An Efficient Algorithm for Mining Coherent Patterns from Heterogeneous Microarrays

Xiang Zhang and Wei Wang
Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, NC 27599, USA
{xiang, weiwang}@cs.unc.edu

Abstract

DNA microarray techniques present a novel way for geneticists to monitor interactions among tens of thousands of genes simultaneously, and have become standard lab routines in gene discovery, disease diagnosis, and drug design. There has been extensive research on coherent subspace clustering of gene expressions measured under consistent experimental settings. This implies that all experiments are run using the same batch of microarray chips with similar characteristics of noise. Algorithms developed under this assumption may not be applicable for analyzing data collected from heterogeneous settings, where the set of genes being monitored may be different and expression levels may be not directly comparable even for the same gene. In this paper, we propose a model, F-cluster, for mining subspace coherent patterns from heterogeneous gene expression data, which is shown effective for revealing truthful patterns and reducing spurious ones. We also develop an efficient and scalable hybrid approach that combines gene-pair based and sample-pair based pruning to generate F-clusters from multiple gene expression matrices simultaneously. The experimental results demonstrate that our model can discover significant clusters that may not be identified by previous models.

1 Introduction

Recent advances in microarray technology have made large amounts of gene expression data available from a variety of different experimental settings. Studies have shown that analyzing microarray data is essential for understanding the gene functions[25], gene regulation[10], cellular process[9] and subtypes of cells [4].

Clustering is a popular method used to analyze microarrays. By grouping the genes that exhibit similar patterns, biologists can gain valuable insights in genetics. Originally developed for general data mining applications, subspace clustering algorithms [3] have demonstrated their utility in gene expression analysis. A special class of subspace clustering algorithms called bi-clustering can effectively identify meaningful gene groups [27, 28]. This approach is motivated by the fact that a group of genes may only have coherent expressions under a subset of the experimental conditions. By finding shifting and/or scaling patterns, bi-clustering algorithms can identify co-regulated genes whose expressions differ in value but are highly correlated. Tricluster [29] extends the model to the temporal domain. It finds clusters in gene-sample-time space. The algorithm first finds the bi-clusters in the matrix at each time point, then combines those bi-clusters sharing coherent patterns along the time dimension to generate triclusters.

Although previous coherent subspace clustering methods have demonstrated their usefulness, all of these methods focus on analyzing expression data generated by a single microarray technique. However, there have been several microarray techniques using fundamentally different mechanisms to measure gene expression levels. Some widely used techniques include Affymetrix oligonucleotide microarrays [16], cDNA microarrays [20], and serial analysis of gene expression (SAGE) [26]. The expression levels reported by different techniques are not necessarily comparable with each other. Since previously proposed subspace clustering methods were developed under the assumption that the gene expressions are measured under consistent experimental settings, they can not be readily applied to analyze data generated by different techniques.

To overcome the limitation of previous methods, we propose a more general model for mining coherent subspace clusters from multiple microarrays that may be generated by different techniques.

1.1 Motivation

Methods for subspace clustering heterogeneous data are needed and will benefit wide applications.

First, gene clusters preserved in multiple microarrays generated by different techniques are more likely to be biologically relevant to each other than those preserved in only a single matrix. Previous study [12] has shown that gene

19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)
0-7695-2868-6/07 $25.00 © 2007 IEEE

pairs having similar expression patterns across multiple microarray platforms have higher correlation with functional terms classified in Gene Ontology (GO) than those only showing similar patterns in a single microarray.

Second, data heterogeneity exists in other bioinformatics applications beyond gene expression analysis. For example, we want to predict whether a gene is up or down regulated in a particular experiment [17]. The decision rules can be learned based on (1) the presence of binding site subsequences (motifs) in the gene's regulatory region and (2) the expression levels of regulators, such as transcription factors, in the experiment. The promoter sequences can be modelled as a $\{0, 1\}$ matrix encoding the presence or absence of a certain motif in a sequence. Now the task is to find clusters of genes that are preserved in the subspace of promoter sequence matrix, gene expression profile, and regulator expression profile.

Finally, this problem is also observed in other application domains. For example, in transaction databases, if a group of customers show similar transactional patterns across different databases, the patterns are unlikely to be coincidence.

1.2 Matrix Concatenation does not Resolve Heterogeneity

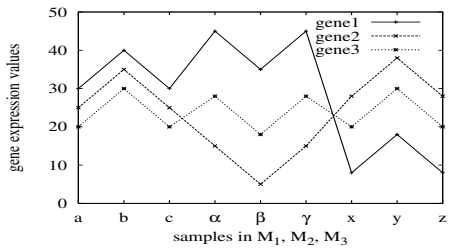
A natural approach to subspace clustering of the expression levels from multiple arrays would be to simply concatenate the samples from all arrays to create a single subspace clustering problem to be solved by existing methods like [27, 28]. However, the heterogeneity of the arrays disqualifies this approach. While we expect that co-expressed genes exhibit related expression levels in each individual array (or at least in many of the arrays), we can not expect that their expression levels will be related the same way across multiple arrays.

As a result subspace clustering of the concatenated matrices may fail to find a cluster that has sufficient support in each matrix individually. This is illustrated in Figure 1. Genes g_1, g_2, and g_3 are a coherent cluster on the three samples in each of the matrices M_1, M_2 and M_3 considered in isolation. Note that their expression levels relate to each other differently in different matrices (e.g., they may have different scaling and shifting factors). When the matrices are combined, the largest coherent cluster (subject to uniform scaling and shifting) only has 3 samples, which may be insufficient to consider the genes to be co-expressed within the combined array.

It is not sufficient to lower the minimum number of samples for a cluster in the combined array approach. This can introduce spurious subspace clusters as illustrated in Figure 2. In this case there are 2 matrices, M_1 and M_2. Each matrix has 2 samples,and the cluster of genes $\{g_1, g_2\}$ is a coherent cluster in each matrix. In the combined matrix we find that $\{g_1, g_2\}$ is supported by two samples, but in addition cluster $\{g_2, g_3\}$ is supported by two samples as well. Note that $\{g_2, g_3\}$ is a spurious cluster since it is not preserved in either M_1 or M_2.

