5, the running time reaches the peak for all methods. As shown in reference [52], a dataset density of 50% implies mining a database with both positive and negative literals (since each item can be part of a transaction in either negated or non-negated form) and constitutes the most expansive context for expression mining.

5.4.3 Real Datasets

We compared the different expression mining algorithms on the real datasets chess and connect as shown in Figure 5.6. The graphs show the running time as we vary the min_sup for conjunction mining and max_sup for disjunction mining. Both the support thresholds are plotted on the same x-axis. In these experiments, for MC's first phase of mining OR-clauses, we set $max\_sup = 300$ for both chess and connect, and for MD's first phase of mining AND-clauses, we set $min\_sup = 2900$ for chess and $min\_sup = 650$ for connect. We find that as min_sup increases, the running time for CA, MA, MC decreases, but as max_sup increases the time for CO, MO, MD increases.

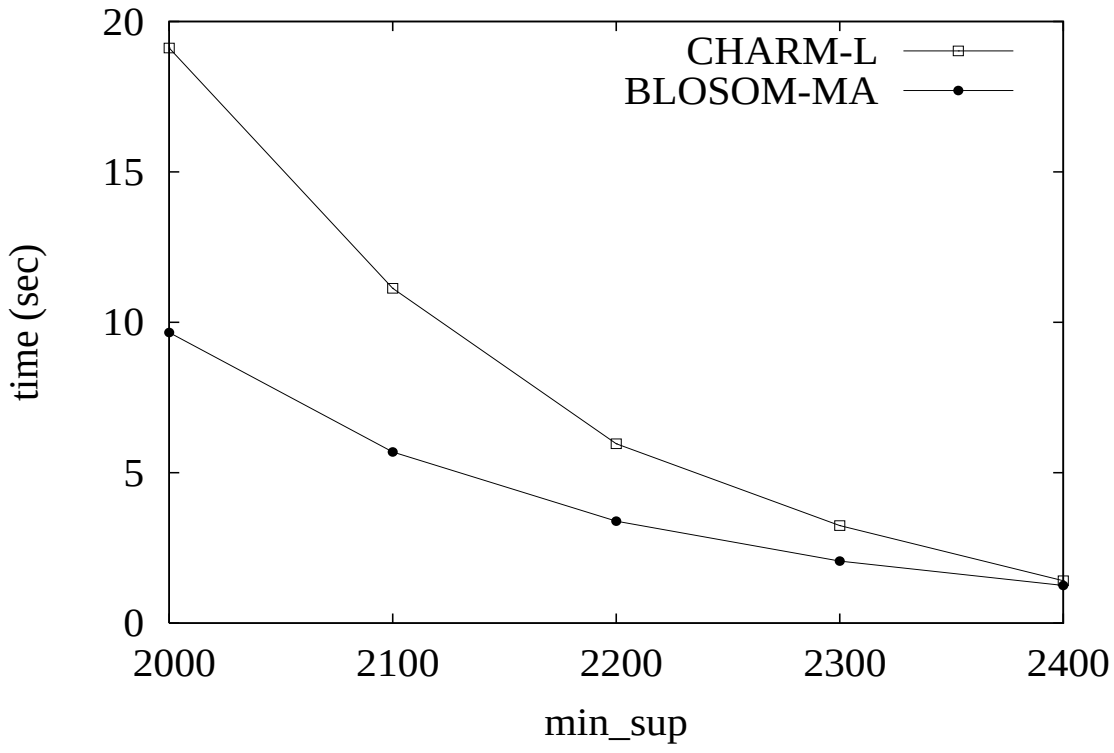


Figure 5.7: BLOSOM-MA **vs. CHARM-L**

We also compared BLOSOM-MA with a related algorithm CHARM-L [52], which can also mine the minimal generators for AND-clauses (i.e., itemsets). From Figure 5.7 we can see that BLOSOM-MA can be about two times faster than CHARM-L, and the gap is increasing with decreasing support.

