Analysis of Gene Expression Data **Spring Semester, 2002**

Lecture 8: May 30, 2002

Lecturer: Ron Shamir and Roded Sharan *Scribe: Irit Gat and Amos Tanay*

8.1 Introduction to Biclustering

As we have seen in previous lectures, a very central method in the analysis of gene expression data is clustering. Clustering algorithms are used to transform a very large matrix of expression values to a more informative collection of gene or condition sets. The members of each cluster are assumed to share function or form some biological module. Clustering is a *global* technique and as such has several limitations. First, clustering partitions the elements into groups, so each element may appear in at most one group. Second, when clustering gene expression data, we group the genes according to their behavior over *all* experiments (similarly for conditions). This may be problematic when working with large databases that may include many different conditions, only few of which trigger some common gene behavior.

Suppose we are studying yeast cell biology. We may try to use gene expression data in order to identify functional modules, i.e., large sets of genes sharing some important cellular function or process. Clustering the genes may give good results as long as we are targeting a very concrete subsystem. Indeed several works ([17, 7]) used clustering to vastly expand our knowledge on cell cycle or stress response. However, many genes, even in our particular example, are important to both stress response and cell cycle as the two systems are intimately related, so clustering the joint data set would have to create some arbitrary partition of the two systems, loosing information on their common parts.

A second example for the limitations of clustering comes from clinical studies of cancer. We may cluster tissues of many patients suffering from several type of cancer in order to try and identify clinically important subclasses. We may also try to compare our classes with some additional information on the patients (age, sex, type of cancer, smoking years, prognosis). When using global clustering of the tissues we can only find one (hopefully the most significant) signal in the data, for example we may separate the different cancer types. Other signals, which may be important, will be missed since we are associating tissues with a single effect.

To try and address these shortcomings, the concept of **biclustering** was introduced to gene expression analysis. Biclustering was first defined in the seventies [10] and was applied to several domains before Cheng and Church [4] coined its usage in computational biology. Given a gene expression matrix, we search for submatrices that are tightly co-regulated according to some scoring criterion. We do not require the identified submatrices

to be disjoint or to cover the entire matrix, instead we wish to build a diverse collection of submatrices that will capture all the significant signals in our data.

In the following lecture we will describe three approaches to biclustering. We shall first describe the work of Cheng and Church who used a simple uniformity criterion to score submatrices and developed a greedy algorithm to identify high scoring ones. We will next describe a more recent method, called SAMBA, developed for identifying biclusters of high statistical quality using combinatorial algorithms [18]. SAMBA provides well defined statistical model for the data and is guaranteed to find the optimum (under some assumption). Finally we shall outline a third biclustering approach, called plaid models, which originates from statistics and fits the expression matrix to a linear sum of submatrix signals.

Before going into details, we state again the basic reasoning of using biclustring and the key differences between biclustering and standard clustering. Biclustering is a **local** technique by nature, i.e., we try to find local, significant signals in the data. Clustering, on the other hand, tries to model the whole dataset by reducing it to a collection of subsets. A successful collection of biclusters will provide a more detailed model of the data and can uncover more biological implications of it. However, biclustering results will be harder to interpret.

8.2 Cheng and Church's Algorithm

The first application of biclustering to gene expression data was the work of Cheng and Church (2000). Stating its goal as the ability to find signals more delicate than clusters, the methodology is based on a simple uniformity goal (the Mean Residue Score, defined below) and uses a greedy algorithm to find one bicluster, combined iteratively to produce a collection of biclusters.

8.2.1 The Algorithm

We denote the input matrix of expression data as $A = (a_{ij})$ and the rows (columns) set by R (C). A submatrix is denoted by $A_{IJ}(I \subseteq R, J \subseteq C)$ and we use the auxiliary notation $a_{Ij} = \frac{\sum_{i \in I} a_{ij}}{|I|}$ (sub column average) $a_{iJ} = \frac{\sum_{j \in J} a_{ij}}{|J|}$ (sub row average) and $a_{IJ} = \frac{\sum_{i \in I, j \in J} a_{ij}}{|I||J|}$ (submatrix average).

We define the *Residue Score* of an element a_{ij} in a submatrix A_{IJ} as $RS_{IJ}(i,j) = a_{ij} - a_{Ij} - a_{iJ} + a_{IJ}$ and the *Mean Residue Score* of the submatrix as $H(I,J) = \sum_{i \in I, j \in J} \frac{RS_{ij}^2}{|I||J|}$. A completely uniform matrix will score zero. A submatrix in which all entries are the sum of a column parameter and a row parameter ($a_{ij} = b_i + c_j$) would also score zero. On the other hand a random submatrix (normally distributed with any parameter) would have the variance of the distribution as its expected score. We define a δ bicluster to be a submatrix

(I, J) for which $H(I, J) \leq \delta$. The biclustering algorithm will search for a δ-bicluster assuming that the parameter δ was chosen appropriately to avoid random signal identification. For example, we may choose δ as the minimal (i.e. best) score of the output of a clustering algorithm.