	M_1			M_2			M_3		
	a	b	c	α	β	γ	x	y	z
g_1	30	40	30	45	35	45	8	18	8
g_2	25	35	25	15	5	15	28	38	28
g_3	20	30	20	28	18	28	20	30	20

(a) Expression values of g_1, g_2, g_3

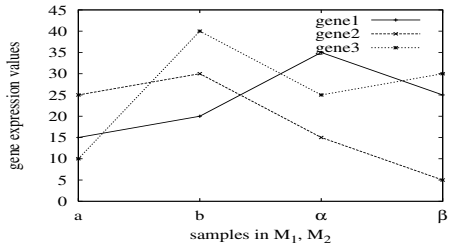


(b) Expression patterns of g_1, g_2, g_3

Figure 1. Example 1

	M_1		M_2	
	a	b	α	β
g_1	15	20	35	25
g_2	25	30	15	5
g_3	10	40	25	30

(a) Expression values of g_1, g_2, g_3



(b) Expression patterns of g_1, g_2, g_3

Figure 2. Example 2

The fundamental reason why concatenating matrices does not work is that expression levels measured by different microarray techniques are not necessarily comparable with each other. They should not be treated in the same manner as those collected in a homogeneous setting (i.e., data in the same matrix). Instead this problem must be solved by identifying clusters in the arrays individually.

1.3 Goal

To overcome the limitations of previous models, we propose the model of *F-cluster* (Frequent cluster), which mines genes co-expressed in subsets of samples within multiple

microarrays. Intuitively, an F-cluster is a group of genes that show similar expression patterns in some subspaces (of samples) in at least a (user-specified) number of matrices. For example, as shown in Figure 1, the gene group $\{g_1, g_2, g_3\}$ is an F-cluster which is preserved in matrices M_1, M_2, and M_3.

F-clusters can be viewed as *consensus* subspace clusters in multiple microarray matrices. To mine the F-clusters, in a straightforward manner, we can adopt a 2-step approach. In the first step, we mine subspace clusters in each matrix. Then in the second step, we find F-clusters from these subspace clusters. However, there are usually thousands to millions of subspace clusters (even if we only keep maximal subspace clusters) in a single matrix, so this 2-step approach can become very inefficient. To efficiently find F-clusters, we propose a method that simultaneously mines multiple matrices.

Previous methods for mining subspace clusters (or biclusters) in a single matrix can be classified into 2 categories: sample based enumeration and gene based enumeration. These methods explore the search space by enumerating either the combinations of genes or the combinations of samples. These two approaches, as discussed in detail in Section 4, are inefficient to find F-clusters. The gene based enumeration will generate a huge number of candidate F-clusters and the sample based enumeration is inefficient to validate candidates. We propose a hybrid approach to mine the F-clusters. This approach generates candidate F-clusters based on samples and validates them using gene-pairs. Our method incorporates effective pruning strategies that can avoid unnecessary candidate validations.

1.4 Contributions

We propose the F-cluster model for mining coherent subspace clusters in multiple gene expression data generated by different microarray techniques. We develop an efficient hybrid approach that combines gene-pair based and sample-pair based pruning to generate F-clusters from multiple gene expression matrices simultaneously. Our experimental results show that the F-cluster model is able to capture significant gene clusters which may not be found using previous models.

2 Related work

Recent studies have shown that microarray clustering algorithms are able to discover patterns with significant biological meanings. For example, clustering genes with similar expression patterns helps to infer unknown functions of genes [10] and gives rise to hypotheses regarding the mechanism of the transcriptional regulatory network [9]. Existing clustering algorithms applicable to microarray data analysis can be classified into either *full space clustering algorithms* (e.g., [10, 11, 23, 25]), or *subspace clustering algorithms* (e.g., [1, 2, 3, 14, 27, 28, 29]). In this section, we briefly survey some representative methods.

Many classical clustering algorithms are full-space clustering algorithms. Some widely used approaches include K-means algorithms [25], hierarchical algorithms [5, 10], and graph-theoretic approaches [7, 21]. A recent approach [13] employed a graph mining algorithm to find co-expressed genes in multiple microarrays. Each microarray is modelled as an unweighted and undirected graph. Two genes are connected with an edge if they show high correlation across all samples in a microarray. Coherent dense subgraphs are then mined from these graphs and are used to model co-expressed gene groups. A common limitation of these full-space clustering methods is that they are only able to identify clusters in the space formed by all features (i.e., all experimental samples in a microarray). However, a group of genes with similar functions may only be co-expressed under a subset of samples [8, 14, 27], which motivated the development of subspace clustering methods.

Subspace clustering was first studied in [1, 2, 3] to discover clusters of objects that share similar values in a subset of dimensions. A distance function, such as Euclidean distance or cosine distance, is usually adopted as the similarity measure. This allows the utilization of spatial distribution of objects as an effective pruning mechanism, but at the same time, limits the gene expression patterns that can be discovered. Recent bi-clustering algorithms [8, 14, 27] overcome this limitation by finding coherent patterns allowing arbitrary shifting and scaling by a constant (but unknown) amount. The algorithms not only identify the clusters of genes, but also show in which subspaces the genes form clusters.

Some recent work [28, 29] studied the problem of mining gene-sample-time microarray data. Tricluster [29] introduced the problem of finding clusters in the microarray data generated at synchronized time points for the same set of samples. For example, $(\{g_0, g_2, g_7\} \times \{s_1, s_3, s_9\} \times \{t_2, t_6\})$ can be interpreted as "at time points t_2 and t_6, genes $\{g_0, g_2, g_7\}$ show similar expression patterns in samples s_1, s_2 and s_3". Tricluster algorithm builds on bi-cluster subroutines. The algorithm first mines bi-clusters in each matrix by searching for maximal cliques in the weighted range graph which is a compact representation of similar expression ranges in the dataset between any pair of samples. Then it finds the triclusters by combining bi-clusters along the time dimension. In [28], a similar 2-step approach was taken to mine clusters that exhibit shifting and scaling patterns in gene-sample-time microarray data. Despite differences in cluster definitions and mining algorithms, a common limitation of these methods is that they assume the same set of samples are examined over time and the expression measurements are compatible with each other. Thus they are not readily applicable to heterogenous microarray

datasets. In this paper, we do not require homogeneity. Our approach will be applicable to both homogeneous and heterogenous data.

Some statistical/probablistic methods have been proposed for the study of heterogenous biological data. In [24], the authors proposed a statistical algorithm that models heterogenous datasets as a weighted bipartite graph and applies heuristics to identify subsets of genes that jointly respond across a subset of samples. However, this method cannot identify shifting/scaling coherent patterns which have been demonstrated useful in discovering highly co-regulated genes [27, 28]. Moveover, the heuristic methods do not guarantee completeness and optimality.