5.5 Conclusions

In this chapter we present the first algorithm, BLOSOM, to simultaneously mine closed boolean expressions over attribute sets and their minimal generators. Our four-category division of the space of boolean expressions yields a compositional approach to the mining of arbitrary expressions, along with their minimal generators. The pruning operators employed here have resulted in orders of magnitude speedup, producing highly efficient implementations.

CHAPTER 6
Future Work

Our future work will expand our previous approaches along both directions: mining subspace and boolean patterns. For arithmetic patterns we consider more general notions of similarity, whereas for boolean patterns we apply our boolean expression mining to finding redescriptions between gene ontology concepts.

6.1 Mining Subspace Patterns from Real-valued Datasets

For the subspace pattern extraction from the real-valued datasets, we plan to render a new clustering model with more general definitions on the basis of our previous work. If some points sit approximately on a line in some subspace, then they will be grouped together in that subspace to form a bicluster. In this way, for any column pair, all the points of a bicluster can be expressed as a linear function pattern, i.e. $y \approx ax + b$, which covers the scaling pattern ($y \approx ax$) and shifting pattern [44] ($y \approx x + b$) as two specific cases.

Currently, MICROCLUSTER and TRICLUSTER use scaling/shifting patterns [44]. We plan to combine these two into a more general "linear" relationship pattern. In other words, all the points of a cluster are approximately (within ε tolerance) on a line in the subspace specified by the cluster, i.e., they have a linear relationship pattern in its subspace, which can be expressed as $a_0x_0 + a_1x_1 + \cdots + a_kx_k + b \approx 0$ where k is the number of columns in the bicluster. So this new problem is a more general problem, i.e., the clustering on scaling and shifting patterns are special cases of this new clustering problem. Now let's discuss the problem in detail. We name the problem as *F-cluster* (function clustering), i.e, all the points of a cluster conform to the same linear function. We define an *F-cluster* as follows.

6.1.1 Definitions and Problem Statement

Let $P = \{p_0, p_1, \cdots, p_{n-1}\}$ be a set of n points, and let $D = \{d_0, d_1, \cdots, d_{m-1}\}$ be a set of m dimensions. A dataset is a real-valued $n \times m$ matrix $H = P \times D = \{h_{ij}\}$

(with $i \in [0, n-1], j \in [0, m-1]$), whose rows and columns correspond to points and dimensions, respectively. Each entry h_{ij} records the value of point i at dimension j. For example, Table 6.1(a) shows a dataset with 8 points and 7 dimensions. Certain cells have been left blank; we assume that these are filled by some random real values.

	d_0	d_1	d_2	d_3	d_4	d_5	d_6
p_0	2	13		22			7
p_1							
p_2		6		8	8		
p_3		8		12	9		
p_4	1	10		16	10		5
p_5	4	19		34			11
p_6							
p_7	3	16		28	13		9

(a)

	d_2	d_0	d_6	d_3	d_1	d_4	d_5
p_1							
p_5		4	11	34	19		
p_0		2	7	22	13		
p_4		1	5	16	10	10	
p_7		3	9	28	16	13	
p_2				8	6	8	
p_3				12	8	9	
p_6							

(b)

Table 6.1: (a) An Example Dataset. (b) Some *F-cluster*

Let $C = X \times Y$ be a bicluster (defined in Chapter 3), we call C a **F-cluster** iff: 1) C is a maximal bicluster, and 2) there exists a line L in space Y such that for any $x \in X$, the distance of x to L on any two dimensional projected subspace of Y is less or equal to ε. This definition is given based on the following lemmas.

Lemma 5 *(Projection Property 1)*
If L is a line in k dimensional space S, then L's projection to any subspace of S is also a line.