The optimization problem of identifying the largest δ-bicluster (the one for which $|I| = |J|$ is the largest) is NP hard as can be seen by a simple reduction from BALANCED COMPLETE BIPARTITE SUBGRAPH. We are thus interested in heuristics for finding a large δ bicluster in reasonable time. We next present such heuristic which is a greedy algorithm in essence, show how to speed it up and use it as a subroutine for finding many biclusters.

A naive greedy algorithm for finding a δ-bicluster may start with the entire matrix and at each step try all single rows/columns addition/deletion, applying the best operation if it improves the score and terminating when no such operation exists or when the bicluster score is below δ. However, simply recalculating all averages and mean residues for each operation may be too expensive for large matrices. Cheng and Church's algorithm uses the structure of the mean residue score to enable faster greedy steps. The idea is based on the following lemma:

Lemma 8.1 *The set of rows that can be completely or partially removed with the net effect of decreasing the mean residue score of a bicluster A_{IJ} is :*

$$R = \{i \in I; \frac{1}{|J|} \sum_{j \in J} RS_{I,J}(i, j) > H(I, J)\} \tag{8.1}$$

In words, it is safe to remove any row for which the average contribution to the total score is greater then its relative share. The same argument is correct for columns and gives rise to the following greedy algorithm that iteratively remove rows/columns with the maximal average residue score (Figure 8.1).

Note that since a 1 by 1 submatrix is always a δ-bicluster we should hope that the deletion algorithm will terminate with a large bicluster. It is natural to try and add rows/columns in an analogous way, using the equivalent lemma and algorithm (Figure 8.2):

Lemma 8.2 *The set of rows (columns) that can be completely or partially added with the net effect of decreasing the score of a bicluster A_{IJ} is :*

$$R = \{i \notin I; \frac{1}{|J|} \sum_{j \in J} RS_{I,J}(i, j) \leq H(I, J)\} \tag{8.2}$$

The exact details of the heuristic are not necessarily optimal for any situation. For example, the algorithm presented here is tailored for cases where there are much more rows than columns.

Input: Expression matrix A on genes S, conditions C and a parameter δ.
Output: A_{IJ} a δ-bicluster.
Init: $I = S$,$J = C$.
Iteration:
Calculate a_{Ij}, a_{iJ} and $H(I,J)$. If $H(I,J) \leq \delta$ output I, J.
For each row calculate $d(i) = \frac{1}{|J|}\sum_{j\in J} RS_{I,J}(i,j)$.
For each column calculate $e(j) = \frac{1}{|I|}\sum_{i\in I} RS_{I,J}(i,j)$.
Take the best row or column and remove it from I or J.

Figure 8.1: Single node deletion algorithm.

Input : Expression matrix A, the parameter δ and I, J specifying a δ bicluster.
Output : AI', J' - a δ-bicluster with $I \subseteq I', J \subseteq J'$.
Iteration
Calculate a_{Ij}, a_{iJ} and $H(I,J)$.
Add the columns with $\frac{1}{|I|}\sum_{i\in I} RS_{I,J}(i,j) \leq H(I,J)$.
Calculate a_{Ij}, a_{iJ} and $H(I,J)$.
Add the rows with $\frac{1}{|J|}\sum_{j\in J} RS_{I,J}(i,j) \leq H(I,J)$.
If nothing was added, halt.

Figure 8.2: Node addition algorithm.

The Cheng-Church algorithm suggests two additional improvements to the basic deletion-addition algorithm. The first improvement suggests a **multiple node deletion** in cases where the data set is large. This is done by removing at each deletion iteration all rows/columns for which $d(i) > \alpha H(I,J)$ for some choice of α. The idea is to perform large steps until the submatrix is relatively small and indeed it is shown that such steps can be done safely (without increasing the score).

The second algorithmic improvement involves the addition of **inverse** rows to the matrix, allowing the identification of biclusters which contains co-regulation and inverse co-regulation (i.e., cases where two genes always change in opposite directions). We shall return to this theme later when discussing SAMBA's models.

As mentioned in the introduction, the goal of a biclustering algorithm is to identify all (or many of) the signals in the data set, so clearly, finding one bicluster is not enough. The Cheng-Church solution to this requirement uses the δ-bicluster algorithm as a subroutine

and repeatedly applies it to the matrix. In order to avoid finding the same bicluster over and over again, the discovered bicluster is masked away from the data, by replacing the values of its submatrix by random values. The general biclustering scheme is outlined in Figure 8.3.

Input : Expression matrix A, the parameter δ, the number of biclusters to report n
Output : n δ-biclusters in A.
Iteration
Apply multiple node deletion on A giving I', J'.
Apply node addition on I', J' giving I'', J''.
Store I'', J'' and replace $A_{I'',J''}$ values by random numbers.

Figure 8.3: Cheng-Church biclustering algorithm.