3 The F-Cluster Model

This section presents the formal definition of the F-cluster model for mining coherent subspace clusters in multiple microarray matrices. We also give a brief comparison between our model and previous models. At the end of this section, we analyze the complexity of the search space of F-clusters.

3.1 Definitions and Problem Statement

Let $G = \{g_1, g_2, \ldots, g_I\}$ be a set of I genes and $M = \{M_1, M_2, \ldots, M_K\}$ be a set of K microarray data matrices for G. Each matrix $M_k = G \times S_k = \{v_{ij_k}\}$ $(1 \leq k \leq K)$ is an $I \times J_k$ matrix of real numbers, where S_k is the set of samples in matrix M_k. $J_k = |S_k|$ denotes the number of samples in M_k and v_{ij_k} denotes the expression level of gene $g_i (1 \leq i \leq I)$ in sample $s_{j_k} (1 \leq j_k \leq J_k)$. For example, Tables 1(a) to 1(c) show a dataset of 3 microarray matrices for 6 genes. Note that each matrix may have independent sample set. There may be no direct correspondence between samples in different matrices. In principle, the gene sets used in different matrices may not be identical. In practice, there is substantial overlap between the gene sets measured by different microarray techniques. Therefore, we focus on the set of genes that are common to all types of microarray chips. The proposed model and algorithms can be easily generalized to accommodate genes that are absent on some microarray chips.

Informally speaking, our goal is to find the complete set of gene clusters, each of which appears in at least δ $(1 \leq \delta \leq K)$ matrices. The cluster in each matrix can be defined in many different ways. In this paper, we adopt the pattern-based (subspace)[1] cluster definition [27].

A *cluster* C in matrix M_k is a sub-matrix of M_k: $C = X \times Y_k = \{v_{ij_k}\}$, where $X \subseteq G$ and $Y_k \subseteq S_k$, such that for any 2×2 sub-matrix $\begin{pmatrix} v_{ax} & v_{ay} \\ v_{bx} & v_{by} \end{pmatrix}$ of C, we have $||(v_{ax} - v_{bx}) - (v_{ay} - v_{by})|| \leq \epsilon$, where ϵ is the user-specified cluster threshold. Intuitively, ϵ describes the allowed disparity between the expression patterns of two genes in two samples. The smaller the value of ϵ, the more similar the expression patterns of the two genes. The *size* of cluster C is defined as the number of genes $|X|$ in C. For example, $(\{g_1, g_2\} \times \{s_{1_1}, s_{2_1}\}_{M_1})$ represents a cluster of size 2 in matrix M_1, whose genes $\{g_1, g_2\}$ are co-expressed in samples $\{s_{1_1}, s_{2_1}\}$.

An ***F-cluster*** is of form $FC = X \times \bar{Y}$, where $X \subseteq G$ is a set of genes, $\bar{Y} = \{Y_k\}$ is a set of sample sets. $Y_k \subseteq S_k$ is a set of samples in matrix M_k and $C = X \times Y_k$ is a cluster[2] in M_k. M_k is a *supporting matrix* of FC. We refer to the number of genes as the *size* of FC and the number of matrices supporting FC (i.e., $|\bar{Y}|$) as the *support* of FC. For example, $(\{g_1, g_2, g_3\} \times \{\{s_{1_1}, s_{4_1}, s_{5_1}\}_{M_1}, \{s_{2_2}, s_{6_2}\}_{M_2}\})$ is a size 3 F-cluster in the dataset shown in Table 1(a) to 1(c), with supporting matrices M_1 and M_2.

	s_{1_1}	s_{2_1}	s_{3_1}	s_{4_1}	s_{5_1}	s_{6_1}	s_{7_1}	s_{8_1}	s_{9_1}
g_1	1	12	1	1	1	17	4	62	42
g_2	2	23	2	2	2	32	18	21	31
g_3	3	34	79	3	3	73	6	52	37
g_4	5	43	58	5	5	21	32	49	72
g_5	2	72	33	2	41	77	56	53	19
g_6	3	53	28	83	3	33	47	26	52

(a) Matrix M_1

	s_{1_2}	s_{2_2}	s_{3_2}	s_{4_2}	s_{5_2}	s_{6_2}	s_{7_2}
g_1	7	3	8	5	34	3	2
g_2	7	1	8	3	2	1	66
g_3	7	2	8	52	3	2	1
g_4	7	78	8	25	92	53	37
g_5	71	3	33	58	19	3	25
g_6	79	13	41	27	39	62	21

(b) Matrix M_2

	s_{1_3}	s_{2_3}	s_{3_3}	s_{4_3}	s_{5_3}	s_{6_3}	s_{7_3}	s_{8_3}
g_1	4	6	8	54	7	6	23	13
g_2	1	3	5	3	62	48	4	62
g_3	52	2	29	2	3	2	3	31
g_4	38	82	37	62	22	14	27	79
g_5	19	31	97	46	25	38	62	21
g_6	72	39	55	34	62	72	30	43

(c) Matrix M_3

Genes	samples (M_1)	samples (M_2)	samples (M_3)
g_1, g_2, g_3	$s_{1_1}, s_{4_1}, s_{5_1}$	s_{2_2}, s_{6_2}	$\emptyset$

(d) An F-cluster ($\delta = 2, min_g = 2, min_s = 2, \epsilon = 0$)

Table 1. (a)-(c): A dataset containing 3 microarray matrices (d): An F-cluster in the dataset

[1] In the following discussion, we sometime omit the term *subspace* for brevity.

[2] We distinguish the concepts of cluster and F-cluster. A cluster is a set of genes that have similar expression patterns in a single matrix. An F-cluster is a set of genes that have similar patterns in at least δ matrices.

Given two F-clusters $FC_1 = X_1 \times \bar{Y_1}, FC_2 = X_2 \times \bar{Y_2}$, if the two gene sets satisfy $X_1 \subseteq X_2$, we call FC_1 a *sub-cluster* of FC_2 and FC_2 a *super-cluster* of FC_1. A *maximal F-cluster* is an F-cluster whose super-clusters have support less than δ. Since our focus is on mining gene clusters that show similar expression patterns across different matrices, we define sub-cluster in terms of gene sets. For example, $(\{g_1, g_2\} \times \{\{s_{1_1}, s_{3_1}, s_{4_1}, s_{5_1}\}_{M_1}, \{s_{2_2}, s_{4_2}, s_{6_2}\}_{M_2}, \{s_{1_3}, s_{2_3}, s_{3_3}\}_{M_3}\})$ is a sub-cluster of $(\{g_1, g_2, g_3\} \times \{\{s_{1_1}, s_{4_1}, s_{5_1}\}_{M_1}, \{s_{2_2}, s_{6_2}\}_{M_2}\})$. Note that this is different from the definition in [27] which consider both gene set and sample set[3].