Proof: Without loss of generality assume that L is $a_0x_0 + a_1x_1 + \cdots + a_kx_k + b = 0$, then in any S's subspace $U = \{u_0, u_1, \cdots, u_l\}$ where $(0 \leq u_i \leq k)$, L's projection is $a_{u_0}x_{u_0} + a_{u_1}x_{u_1} + \cdots + a_{u_l}x_{u_l} + b = 0$, which is a line. ■

Lemma 6 *(Projection Property 2)*
If L is a curve in k dimensional space S, and L's projection on any two dimensional subspace of S is a line, then L is a line in S.

Proof: Without loss of generality assume that L's projection on any two dimensional subspace $\{i, j\}$ of S is $a_ix_i + a_jx_j + b_{ij} = 0$, then L can be expressed as
$\sum_{0 \leq i,j \leq k \text{ and } i \neq j}(a_ix_i + a_jx_j + b_{ij}) = 0$ in space S, which is also a line equation. ■

Start from this clustering problem definition, we can continue to study the subspace patterns for the real-valued dataset in further depth. Figure 6.1 gives (a) an example dataset and (b) two overlapping *F-cluster*s which are visually clear after some rows/columns permutations. For each column pair of a cluster, there is a line in that two dimensional subspace that contains all the points of cluster. For example for column pair d_0 and d_6, the line is: $2d_0 - d_6 + 3 = 0$. In real datasets, small tolerance threshold ε is allowed, which confines the maximum distance of the point to the line on any two dimensional projected subspace.

6.1.2 Algorithm Design

We will use an enumerating algorithm to find all those subspace lines. Basically the algorithm can be composed of two main steps: finding the lines between all column pairs and constructing a multigraph and clique mining on the graph to get the final *F-cluster*s.

First according to Lemma 6 we need to find all the lines for any two dimensional subspace of the the whole dataset space. We know that a two dimensional line can be expressed as $ax + by + c = 0$, which has two degrees of freedom (DOF). To solve this problem, we can proceed as follows. Since two points completely specify a line, we can enumerate each point pair for any two dimensional subspace to decide if the resulting line approximately contains (i.e., the point line distance is less than or equal to ε) enough points, i.e., whether the number of points is greater than the minimum support min_sup. A possible optimization we need to consider is that if the point pairs which have already been contained in some previous enumerated line need to be enumerated again.

After having got all the two dimensional lines, the next step is to combine them together to form biclusters. Like MICROCLUSTER, if we treat each column as a vertex, and each two dimensional line as an edge, they form a graph. Then we can do clique mining on the graph; each maximal clique will correspond to a *F-cluster* as defined before. Please note there may exist multiple edges between any two vertices. During the cluster mining process, we will do the pruning using three basic constraints: maximum point to line distance: ε, minimum point number min_sup and minimum dimension number min_dim which controls the depth of the enumeration tree.

6.1.3 Experiments Design

We will use both synthetic and real datasets to test the *F-cluster* idea. The synthetic datasets will be used to test the scalability of different input parameters, such as the dataset size, the cluster number, the minimum cluster size, the fluctuation (noise level) of a dataset, etc.. The synthetic code generation will conform to the *F-cluster* definition and allow cluster overlap. At the same time, noise will be embedded in the synthetic clusters to simulate real data phenomenon.

We will also try to find the significant or interesting clusters which may be missed by other bicluster definitions. We will try to use different kinds of real datasets to do the testing, such as gene microarray expression data, business market-basket data, census data, and so on.

6.2 Mining Boolean Patterns from Binary-valued Datasets

For the boolean pattern mining from the binary-valued binary-valued datasets, there are still many interesting issues to consider. The first one involves the effective handling of negative literals without being overwhelmed by dataset density. The second issue is to push tautological considerations deeper into the mining algorithm by designing new pruning operators. In addition, we are led to the general challenge of, given a general propositional reasoning framework, mining only the simplest boolean expressions necessary for inference in that framework.