8.2.2 Experiments

We next describe some of the experiments done by Cheng and Church to validate their approach. Experiments were done using two datasets, one of human lymphoma ([3]) and the other of yeast data ([19]). Working with the yeast data, the parameter δ was chosen to be a bit more then the minimal score of the reported clusters. A large set of random submatrices of varying sizes was then scored and compared to the selected threshold. The simulation showed that small δ-biclusters have a considerable chance of being random (15% for 3 by 6 matrices, 0.06% for 10 by 6 matrices), but larger $\delta - biclusters$ may be far from random (although random here referees to random submatrices rather than random expression data).

Application of the algorithm to both datasets produced 100 biclusters on each, and some effort was made to test their relation with global clustering and biological information. For example, the hierarchical clustering of the lymphoma tissues was compared with the biclusters by testing the condition sets of each bicluster and its partition among the two main branches of the hierarchy. We shall return to the question of how to validate biclustering when discussing the SAMBA algorithm.

8.3 SAMBA

We next describe SAMAB, the biclustering algorithm of [18]. SAMBA stands for Statistical Algorithmic Method for Bicluster Analysis. The motivation underlying its development is the need for a fast biclustering method that would produce statistically significant results as part of its design. The key point in the understanding of SAMBA is the statistical model

used by the bicluster scoring scheme. The hope is that a model which capture the essential features of the data would guarantee high quality results.

8.3.1 Data Modeling

The intuitive notion of a bicluster is a subset of genes that exhibit similar expression patterns over a subset of conditions. Following this intuition we define a bicluster as a subset of genes that *jointly respond* across a subset of conditions, where a gene is termed responding in some condition if its expression level changes significantly at that condition w.r.t. its normal level.

The expression data is modeled as a bipartite graph whose two parts correspond to conditions and genes, respectively, with edges for significant expression changes. Later, we shall assign weights to the vertex pairs of the bipartite graph according to a statistical model, so that heavy subgraphs correspond to significant biclusters. We can tag each edge to incorporate the direction of regulation (up or down) as we shall see later. For now assume edges are not tagged.

Formally, given an input gene expression dataset we form a bipartite graph $G = (U, V, E)$ (see Figure 8.4 for an example). In this graph, U is the set of conditions, V is the set of genes, and $(u, v) \in E$ iff v responds in condition u, that is, if the expression level of v changes significantly in u. A bicluster corresponds to a subgraph $H = (U', V', E')$ of G, and represents a subset V' of genes that are co-regulated under a subset of conditions U' (see Figure 8.4). The *weight* of a subgraph (bicluster) is the sum of the weights of gene-condition pairs in it, including edges and non-edges.

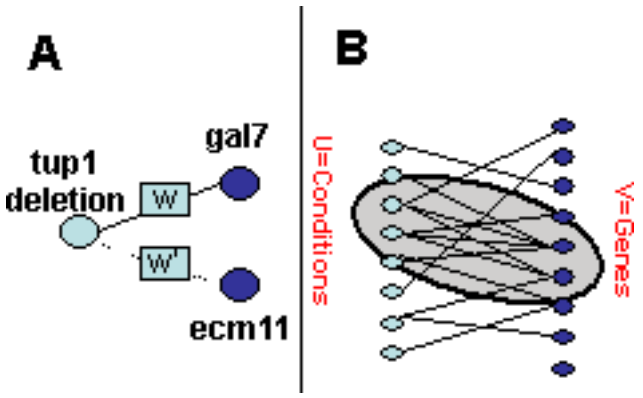


Figure 8.4: Source: [18]. SAMBA modeling : Gene expression data is modeled using a bipartite graph whose two sides correspond to the set of conditions U and the set of genes V. An edge (u, v) indicates the response of gene v in condition u. A statistical model assigns weights to the edges and non-edges of the graph. A: Part of the graph showing the condition "tup1 deletion" and its effect on the genes "gal7" (response) and "ecm11" (no response). B: A heavy subgraph (shaded) representing a significant bicluster.

In order to assign statistical meaning to the weight of a subgraph, the authors develop statistical models for the bipartite graph representation of expression data. Using these models one can derive scoring schemes for assessing the significance of an observed subgraph (corresponding to a bicluster). This is done so that the score can be expressed as a sum of independent contributions from each of the node pairs (condition-genes) in the subgraph. Using this model, we can reduce the biclustering problem to the problem of finding heavy subgraphs in a bipartite graph.

8.3.2 A Simple Model

The first statistical model we present is a simplistic one, and is presented as a motivation for the more sophisticated model that will follow. Let $H = (U', V', E')$ be a subgraph of G. Denote $|U'| = m', |V'| = n'$. Let $p = \frac{|E|}{|U||V|}$, and let $k' = |E'|$. The model assumes that edges occur independently and equiprobably with density p. Denote by $BT(k, p, n)$ the binomial tail, i.e., the probability of observing k or more successes in n trials, where each success occurs independently with probability p. Then the probability of observing a graph at least as dense as H according to this model is $p(H) = BT(k', p, n'm') = \sum_{k''=k'}^{n'm'} \binom{n'm'}{k''} p^{k''}(1-p)^{n'm'-k''}$.