Given user-specified thresholds δ, min_g, and min_s, our objective is to find all maximal F-clusters which have at least min_g genes, at least δ supporting matrices, and at least min_s samples in each supporting matrix. In practice, a user may wish to specify different thresholds for different matrices. Our algorithms can be extended to handle this scenario. In the remainder of this paper, we assume that the same threshold is used for all matrices for simplicity of explanation.

3.2 Comparison with Previous Models

Previous coherent subspace clustering methods focus on mining data generated under homogeneous settings. Our model, on the other hand, can be applied to microarray data collected from heterogeneous settings. Since experimental settings used in different microarray techniques and the samples may be different and incompatible, we do not require the genes in an F-cluster have same expression patterns in different microarrays. As shown in Figure 1, the F-cluster $\{g_1, g_2, g_3\}$ is preserved in M_1, M_2, and M_3 separately. But the expression patterns are not preserved across matrices. (There are cross-overs between matrices as shown in Figure 1.) Our experimental results show that our model can identify clusters that cannot be found by previous methods.

3.3 Exponential Search Space

There can be as many as 2^I potential gene clusters in each matrix, where I typically ranges from thousands to tens of thousands. Identifying those supported by at least δ out of K matrices requires intelligent pruning strategies to confine the search space. In this paper, we propose a method that mines the F-clusters simultaneously from all the matrices. This method enumerates the candidate F-clusters over the small set of sample-pairs, and validate the candidates by the gene-pairs. We employ four effective pruning strategies to avoid unnecessary candidate validations.

[3] In [27], a cluster $C_1 = A \times B$ is a sub-cluster of $C_2 = C \times D$ if and only if $A \subseteq C$ and $B \subseteq D$.

Genes	samples (M_1)	samples (M_2)	samples (M_3)
g_1, g_2	$s_{1_1}, s_{3_1}, s_{4_1}, s_{5_1}$	$s_{2_2}, s_{4_2}, s_{6_2}$	$s_{1_3}, s_{2_3}, s_{3_3}$
g_2, g_3	$s_{1_1}, s_{4_1}, s_{5_1}$	$s_{2_2}, s_{5_2}, s_{6_2}$	$s_{2_3}, s_{4_3}, s_{7_3}$
g_1, g_3	$s_{1_1}, s_{4_1}, s_{5_1}, s_{7_1}$	$s_{2_2}, s_{6_2}, s_{7_2}$	$s_{2_3}, s_{5_3}, s_{6_3}$

Table 2. Frequent gene-pair MDSs

4 The Algorithms

In this section, we discuss two algorithms which mine F-clusters simultaneously from multiple matrices. The first algorithm is a straightforward approach which applies Apriori property to generate candidates in a bottom-up fashion. We then present a hybrid algorithm that integrates sample based and gene based enumeration and incorporates effective pruning strategies. Before formally presenting the algorithms, we first introduce some basic concepts.

4.1 Frequent MDSs

We adopt the notion of *Maximum Dimension Set (MDS)* introduced in [27]. In matrix M_k, a cluster $clu_1 = (\{g_1, g_2\} \times Y_k)$ is a *gene-pair MDS* (Maximum Dimension Set), if there does not exist a cluster $clu_1^{'} = (\{g_1, g_2\} \times Y_k^{'})$ such that $Y_k \subset Y_k^{'}$. A cluster $clu_2 = (X \times \{s_{1_k}, s_{2_k}\})$ is a *sample-pair MDS*, if there does not exist a cluster $clu_2^{'} = (X^{'} \times \{s_{1_k}, s_{2_k}\})$ such that $X \subset X^{'}$. Intuitively, a gene-pair or sample-pair MDS in a matrix is a maximal cluster containing only two genes or two samples. Since MDSs can be computed efficiently [27] for any given matrix, our algorithms utilize MDSs to mine F-clusters. Please refer to [27] for the method of finding MDSs in one matrix.

A *frequent gene-pair MDS* is an F-cluster of two genes, which contains a set of gene-pair MDSs from at least δ supporting matrices (for the same pair of genes). Table 2 shows an example of frequent gene-pair MDS in the dataset shown in Table 1(a) to 1(c) with $\epsilon = 0$, $min_s = 3$ and $\delta = 3$. The following strategies can be used for initial pruning. The proof of the correctness is straightforward and thus omitted here.

Prune MDSs: (1) A gene must appear in at least $\binom{min_s}{2}$ sample-pair MDSs in at least δ matrices, and in at least $(min_g - 1)$ frequent gene-pair MDSs. (2) A sample must appear in at least $\binom{min_g}{2}$ frequent gene-pair MDSs, and in at least $(min_s - 1)$ sample-pair MDSs.

4.2 The Basic F-cluster Miner

In this section, we show a basic approach to find the F-clusters using the Apriori property: *the support of an F-cluster is always smaller than or equal to that of its sub-clusters*. The detailed algorithm is shown in Algorithm 1. It first finds frequent gene-pair MDSs and sample-pair MDSs (line 1), and applies the above two strategies for initial pruning (line 2). The algorithm then proceeds in iterations. In

Algorithm 1: The basic F-cluster Miner

Input: a set of gene expression matrices $\{M_1, M_2, \ldots, M_K\}$, cluster threshold ϵ, minimum number of genes min_g, minimum number of samples min_s, minimum support δ.
Output: the complete set of F-clusters.

```
1  Find frequent gene-pair MDSs and sample-pair MDSs;
2  Prune frequent gene-pair MDSs and sample-pair MDSs;
3  FC_2 ← frequent gene-pair MDSs;
4  for (l = 3; FC_{l-1} ≠ ∅; l++) do
5      CFC_l ← GenerateCandidate(FC_{l-1});
6      for c ∈ CFC_l do
7          if SubsetCheck (c, FC_{l-1}) is true then
8              if Validate (c, FC_{l-1}) is true then
9                  FC_l ← c;
10             end
11         end
12     end
13 end
14 Return ⋃_l FC_l(l ≥ min_g).
```

each iteration, the algorithm generates larger candidates by extending the current F-clusters to include one more gene (line 5). For each candidate c of size l, there are l sub-clusters of size $(l-1)$. In line 7, the `SubsetCheck` function checks if all of these l sub-clusters are in the set of F-clusters FC_{l-1}. If any sub-cluster is absent, then the candidate is dropped immediately. The `Validate` function in line 8 joins the l sub-clusters of size $(l-1)$ to validate if the candidate has at least δ supporting matrices. Note that in each supporting matrix, the candidate must have at least min_s samples. If a candidate is an F-cluster, the algorithm outputs it. The algorithm terminates if no more candidate can be generated.