The boolean pattern mining can also be regarded as the redescription problem. We also plan to apply the boolean pattern mining to the gene ontology (GO) hierarchies (of different organisms or different ontologies). The GO project (www.geneontology.org) aims at developing three structured, controlled vocabularies (ontologies) that describe gene products in terms of their associated biological processes, cellular components and molecular functions in a species-independent manner. GO has three hierarchies: PROCESS, FUNCTION and LOCATION, each of which has a set of GO terms and the corresponding genes, and maps the genes that share same process, function or cellular location respectively.

GO hierarchy can be viewed as a directed acyclic graph (DAG) for each category (function, process, location). If we let each term be an item and each gene be a transac-

tion, then we can create a transaction$\times$item dataset. If a gene belongs to a term or any descendant of a term, we mark their cross position as true, otherwise we mark it as false. This dataset can be mined to get some useful information about gene ontology. First we do the closed set mining on the dataset, where every closed set denotes a rooted subtree of GO. Actually we can prove that all rooted subtrees of GO are represented in the set of closed sets. Second we identify a set of rooted subtrees from all rooted subtrees of GO, such that they form a partition of the GO hierarchy. Third, if we apply the two steps above on two different GO hierarchies, we will get two partitions. Then we will establish one-to-one mapping between these two partitions. Mapping between GO categories can reveal important information. For example, we may map PROCESS concepts to LOCATION concepts to find out, say, which genes share similar processes and locations. Another example is that we can compare GO concepts across organisms, to find out, say, how human genes relate to mouse genes. The key point here is how to choose the closed sets we mined to partition the two GO DAGs so as to construct a one-to-one mapping between them. This idea can also be expanded to associate multiple GO hierarchies.

LITERATURE CITED

[1] C. C. Aggarwal, C. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park. Fast algorithms for projected clustering. In *ACM SIGMOD Conference on Management of Data*, 1999.

[2] C. C. Aggarwal and P. S. Yu. Finding generalized projected clusters in high dimensional spaces. In *ACM SIGMOD Conference on Management of Data*, 2000.

[3] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *ACM SIGMOD Conference on Management of Data*, 1998.

[4] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. *Advances in KDD, AAAI Press*, pages 307-328, 1996.

[5] M.-L. Antonie and O. R. Zaiane. Mining positive and negative association rules: an approach for confined rules. In *European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2004.

[6] Z. Bar-Joseph. Analyzing time series gene expression data. *Bioinformatics*, 20(16):2493-2503, 2004.

[7] Y. Bastide, N. Pasquier, R. Taouil, G. Stumme, and L. Lakhal. Mining minimal non-redundant association rules using frequent closed itemsets. In *International Conference on Computational Logic*, 2000.

[8] R. J. Bayardo and R. Agrawal. Mining the most interesting rules. In *ACM SIGKDD Internatioal Conference on Knowledge Discovery and Data Mining*, 1999.

[9] A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini. Discovering local structure in gene expression data: the order-preserving submatrix problem. In *6th Annual International Conference on Computational Biology, RECOMB*, 2002.

[10] A. Ben-Dor, R. Shamir, and Z. Yakhini. Clustering gene expression patterns. In *3rd Annual International Conference on Computational Biology, RECOMB*, 1999.

[11] N. H. Bshouty. Exact learning boolean functions via the monotone theory. *Information and Computation*, 123(1):146-153, 1995.

[12] T. Calders and B. Goethals. Minimal k-free representations of frequent sets. In *European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2003.

[13] Y. Cheng and G. M. Church. Biclustering of expression data. In *8th International Conference on Intelligent Systems for Molecular Biology*, pages 93-103, 2000.

[14] M. B. Eisen, P.T. Spellman, P.O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Science*, USA, 95(25):14863-14868, 1998.

[15] S. Erdal, O. Ozturk, D. Armbruster, H. Ferhatosmanoglu, and W. C. Ray. A time series analysis of microarray data. In *4th IEEE International Symposium on Bioinformatics and Bioengineering*, May 2004.