Our goal is to find a subgraph H with lowest $p(H)$. By bounding the terms of the binomial tail using the first term (which is the largest, assuming that $p < 1/2$), we obtain the following upper bound for $p(H)$: $p^*(H) \leq p^{k'}(1-p)^{n'm'-k'} \sum_{k''=k'}^{n'm'} \binom{n'm'}{k''} \leq 2^{n'm'} p^{k'}(1-p)^{n'm'-k'}$. Seeking a subgraph H minimizing $\log p^*(H)$ is equivalent to finding a maximum weight subgraph of G where each edge has positive weight $(-1 - \log p)$ and each non-edge has negative weight $(-1 - \log(1-p))$ since the weight of a subgraph are $log(p^*(H'))$ under these weights.

8.3.3 A Refined Model

The simple model presented above is far from the reality. In fact, if we look at the degree distribution of real gene expression datasets we can identify a very non uniform behavior, where some of the conditions and genes have very high degrees and others very low degrees. Assuming a simple random graph model would result in very high scores for the "bicluster" defined by all high degree nodes in the graph. We next describe a refined null model that takes into account the variability of the degrees in G, i.e., it incorporates the characteristic behavior of each specific condition and gene. This behavior may reflect some experiment artifact (some genes are more likely to have noisy measurements) or some biological actual meaning (some of the genes participate in many of the cellular pathways).

Let $H = (U', V', E')$ be a subgraph of G and denote $\overline{E'} = (U' \times V') \setminus E'$. For a vertex $w \in U' \cup V'$ let d_w^G denote its degree in G. The refined null model assumes that the occurrence of each edge (u, v) is an independent Bernoulli variable with parameter $p_{u,v}$. The probability

$p_{u,v}$ is the fraction of bipartite graphs with degree sequence identical to G that contain the edge (u, v), or

$$p_{u,v} = \frac{|\{G' = (U, V, E') | d_w^{G'} = d_w^G, (u, v) \in E'\}|}{|\{G' = (U, V, E') | d_w^{G'} = d_w^G\}|} \tag{8.3}$$

We can estimate $p_{u,v}$ using a Monte-Carlo like process, starting from the original graph and performing a sequence of random edge swaps that preserve the degrees (A formal proof of a sampling lemma in the space of fixed degree graphs is however still an open problem). The probability of observing H is thus $p(H) = (\prod_{(u,v)\in E'} p_{u,v}) \cdot (\prod_{(u,v)\in \overline{E'}} (1 - p_{u,v}))$. However, we cannot simply compare subgraphs according to this probability, since it improves (decreases) as the size of H increases.

To overcome this problem, we use a likelihood ratio to capture the significance of biclusters. The null model is as stated above. For the alternative model we assume that each edge of a bicluster occurs with constant probability $p_c > \max_{(u,v)\in U\times V} p_{u,v}$. This model reflects our belief that biclusters represent approximately uniform relations between their elements. The log likelihood ratio for H is therefore:

$$\log L(H) = \sum_{(u,v)\in E'} \log \frac{p_c}{p_{u,v}} + \sum_{(u,v)\in \overline{E'}} \log \frac{1 - p_c}{1 - p_{u,v}}$$

Setting the weight of each edge (u, v) to $\log \frac{p_c}{p_{u,v}} > 0$ and the weight of each non-edge (u, v) to $\log \frac{1-p_c}{1-p_{u,v}} < 0$, we conclude that the score of H is simply its weight.

8.3.4 Finding heavy subgraphs

Having assigned weights for edges in our model bipartite graph, such that the weight of a subgraph encode some reasonable likelihood ratio, we now turn to the problem of identifying the maximum likelihood bicluster, which is, under our formulation, the heaviest subgraph.

The computational problem of finding the largest node biclique (the one with the largest number of vertices) in a bipartite graph has an elegant polynomial time algorithm (using matching). Our problem, however, is more closely related to the problem of finding the largest edge biclique (the biclique with the largest number of edges) which is NP-hard for both unweighted [15] and weighted graphs [11]. In fact, the problem of finding the heaviest bipartite subgraph when edges are assigned positive weights and non edges negative weights is hard as well (by reduction from CLIQUE). We thus enforce additional limitation on our graph, namely, restrict the in degree of the genes side, so that a gene that respond in more then d conditions is ignored. This enables us to develop a polynomial algorithm which is later used as the basis for a practical implementation that can avoid the degree restriction. Note that according to the statistical model, genes with high indegree contribute less to the significance of a bicluster, so running the algorithm without them may not be a very restricting limitation.

8.3.5 Maximum Bounded Biclique

We start by describing an $O(|V|2^d)$-time algorithm to find a maximum weight biclique in a bipartite graph whose gene vertices have d-bounded degree. This algorithm will be a key component in the more involved algorithms that follow.

Let $G = (U, V, E)$ be a bipartite graph. We say that G has *d-bounded gene side*, if every $v \in V$ has degree at most d. Let $w : U \times V \to \mathcal{R}$ be a weight function. For a pair of subsets $U' \subseteq U, V' \subseteq V$ we denote by $w(U', V')$ the weight of the subgraph induced on $U' \cup V'$, i.e., $w(U', V') = \sum_{u \in U', v \in V'} w((u, v))$. The *neighborhood* of a vertex v, denoted $N(v)$, is the set of vertices adjacent to v in G. We denote $n = |V|$ throughout.