This basic approach is inefficient for mining F-clusters of large size. To find an F-cluster of size l, all of its 2^l sub-clusters have to be generated and validated. If $min_g = 30$, then the algorithm needs to generate at least 2^{30} sub-clusters before a valid F-cluster is reached.

4.3 A Hybrid Approach

In this section, we present a much more efficient algorithm to mine F-clusters. This algorithm takes advantage of a hybrid search strategy, i.e., it enumerates the combinations of sample-pair MDSs to generate candidates, and uses frequent gene-pairs MDSs to validate the candidates. By using the hybrid search strategy, we can avoid expensive clique detection, which is necessary in previous methods, such as [27]. Our method also incorporates novel strategies to prune the search space.

4.3.1 The Algorithm

As shown in Algorithm 2, the algorithm starts by finding all frequent gene-pair MDSs and sample-pair MDSs (lines 1 and 2). Then it applies the initial pruning presented in Section 4.1 (line 3). In line 4, the algorithm sorts the sample-pair MDSs into an ordered list stored in SP. The idea is to place the sample-pair MDS with highest potential pruning power first in the list. In SP, sample-pair MDSs are grouped by matrices, and are placed in ascending order of the number of sample-pair MDSs per matrix. That is, if the number of sample-pair MDSs of a matrix M_a is smaller than that of M_b, the sample-pair MDSs of M_a will be placed before those of M_b in SP. For the sample-pair MDSs in the same matrix, they are placed in ascending order of the size of their gene sets. We will see later that examining sample-pair MDSs in this order maximizes the pruning power and accelerate the enumeration process.

Algorithm 2: The Hybrid F-cluster Miner

Input: a set of gene expression matrices $\{M_1, M_2, \ldots, M_K\}$, cluster threshold ϵ, minimum number of genes min_g, minimum number of samples min_s, minimum support δ.
Output: the complete set of F-clusters.

```
1  FGP ← frequent gene-pair MDSs
2  SP ← sample-pair MDSs in each matrix;
3  Prune FGP and SP;
4  Sort sample-pair MDSs in SP;
5  EnumStack ← SP;
6  while EnumStack ≠ ∅ do
7      c' = EnumStack.pop();
8      for sp ∈ SP do
9          c = Intersect(sp, c');
10         if c.size ≥ min_g then
11             if c.potential-support ≥ δ then
12                 if c ∉ FC ∪ IFC then
13                     if Validate(c, FGP) is true then
14                         FC ← c;
15                     else
16                         IFC ← c;
17                         EnumStack.push(c);
18                     end
19                 end
20             else
21                 EnumStack.push(c);
22             end
23         end
24     end
25 end
26 Return FC.
```

In the next step, the algorithm makes a copy of SP into a stack *EnumStack* (line 5) with the sample-pair MDS of the highest pruning power at the top. Then the algorithm starts to enumerate the combinations of sample-pair MDSs stored in *EnumStack* to generate candidate F-clusters. The function `Intersect` in line 9 returns a candidate c. The gene set of c is the intersection of the gene sets of a sample-pair MDS in SP and the element c' at the top of *EnumStack*. Please note that c' only needs to intersect with the sample-pair MDSs that have not been used in generating c'. In line 13, the `Validate` function validates if a candidate is F-cluster. If it is an F-cluster, it will be stored in the set of F-

clusters FC. Otherwise, it will be put into the set of invalid candidate F-clusters IFC, and pushed back into the stack for further enumeration.

For example, let's consider the dataset shown in Table 1(a) to 1(c). Suppose $\delta = 2, \epsilon = 0, min_g = 3, min_s = 3$. The algorithm first finds the frequent gene-pair MDSs and sample-pair MDSs. It then sorts the sample-pair MDSs in SP. The resulting sample-pair MDSs in SP are shown in Table 3(a). (For simplicity of presentation, we do not show the initial pruning here.) As we can observe from the table, the sample-pair MDSs of matrix M_2 is placed before those of M_1, since the number of sample-pair MDSs of M_2 is smaller than that of M_1. In each matrix, the sample-MDSs are sorted so that the MDSs containing less genes are ordered before the MDSs containing more genes. Then the sorted sample-pair MDSs are pushed into $EnumStack$ for enumeration. The initial inputs of function `Intersect` are $(\{g_1, g_2, g_3, g_4\} \times \{\{s_{1_2}, s_{3_2}\}_{M_2}\})$ and $(\{g_1, g_2, g_3, g_5\} \times \{\{s_{2_2}, s_{6_2}\}_{M_2}\})$. The resulting candidate contains gene set $\{g_1, g_2, g_3\}$.

Candidate validation (line 13) is a nontrivial process. We propose to utilize frequent gene-pair MDSs to expedite this process, which will be discussed further in Section 4.3.2. Our algorithm also incorporates effective pruning strategies to avoid unnecessary validations. Here we give an overview of these pruning strategies. The detailed explanations can be found in Section 4.3.3. In line 10, we use the threshold min_g to prune. If a candidate does not contain sufficiently many genes, we do not need to validate it. In line 11, we use potential support to prune and only retain a candidate if its potential support is at least δ. In line 12, the algorithm checks if the candidate is already validated before. If it is either in FC or IFC, then there is no need to validate. We also use minimum support to prune so that we only need to enumerate the combinations of a subset of the sample-pair MDSs in *EnumStack*.

4.3.2 Candidate Validation

To validate a candidate, we *join* all frequent gene-pair MDSs that are sub-clusters of the candidate. For example, suppose that $\delta = 2, min_g = 3, min_s = 2$. The frequent gene-pair MDSs are shown in Table 2. To validate a candidate containing genes $(\{g_1, g_2, g_3\})$, we take the intersection of the sample sets of these gene-pair MDSs within each matrix. If the size of the resulting sample set is at least min_s, the matrix is a supporting matrix. An F-cluster is validated if the number of supporting matrices is at least δ. For example, $(\{g_1, g_2, g_3\} \times \{\{s_{1_1}, s_{4_1}, s_{5_1}\}_{M_1}, \{s_{2_2}, s_{6_2}\}_{M_2}\})$ is a valid F-cluster. During the validation process, the frequent gene-pair MDSs are stored in a prefix tree for fast access.