[16] J. Feng, P. E. Barbano, and B. Mishra. Time-frequency feature detection for timecourse microarray data. In *2004 ACM Symposium on Applied Computing*, 2004.

[17] V. Filkov, S. Skiena, and J. Zhi. Analysis techniques for microarray time-series data. In *5th Annual International Conference on Computational Biology, RECOMB*, 2001.

[18] J. H. Friedman and J. J. Meulman. Clustering objects on subsets of attributes. *Journal of the Royal Statistical Society Series B*, 66(4):815, 2004.

[19] B. Ganter and R. Wille. Formal concept analysis. *Mathematical Foundations*, Springer-Verlag, 1999.

[20] B. Goethals and M. J. Zaki. Advances in frequent itemset mining implementations: report on FIMI'03. *SIGKDD Explorations*, 6(1):109-117, 2003.

[21] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharma. Discovering all most specific sentences. *ACM Transactions on Database Systems*, 28(2):140-174, 2003.

[22] E. Hartuv, A. Schmitt, J. Lange, S. Meier-Ewert, H. Lehrach, and R. Shamir. An algorithm for clustering cDNAs for gene expression analysis. In *3rd Annual International Conference on Computational Biology, RECOMB*, 1999.

[23] D. Jiang, J. Pei, M. Ramanathany, C. Tang, and A. Zhang. Mining coherent gene clusters from gene-sample-time microarray data. In *10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004. *International Journal of Computer and Information Science*, 5(3), 1976.

[24] M. Kryszkiewicz. Concise representation of frequent patterns based on disjunction-free generators. In *1st IEEE International Conference on Data Mining, ICDM*, 2001.

[25] J. Liu and W. Wang. OP-Cluster: clustering by tendency in high dimensional spaces. In *3rd IEEE International Conference on Data Mining, ICDM*, pages 187-194, 2003.

[26] S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):24-45, 2004.

[27] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203-226, 1982.

[28] C. S. Moller, F. Klawonn, K. H. Cho, H. Yin, and O. Wolkenhauer. Clustering of unevenly sampled gene expression time-series data. *Fuzzy Sets and Systems*, 2004.

[29] T. M. Murali and S. Kasif. Extracting conserved gene expression motifs from gene expression data. In *Pacific Symposium on Biocomputing*, 2003.

[30] A. A. Nanavati, K. P. Chitrapura, S. Joshi, and R. Krishnapuram. Association rule mining: mining generalised disjunctive association rules. In *International Conference on Information and Knowledge Management*, 2001.

[31] L. Parida and N. Ramakrishnan. Redescription mining: structure theory and algorithms. In *20th National Conference on Artificial Intelligence*, July 2005.

[32] J. Pei, X. Zhang, M. Cho, H. Wang, and P. S. Yu. Maple: A fast algorithm for maximal pattern-based clustering. In *3rd IEEE International Conference on Data Mining, ICDM*, pages 19-22, 2003.

[33] C. M. Procopiuc, M. Jones, P. K. Agarwal, and T. M. Murali. A Monte Carlo algorithm for fast projective clustering. In *ACM SIGMOD International Conference on Management of Data*, 2002.

[34] N. Ramakrishnan, D. Kumar, B. Mishra, M. Potts, and R. F. Helm. Turning cartwheels: an alternating algorithm for mining redescriptions. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug 2004.

[35] M. F. Ramoni, P. Sebastiani, and I. S. Kohane. Cluster analysis of gene expression dynamics. *Proceedings of the National Academy of Sciences*, USA, 99(14):9121-9126, July 2002.

[36] A. Savasere, E. Omiecinski, and S. B. Navathe. Mining for strong negative associations in a large database of customer transactions. In *International Conference on Data Engineering, ICDE*, 1998.

[37] A. Savinov. Mining possibilistic set-valued rules by generating prime disjunctions. In *European Conference on Principles and Practice of Knowledge Discovery in Databases*, 1999.