Problem 8.1 (Maximum Bounded Biclique) Given a weighted bipartite graph G with d-bounded gene side, find a maximum weight complete subgraph of G.

Theorem 8.3 *The maximum bounded biclique problem can be solved in $O(n2^d)$ time and space.*

Proof: Observe that a maximum bounded biclique $H^* = (U^*, V^*, E^*)$ in G must have $|U^*| \leq d$. Figure 8.5 describes a hash-table based algorithm that for each vertex $v \in V$ scans all $O(2^d)$ subsets of its neighbors, thereby identifying the heaviest biclique. Each hash entry corresponds to a subset of conditions and records the total weight of edges from adjacent gene vertices. An iteration over subsets of $N(v)$ can be done in $O(2^d)$ time. Hence, the algorithm spends $O(n2^d)$ time on the hashing and finding U_{best}. Computing V_{best} can be done in $O(nd)$ time, so the total running time is $O(n2^d)$. The space complexity is $O(n2^d)$ due to the hash-table. ■

```
MaxBoundBiClique(U, V, E, d):
Initialize a hash table weight; weight_best ← 0
For all v ∈ V do
    For all S ⊆ N(v) do
        weight[S] ←weight[S]+
                    max{0, w(S, {v})}
        If (weight[S] > weight_best)
            U_best ← S
            weight_best ← weight[S]
Compute V_best = ∩_{u∈U_best} N(u)
Output (U_best, V_best)
```

Figure 8.5: Source: [18]. An algorithm for the maximum bounded biclique problem.

Note that the algorithm can be adapted to give the k condition subsets that induce solutions of highest weight in $O(n2^d \log k)$ time using a priority queue data structure.

8.3.6 Finding Heavy Subgraphs

We will now show how to extend the latter algorithm to find heavy subgraphs which are not necessarily complete. For simplicity we shall describe the algorithm assuming that each edge has weight $+1$ and each non-edge has weight -1. Extension to more general weights can be done in a similar manner. Formally, given a bipartite graph $G = (U, V, E)$ define a weight function $w : U \times V \rightarrow \{-1, 1\}$ such that $w((u, v)) = 1$ for $(u, v) \in E$, and $w((u, v)) = -1$ for $(u, v) \in (U \times V) \setminus E$. Consider the following problem:

Problem 8.2 (Maximum Bounded Bipartite Subgraph) Given a bipartite graph G with d-bounded gene side, find a maximum weight subgraph of G.

Lemma 8.4 *Let $H^* = (U^*, V^*, E^*)$ be a maximum weight subgraph of G. Then every vertex in H^* is connected to at least half the vertices on the other side of H^*.*

Proof: Follows from the choice of weights, since if a vertex $v \in V^*$ has less than $\lceil |U^*|/2 \rceil$ neighbors, then removing v from H^* will result in a heavier subgraph. The proof for $u \in U^*$ is symmetric. ■

Corollary 8.5 *A maximum weight subgraph of G has at most $2d$ vertices from U.*

Lemma 8.6 *Let $H^* = (U^*, V^*, E^*)$ be a maximum weight subgraph of G. For each set $X \subseteq U^*$ there exists a subset $Y \subseteq X$ with $|Y| \geq \lceil |X|/2 \rceil$ such that $Y \subseteq N(v)$ for some $v \in V^*$.*

Proof: Assume there exists $X \subseteq U^*$ such that all subsets $X \cap N(v), v \in V^*$ are of size smaller than $\lceil |X|/2 \rceil$. Then the weight of the subgraph induced on $(U^* \setminus X, V^*)$ exceeds that of H^*, a contradiction. ■

The critical observation now is that we can cover the maximum subgraph using no more the $\lfloor \log(2d) \rfloor$ subset of incoming neigbors. Formally we have the corrolary :

Corollary 8.7 *Let $H^* = (U^*, V^*, E^*)$ be a maximum weight subgraph of G. Then U^* can be covered by at most $\lfloor \log(2d) \rfloor$ sets, each of which is contained in the neighborhood of some vertex in V^*.*

Proof: Denote $|U^*| = t$. By Lemma 8.6 there exists a subset $Y \subseteq U^*$ with $|Y| \geq \lceil t/2 \rceil$, such that $Y \subseteq N(v)$ for some $v \in V^*$. The same holds for the set $U^* \setminus Y$, and we can

continue in this manner until we cover U^*. By construction we have at most $\lfloor \log t \rfloor$ sets in the cover. Since $t \leq 2d$ by Corollary 8.5, the result follows. ■

Corollary 8.7 implies an algorithm to find a maximum weight subgraph. The algorithm tests all collections of at most $\lfloor \log(2d) \rfloor$ subsets of neighborhoods of vertices in V. Since there are $O(n2^d)$ such subsets we have:

Theorem 8.8 *The maximum bounded bipartite subgraph problem can be solved in* $O((n2^d)^{\log(2d)})$ *time.*

8.3.7 Incorporating the Direction of Expression Changes

In our discussion so far, the underlying bipartite graph used for modeling the data contained edges for significantly changed genes, but ignored the type of change (increase or decrease in the expression level). We can integrate additional information into the model by associating a sign of "up" or "down" with each edge. We now have three types of binary relations in our bipartite graphs: An "up" edge, a "down" edge or no edge. It is reasonable to look for a bicluster in which the conditions tend to affect genes in a *consistent* way, i.e., two clustered conditions should either have always the same effect or always the opposite effect on each of the genes. This leads to the definition of a consistent biclique: Given a bipartite graph $G = (U, V, E)$ with edge sign function $c : E \rightarrow \{-1, 1\}$, we say that an induced biclique $H = (U', V', E')$ is *consistent* if there exists an assignment $\tau : U' \cup V' \rightarrow \{-1, 1\}$ such that for every $v \in V', u \in U'$ we have $c((u, v)) = \tau(u)\tau(v)$. The maximum consistent biclique problem in degree-bounded graphs can be solved in polynomial time by reduction to the standard maximum biclique problem:

Proposition 8.9 There is an $O(n2^d)$-time algorithm for the maximum consistent bounded biclique problem on graphs with d-bounded gene side.

Proof: Given G and c, we construct the graph $G' = (U \cup \overline{U}, V \cup \overline{V}, E')$, where $\overline{U}$ and $\overline{V}$ are copies of U and V, respectively, and $E' = \{(u, v), (\overline{u}, \overline{v}) | (u, v) \in E, c((u, v)) = 1\} \cup \{(u, \overline{v}), (\overline{u}, v) | (u, v) \in E, c((u, v)) = -1\}$. Suppose that (U', V') induce a consistent biclique in G of size k with a sign assignment τ. Then $\{v \in U' \cup V' | \tau(v) = 1\} \cup \{\overline{v} | v \in U' \cup V', \tau(v) = -1\}$ induce a biclique of size k in G'. Conversely, if (U', V') induce a biclique in G', then no pair $u, \overline{u}$ is contained in it, so $\{v \in U \cup V | v \in U' \cup V' \text{ or } \overline{v} \in U' \cup V'\}$ induce a consistent biclique in G of the same size, where $\tau(v) = 1$ if $v \in U' \cup V'$ and $\tau(v) = -1$ if $\overline{v} \in U' \cup V'$. The claim thus follows from Theorem 8.3. ■

We can use simialr ideas to handle consitent subgraphs. All we need is to ensure that the sum of weights of edges from a node to two opposite sign neighbors is negative. Given

this condition, the algorithms from previous sections can be applied directly and generate consistent subgraphs.

8.3.8 Significance Evaluation

In this section we develop a method for computing the statistical significance of a bicluster. The method computes a "p-value" for a given bicluster B, i.e., the probability of finding at random a bicluster with at least the weight of B. Let $H = (U', V', E')$ be a subgraph. Suppose at first that U' is fixed, and we wish to compute the probability of observing H, given that its weight is maximum among all subgraphs over the same set of conditions U'. To this end, we note that H is obtained by taking into V' all vertices $v \in V$ whose weight $w(\{v\}, U')$ is positive. Let $f_{U'} : V \to \mathcal{R}$ be a function defined as $f_{U'}(v) = \max\{0, w(\{v\}, U')\}$. For each $v \in V$ we can view $f_{U'}(v)$ as a random variable. The weight of H is just $w(H) = \sum_{v \in V} f_{U'}(v)$, a sum of independent random variables. These variables can be shown to satisfy the requirements of Liapunov's generalization of the Central Limit Theorem (cf. [5]), implying that when $|V|$ is sufficiently large, the weight of H is approximately normally distributed. Hence, we can compute the expectation and variance of $w(H)$ and derive a p-value $p(H)$ for observing a subgraph with such weight.

Finally, we have to accommodate for the fact that the subset U' is optimized by the algorithm. For that, we apply Bonferroni's rule and compute an upper bound on the p-value: $p^*(H) = 2^d p(H)$, We call $\log p^*(H)$ the *significance value* of H.

8.3.9 The SAMBA Algorithm

The scoring scheme and combinatorial algorithms developed above can be combined to create the practical SAMBA algorithm. We cannot apply the theoretical algorithm directly on large datasets since for reasonable degree bounds (e.g. 60), 2^D is too large. The suggested solution avoids hashing all subsets of incoming neighbors per gene and exhaust only subset of small size (typically $4 - 6$ conditions). The heaviest condition sets are then used as kernels that are extended by a local search procedure.

SAMBA thus works as follows: We first form the bipartite graph and calculate vertex pair weights using one of the weighting methods described above. We consider a gene to be up (down) regulated in a condition if its standardized level with mean 0 and variance 1 is above 1 (below -1). Depending on the data, we may choose to work with signed or unsigned graphs.