Why do we use the frequent gene-pair MDSs to validate candidate clusters?

Genes	Sample-pairs (M_1)	Sample-pairs (M_2)
g_1, g_2, g_3, g_4	$\emptyset$	$\{s_{1_2}, s_{3_2}\}$
g_1, g_2, g_3, g_5	$\emptyset$	$\{s_{2_2}, s_{6_2}\}$
g_1, g_2, g_3, g_4	$\{s_{4_1}, s_{5_1}\}$	$\emptyset$
g_1, g_2, g_3, g_4, g_5	$\{s_{1_1}, s_{4_1}\}$	$\emptyset$
g_1, g_2, g_3, g_4, g_6	$\{s_{1_1}, s_{5_1}\}$	$\emptyset$

(a) Sample-pair MDSs in matrices M_1 and M_2

Genes	Sample-pairs (M_1)	Sample-pairs (M_2)
g_1, g_2, g_3	$\{s_{4_1}, s_{5_1}\}\{s_{1_1}, s_{4_1}\}\{s_{1_1}, s_{5_1}\}$	$\{s_{1_2}, s_{3_2}\}\{s_{2_2}, s_{6_2}\}$

(b) Candidate generated by function Intersect()

Table 3. Validating candidate by sample-pair MDSs

Note that, in principle, there are two ways to validate a candidate, i.e., we can use either sample-pair MDSs or frequent gene-pair MDSs.

Using sample-pair MDSs we can validate the candidates in the following way. After `Intersect` c' with some sample-pair MDS (line 9), we get a candidate c. We then check if in each supporting matrix of c, the sample-pairs can be merged to a larger sample set. A sample set can only be formed if for any two samples in the sample set, a corresponding sample-pair MDS exists. For example, suppose that $\delta = 2, \epsilon = 0, min_g = 3, min_s = 3$. Using the dataset shown in Table 1(a) to 1(c), the sample-pair MDSs are shown in Table 3(a). We need to examine every combination of sample-pairs. The resulting candidate is shown in Table 3(b). Then we check whether the sample-pairs can be merged for each of the two matrices M_1 and M_2. In matrix M_1, we check if sample-pairs $\{s_{4_1}, s_{5_1}\}$, $\{s_{1_1}, s_{4_1}\}$ and $\{s_{1_1}, s_{5_1}\}$ can be merged. This step is equivalent to finding maximum cliques in a graph: each sample is a vertex in the graph, and each sample-pair is an edge connecting the two vertices. If the size of a maximum clique is at least min_s, then the corresponding sample-pairs can be merged into a sample set and the matrix is a supporting matrix of the candidate. In our example, the maximum clique in matrix M_1 is of size 3, thus the 3 sample pairs can be merged to a set of 3 samples $\{s_{1_1}, s_{4_1}, s_{5_1}\}$. In M_2 the size of maximum clique is 2. Thus the two sample-pairs cannot be merged to a larger sample set. The major drawback of this validation approach is that maximum clique detection is an NP-complete problem [15] and it needs to be performed many times. Previous methods [27, 29] used this approach. However, even the best clique detection algorithms [18, 19, 22] is unable to handle the size and complexity of graphs in our problem.

An alternative and more efficient way to validate a candidate is by joining the frequent gene-pair MDSs. For a size-n

candidate c consisting of gene set $X = \{g_{i_1}, g_{i_2}, \cdots, g_{i_n}\}$, there are $\binom{n}{2}$ frequent gene-pair MDSs that are sub-clusters of X. Thus in order to validate c, we only need to join these $\binom{n}{2}$ frequent gene-pair MDSs.

4.3.3 Pruning Strategies

In this section, we discuss in detail the pruning strategies used by our algorithm. The effectiveness of these pruning strategies are validated by the experimental result shown in Section 5.2.

Pruning strategy 1. In line 10, we use min_g for pruning. If the size of a candidate is smaller than min_g, there is no need to validate it and any other candidates containing it. In each matrix, the sample-pair MDSs are sorted in ascending order of the size in order to expedite the enumeration process (line 4). This is because the intersection of sample-pair MDSs having fewer genes is more likely to be smaller and fail the min_g threshold. By joining small sample-pair MDSs first, we can quickly restrict the search space.

Pruning strategy 2. In line 11, we use potential support to prune. For a candidate c, we say that matrix M_i *potentially supports* c, if c contains at least $\binom{min_s}{2}$ sample-pairs in M_i. A necessary condition for a candidate to be an F-cluster is that the potential support of the candidate must be greater or equal to the minimum support. The rationale is that we need at least $\binom{min_s}{2}$ sample-pairs in order to merge them to a set of min_s samples. The initial pruning step discussed in Section 4.1 is a special case of this strategy. Thus we can "revise" the candidate by intersecting additional sample-pair MDSs until there are at least δ matrices potentially support c. Intuitively, the successive intersections are likely to increase the number of supporting matrices but shrink the gene set of the candidate. If the size of a candidate (i.e., the number of genes) is already smaller than min_g before its potential support meets the minimum support δ, we can prune it without any validation. For example, consider the candidate shown in Table 3(b), suppose $\delta = 2$, and $min_s = 3$. M_1 potentially supports it, but M_2 does not. Thus, we do not need to validate this candidate.

Pruning strategy 3. In line 12, we use FC and IFC to prune. The algorithm stores the F-clusters and invalid candidates in FC and IFC respectively using prefix trees. If a new candidate containing at least min_g genes and its potential support is above δ, the algorithm first checks if the gene set is a prefix of the gene set of any F-cluster in FC. If not, then the algorithm checks if the gene set of any invalid candidate in IFC is a prefix of the gene set of the candidate. If both cases are false, then we perform the necessary candidate validation. For example, suppose that in FC there is an F-cluster containing gene set $gs = \{g_1, g_2, g_3, g_4, g_5\}$. Then any candidate consisting of gene set that is a prefix of gs (e.g., $\{g_1, g_2, g_3\}$) does not need to be validated, since it must also be an F-cluster. A similar example can be easily constructed for checking IFC.