[38] R. Sharan and R. Shamir. CLICK: a clustering algorithm with applications to gene expression analysis. In *International Conference on Intelligent Systems for Molecular Biology*, 2000.

[39] Y. Shima, S. Mitsuishi, K. Hirata, and M. Harao. Extracting minimal and closed monotone DNF formulas. In *International Conference on Discovery Science*, 2004.

[40] P. T. Spellman, G. Sherlock, M. Q. Zhang, V. R. Iyer, K. Anders, M. B. Eisen, P. O. Brown, D. Botstein, and B. Futcher. Comprehensive identification of cell cycle-regulated genes of the yeast saccharomyces cerevisiae by microarray hybridization. *Molecular Biology of the Cell*, 9(12):3273-3297, December 1998.

[41] P. Tamayo, D. Slonim, J. Mesirov, Q. Zhu, S. Kitareewan, E. Dmitrovsky, E. S. Lander, and T. R. Golub. Interpreting patterns of gene expression with self-organizing maps: Methods and application to hematopoietic differentiation. *Proceeedings of the National Academy of Science*, USA, 96(6):2907-2912, 1999.

[42] A. Tanay, R. Sharan, and R. Shamir. Discovering statistically significant biclusters in gene expression data. *Bioinformatics*, 18(Suppl.1):S136-S144, 2002.

[43] C. Tang, L. Zhang, A. Zhang, and M. Ramanathan. Interrelated two-way clustering: an unsupervised approach for gene expression data analysis. In *2nd IEEE International Symposium on Bioinformatics and Bioengineering, BIBE*, 2001.

[44] H. Wang, W. Wang, J. Yang, and P. S. Yu. Clustering by pattern similarity in large data sets. In *ACM SIGMOD International Conference on Management of Data*, 2002.

[45] X. Wu, C. Zhang and S. Zhang. Efficient mining of both positive and negative association rules. *ACM Transactions on Information Systems*, 22(3):381-405, 2004.

[46] E. P. Xing and R. M. Karp. CLIFF: clustering high-dim microarray data via iterative feature filtering using normalized cuts. *Bioinformatics*, 17(Suppl.1):S306-S315, 2001.

[47] Y. Xu, V. Olman, and D. Xu. Clustering gene expression data using a graph-theoretic approach: an application of minimum spanning trees. *Bioinformatics*, 18(4):536-545, 2002.

[48] J. Yang, W. Wang, H. Wang, and P. S. Yu. δ-clusters: Capturing subspace correlation in a large data set. In *18th International Conference on Data Engineering, ICDE*, 2002.

[49] K. Y. Yeung and W. L. Ruzzo. Principal component analysis for clustering gene expression data. *Bioinformatics*, 17(9):763-774, 2001.

[50] M. J. Zaki. Mining non-redundant association rules. *Data Mining and Knowledge Discovery*, 9(3):223-248, 2004.

[51] M. J. Zaki and C.-J. Hsiao. Efficient algorithms for mining closed itemsets and their lattice structure. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):462-478, 2005.

[52] M. J. Zaki and N. Ramakrishnan. Reasoning about sets using redescription mining. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug 2005.

[53] L. Zhao and M. J. Zaki. MicroCluster: efficient deterministic biclustering of microarray data. *IEEE Intelligent Systems*, 20(6):40-49, 2005.

[54] L. Zhao and M. J. Zaki. TriCluster: an effective algorithm for mining coherent clusters in 3D microarray data. In *ACM SIGMOD International Conference on Management of Data*, 694-705, June 2005.

[55] L. Zhao, M. J. Zaki and N. Ramakrishnan. BLOSOM: A framework for mining boolean expressions. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug 2006.

[56] J. Buhler. Anatomy of a comparative gene expression study. http://www.cs.wustl.edu/˜jbuhler/research/array/

[57] The gene ontology. http://www.geneontology.org