In the second phase of the algorithm we apply the hashing technique of the algorithm in Figure 8.5 to find the heaviest bicliques in the graph. In fact, SAMBA looks for the k best bicliques intersecting every given condition or gene. We ignore genes with degree exceeding some threshold d, and hash for each gene only subsets of its neighbors of size ranging from N_1 to N_2.

The third phase of the algorithm performs a local improvement procedure on the biclusters in each heap. The procedure iteratively applies the best modification to the bicluster (addition or deletion of a single vertex) until no score improvement is possible. To avoid similar biclusters whose vertex sets differ only slightly, a greedy algorithm is applied. We iterate over all generated biclusters, ordered by their score, and filter out biclusters whose intersection with a previous solution (number of shared conditions times number of shared genes) is above $L\%$.

An implementation of SAMBA can handle large data sets in a few minutes (15,000 genes, 500 conditions $d = 40, N_1 = 4, N_2 = 6, K = 20, L = 30$).

SAMBA(U, V, E, w, d, N_1, N_2, k):
U : conditions. V : genes.
E : graph edges. w : edge/non-edge weights.
N_1, N_2 : hashed set size limits. k : max biclusters per gene/condition.
Initialize a hash table *weight*.
For all $v \in V$ with $|N(v)| \leq d$ do
    For all $S \subseteq N(v)$ with $N_1 \leq |S| \leq N_2$ do
        $weight[S] \leftarrow weight[S] + w(S, \{v\})$.
For each $v \in V$ set $best[v][1 \ldots k]$ to the k heaviest S such that $v \in S$.
For each $v \in V$ and each of the k sets $S = best[v][i]$
    $V' \leftarrow \cap_{u \in S} N(u)$.
    $B \leftarrow S \cup V'$.
    Do {
        $a = argmax_{x \in V \cup U}(w(B \cup x))$
        $b = argmax_{x \in B}(w(B - x))$
        $if w(B \cup a) > w(B - b) then B = B \cup a$
        else $B = B - b$
    } **While** improving
    Store B.
Post process to filter overlapping biclusters.

Figure 8.6: The practical biclustering algorithm.

8.3.10 Validating Biclustering Quality

As the reader may have noted, for many applications in computational biology, it is hard to compare different algorithms and methodologies and state clearly which one is "better". It is however, very important for any scientific discipline to have means for evaluating the quality

of a given result and to make sure the field is indeed making progress. We present examples of two general methodologies for assessing algorithms performance : comparative analysis, matching algorithmic results with some external knowledge, and intrinsic validation, which uses randomization to evaluate the significance of the signals discovered.

8.3.11 Comparative analysis

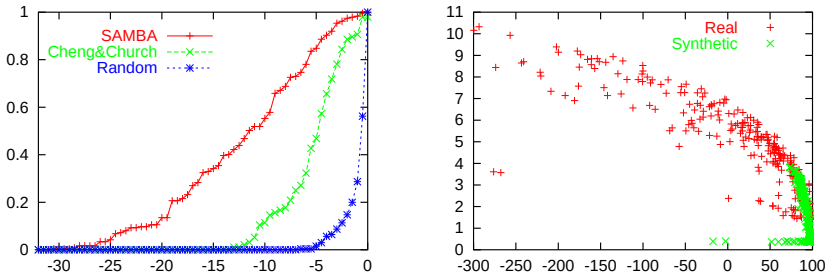
One way for evaluating biclustering algorithms is by using prior biological knowledge as some form of a gold standard that we can compare to our set of biclusters. A *correspondence plot* depicts the distribution of p-values of the produced biclusters, using for evaluation a known (putatively correct) classification of conditions (e.g., to various cancer types) or a given gene annotation. We describe the plot when a classification is given. For each value of p on a logarithmic scale, the plot presents the fraction of biclusters whose p-value is at most p out of the (say) 100 best biclusters.

p-values are calculated according to the known classification as follows: Suppose prior knowledge partitions the m conditions into k classes, $C_1, \ldots, C_k$. Let B be a bicluster with b conditions, out of which b_j belong to class C_j. The p-value of B, assuming its most abundant class is C_i, is calculated as $p(B) = \sum_{k=b_i}^{b} \binom{|C_i|}{k}\binom{m-|C_i|}{b-k}/\binom{m}{b}$. Hence, the p-value measures the probability of obtaining at least b_i elements from the class in a random set of size b. One should note, that high quality biclusters can also identify phenomena that are not covered by the given classification. Nevertheless, we expect a large fraction of the biclusters to conform to the known classification.

As an example for the usage of correspondence plots, we present the analysis of outputs from the two algorithms we have described so far. Running on the same dataset (the lymphoma data of [3]), a collection of biclusters from both algorithm were analyzed using the known classification of conditions to different clinical types (DLBCL, CLL, FL and more). The results (Figure 8.7) clearly indicate that SAMBA's biclusters are much more aligned with the biological information.