Pruning strategy 4. We also use the minimum support δ to prune candidates during enumeration, so that we only enumerate combinations a subset of elements in $EnumStack$. Algorithm 2 employs a total order among all sample-pair MDSs. We only need to intersect sample-pair MDSs from the first $(K - \delta + 1)$ matrices in *EnumStack* with sample-pair MDSs of lower ranks in SP. The sample-pair MDSs from the last $(\delta - 1)$ matrices in *EnumStack* do not need to be checked, since the candidates generated by intersecting them either do not have enough supporting matrices or have been generated before. This can be proved easily by contradiction. (The proof is omitted here.) The following example illustrates this pruning strategy. Suppose that there are 5 matrices, with 10, 20, 30, 40, and 50 sample-pair MDSs, respectively. If minimum support $\delta = 3$, we only need to intersect the top 60 sample-pair MDSs in *EnumStack* (from the first 3 matrices) with those MDSs of lower ranks in SP. There is no need to check the last 120 MDSs since either there are at most 2 matrices supporting the candidates generated by them or the candidates have been generated before. Recall that our algorithm places matrices with large number of MDSs at the bottom of *EnumStack*. This order is optimal in minimizing the number of MDSs that need to be examined.

5 Experiments

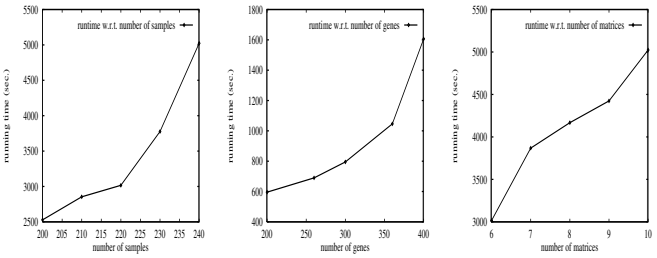
To evaluate the efficiency and effectiveness of the F-cluster algorithm, we performed experiments on real gene expression profiles generated by different microarray techniques. The experiments were performed on a 2.6-GHz PC with 1G memory running WindowsXP system.

5.1 Data Sources

We use the microarray dataset in [12] to perform our experiments. The dataset includes microarrays generated by 3 different techniques, i.e., serial analysis of gene expression (SAGE), cDNA microarrays, and Affymetrix oligonucleotide microarrays.

5.2 Efficiency

We compared the naive 2-step approach (Section 1.3), the basic bottom-up approach (Section 4.2) and the hybrid approach (Section 4.3). But both the naive 2-step approach and the basic bottom-up approach are usually too inefficient to run to completion and cause memory overflow (because of the huge number of candidate clusters generated). Therefore, we only report the results of the hybrid approach. Since the original dataset only has 3 gene expression matrices, in order to perform an in-depth analysis, we partitioned the SAGE microarray matrix into 10 submatrices as the test dataset to demonstrate the efficiency of our algo-



(a) Varying # samples (b) Varying # genes (c) Varying # matrices

Figure 3. Efficiency evaluation

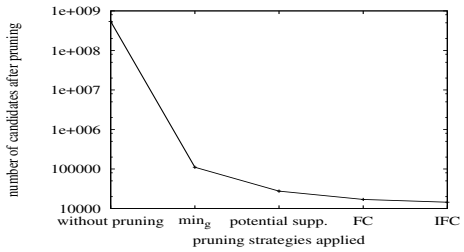


Figure 4. number of candidates after pruning

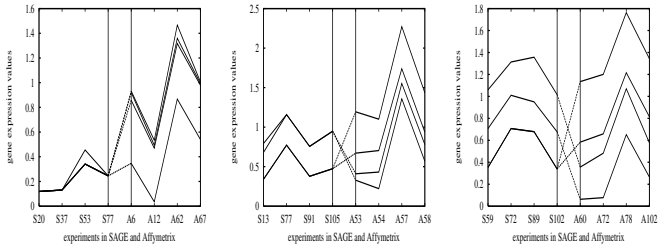
rithm. Each matrix contains 500 genes and 24 samples. We set $min_g = 5\% \cdot \#genes$, $min_s = 2$, $\epsilon = 0.15$ and $\delta = 2$. Figures 3(a) to 3(c) show the running time of our algorithm as a function of different parameters.

As shown in Figure 3(a), the actual running time is approximately quadratic in the number of samples. Theoretically, the worst case running time is exponential with respect to the number of samples, since we enumerate the combinations of sample-pair MDSs to generate the candidate clusters. The quadratic performance demonstrates the effectiveness of the pruning strategies used by the algorithm. Figure 3(b) shows that the running time of our algorithm follows a similar trend in terms of number of genes. This is because even though the candidates are generated by enumerating the combinations of sample-pairs MDSs, we use frequent gene-pair MDSs to validate the candidates. The number of frequent gene-pair MDSs is roughly quadratic in the number of genes. Figure 3(c) shows an approximate quadratic running time with respect the number of matrices. The reason is that under the same minimum support threshold, the more matrices the dataset has, the bigger the F-clusters are. Thus in this case it is likely that the enumeration process need not go very deep into the search space before the candidates are validated to be F-clusters. The result shows the tradeoff between the number of sample-pairs and size of the F-clusters.

Figure 4 demonstrates the effectiveness of the pruning methods used by our algorithm. The dataset contains 200 genes and 10 matrices, with 24 samples in each matrix. The

F-clusters	Ensembl IDs	GO terms
Cluster1	ENSG00000175467 ENSG00000114514 ENSG00000109756 ENSG00000182606	intracellular (GO: 0005622)
Cluster2	ENSG00000116754 ENSG00000167325 ENSG00000112739 ENSG00000005339	nucleobase, nucleoside, nucleotide and nucleic acid metabolism (GO: 0006139)
Cluster3	ENSG00000142230 ENSG00000121031 ENSG00000196776 ENSG00000147162	protein binding (GO: 0005515)

Table 4. F-clusters and corresponding GO terms



(a) Cluster 1 (b) Cluster 2 (c) Cluster 3

Figure 5. Three F-clusters

parameters are set to be the same as in Figure 3(a) to 3(c). The figure shows the number of candidates before and after applying the pruning strategies sequentially as described in Section 4.3.3, i.e., pruning by min_g, potential support, and checking FC and IFC. As we can see from the figure, the number of candidates is reduced by orders of magnitude after applying these pruning methods.

5.3 Effectiveness

In this section, we show some F-clusters found by our algorithm. We ran our algorithm on SAGE, cDNA, and Affymetrix microarrays. Each matrix includes 350 randomly chosen genes that are shared by all 3 microarrays and 130 randomly chosen samples. We require that an F-cluster must be preserved in at least 2 microarrays and set $\epsilon = 0.15$.

Due to space limitation, we only report 3 size-4 F-clusters that are preserved in SAGE and Affymetrix microarrays. The Gene Ontology (GO) is a controlled vocabulary that describes the roles of genes and proteins in all organisms [6]. GO is composed of three independent ontologies: biological process, molecular function, and cellular component. We used Uniprot databases

(http://www.pir.uniprot.org) and Ensembl v37 databases (http://www.ensembl.org) to verify the biological significance of these F-clusters. Table 4 shows the Ensembl gene IDs of the genes in the F-clusters and the corresponding GO terms that confirm the F-clusters. As we can see from the table, genes in each cluster share the same GO annotation, which verifies the biological significance of the F-clusters.

Figure 5(a) to 5(c) show the expression patterns of the 3 F-clusters. As we can see from the figures, genes in each of the cluster have similar expression patterns in both microarrays. However, the expression patterns across different matrices cross over each other and do not share the same scaling and shifting factors. Clearly, applying the existing methods to the concatenated microarrays cannot identify such patterns.

6 Conclusion

Recently there has been extensive research in analyzing microarray data. All previous approaches focused on mining patterns in a single matrix. In this paper, we propose F-cluster model to overcome this limitation. This model can find the clusters that are preserved in expression data generated by different microarray techniques. We develop an efficient algorithm to mine the F-clusters simultaneously from multiple matrices. Different from previous sample based or gene based enumeration method, our approach adopts a hybrid search strategy that combines the benefits of both. This approach incorporates effective pruning strategies to avoid unnecessary computations. Our experimental results on real dataset demonstrate the effectiveness of the F-cluster model and the efficiency of the hybrid mining algorithm. Last but not least, even though we focus on mining heterogeneous data, F-cluster can readily handle homogeneous matrices. In our future work, we plan to extend our method to take into account missing values.

References

[1] C. Agarwal, C. Procopiuc, J. Wolf, P. Yu, and J. Park. Fast algorithms for projected clustering. In *SIGMOD*, 1999.

[2] C. Aggarwal and P. Yu. Finding generalized projected clusters in high dimensional spaces. In *SIGMOD*, 2000.

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

[4] A. Alizadeh and et al. Distinct types of diffuse large b-cell lymphoma identified by gene expression profiling. *Nature*, 403:503–11, 2000.

[5] U. Alon, N. Barkai, D. Notterman, K. Gish, S. Ybarra, D. Mack, and A. Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide array. In *Proc. Natl. Acad. Sci. USA*, volume 96, pages 6745–50, 1999.

[6] M. Ashburner and et al. Gene ontology: tool for the unification of biology. *The gene ontology consortium, Nat. Genet.*, 25:25–29, 2000.

[7] A. Ben-Dor, R. Shamir, and Z. Yakhini. Clustering gene expression patterns. *Journal of Comp. Bio.*, 6:281–97, 1999.

[8] Y. Cheng and G. Church. Biclustering of experssion data. In *Proc. of the ISMB*, 2000.

[9] P. D'haeseleer, X. Wen, S. Fuhrman, and R. Somogyi. Mining the gene expression matrix: inferring gene relationships from large scale gene expression data. *Info. Process. in Cells and Tissues*, pages 203–12, 1998.

[10] M. Eisen, P. Spellman, P. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proc. Natl. Acad. Sci. USA*, 95:14863–68, 1998.

[11] N. Friedman and et al. Using bayesian network to analyze expression data. In *ISMB*, 2000.

[12] O. Griffith and et al. Assessment and integration of publicly available sage, cdna microarray, and oligonucleotide microarray expression data for global coexpression analyses. *Genomics*, 86(4):476–48, 2005.

[13] H. Hu, X. Yan, Y. Huang, J. Han, and X. Zhou. Mining coherent dense subgraphs across massive biological networks for functional discovery. *ISMB*, 2005.

[14] D. Jiang, J. Pei, M. Ramanathan, C. Tang, and A. Zhang. Mining coherent gene clusters from gene-sample-time microarray data. In *SIGKDD*, 2004.

[15] R. Karp. Reducibility among combinatorial problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, 1972.

[16] D. Lockhart and et al. Expression monitoring by hybridization to high-density oligonucleotide arrays. *Nat. Biotechnol.*, 14:1675–80, 1996.

[17] M. Middendorf, A. Kundaje, C. Wiggins, Y. Freund, and C. Leslie. Predicting genetic regulatory response using classification. In *Proc. of ISMB*, 2004.

[18] J. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3:23–28, 1965.

[19] R. Osteen. Clique detection algorithms based on line addition and line removal. *SIAM Journal on Applied Mathematics*, 26(1):126–35, 1974.

[20] M. Schena, D. Shalon, R. Davis, and P. Brown. Quantitative monitoring of gene expression patterns with a complementary dna microarray. *Science*, 270:467–70, 1995.

[21] R. Shamir and R. Sharan. Click: A clustering algorithm for gene expression analysis. In *ISMB*, 2000.

[22] V. Stix. Finding all maximal cliques in dynamic graphs. *Computational Optimization Appl.*, 27(2):173–86, 2004.

[23] P. Tamayo and et al. Interpreting patterns of gene expression with self-organizing maps: Methods and application to hematopoietic differentiation. In *Proc. Natl. Acad. Sci. USA*, volume 96, pages 2907–12, 1999.

[24] A. Tanay, R. Sharan, M. Kupiec, and R. Shamir. Revealing modularity and organization in the yeast molecular network by integrated analysis of highly heterogeneous genomewide data. *Proc. Natl. Acad. Sci. USA*, 101(9):2981–86, 2004.

[25] S. Tavazoie, J. Hughes, M. Campbell, R. Cho, and G. Church. Systematic determination of genetic network architecture. *Nat. Genetics*, 22:281–85, 1999.

[26] V. Velculescu, L. Zhang, B. Vogelstein, and K. Kinzler. Serial analysis of gene expression. *Science*, 270, 1995.

[27] H. Wang, W. Wang, J. Yang, and P. Yu. Clustering by pattern similarity in large data sets. In *SIGMOD*, 2002.

[28] X. Xu, Y. Lu, A. Tung, and W. Wang. Mining shifting-and-scaling co-regulation patterns on gene expression profiles. In *ICDE*, 2006.

[29] L. Zhang and M. Zaki. Tricluster: An effective algorithm for mining coherent clusters in 3d microarrary data. In *SIGMOD*, 2005.