8.3.12 Intrinsic validation

A second, important method for validating the quality of biclusters is by analyzing the results on random data sets. We should make sure that the results we consider as statistically significant are not obtained from random data. The details of randomization may be critical to the integrity of such test. Assume we are using a uniformly random graph model (the simple model described above) and we randomize the data according to it. Then the artifact we have mentioned before, causing the identification of high degree nodes as biclusters would not be discovered since the random graph model will follow our originally restricting assumption.



A B

Figure 8.7: Source: [18]. A: Performance of different weighting schemes and algorithms. Correspondence plots for SAMBA, the algorithm of Cheng and Church [4], and random biclusters. Likelihood weights use $p_c = 0.9$. B: Scatter plots of significance values on synthetic and real data. x-axis: significance value, y-axis: bicluster weight.

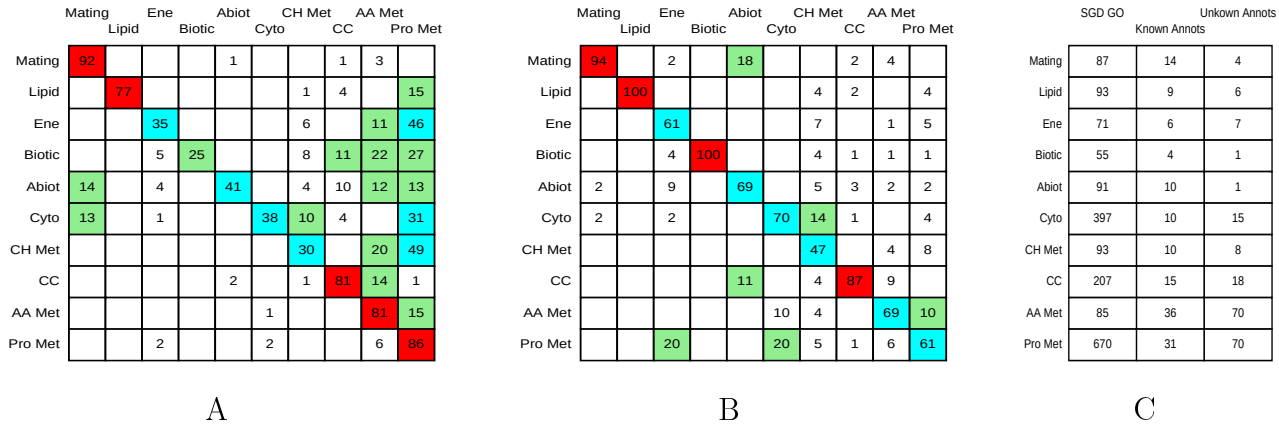
Figure 8.7(B) describes the results of a randomization test done on SAMBA using a degree preserving random graph model. The analysis was done on two datasets, first the real data was biclustered and the significance of each bicluster was calculated. Then the same procedure was done with a random graph preserving all vertex degrees. The scatter plot not only demonstrate that heavy biclusters are non random but also provides empirical evidence to the relation of the SAMBA likelihood score and the more formal significance measure.

8.3.13 Functional Annotation using biclusters

The output of a biclustering algorithm is a collection of significant local signals in the data. Such signals may be used in diverse application. One example for an application of biclusters is for automatic annotation of genes. The budding yeast, which is one of the most characterized eukaryotes up to date, contains about 6200 genes, only half of which have a known function. We can use a large database of gene expression and a set of derived biclusters to try and associate unknown genes with some function. The idea is simple, whenever a majority of the characterized genes in a bicluster share a common functional class, it is likely that the other genes in the bicluster are also related to this class.

A compiled dataset of yeast gene expression, including 515 conditions for the 6,200 yeast ORFs was used to test this idea. The data was collected from five different experiments [12, 7, 8, 17, 13]. Analysis by SAMBA generated 2,406 biclusters ranging over 4,961 genes and 515 conditions. The source for the known functional annotationaswas the SGD database [2], which includes 3000 annotations using the Gene Ontology [9] vocabulary.

The bicluster set was filtered to include only those biclusters in which more than 60% of their annotated members had the same class. Out of those, only biclusters that were func-



	Mating	Lipid	Ene	Biotic	Abiot	Cyto	CH Met	CC	AA Met	Pro Met
Mating	92				1			1	3	
Lipid		77					1	4		15
Ene			35				6		11	46
Biotic			5	25			8	11	22	27
Abiot	14		4		41		4	10	12	13
Cyto	13		1			38	10	4		31
CH Met							30		20	49
CC					2		1	81	14	1
AA Met						1			81	15
Pro Met			2			2			6	86

A

	Mating	Lipid	Ene	Biotic	Abiot	Cyto	CH Met	CC	AA Met	Pro Met
Mating	94		2		18			2	4	
Lipid		100					4	2		4
Ene			61				7		1	5
Biotic			4	100			4	1	1	1
Abiot	2		9		69		5	3	2	2
Cyto	2		2			70	14	1		4
CH Met							47		4	8
CC					11		4	87	9	
AA Met						10	4		69	10
Pro Met			20			20	5	1	6	61

B

	SGD GO	Known Annots	Unkown Annots
Mating	87	14	4
Lipid	93	9	6
Ene	71	6	7
Biotic	55	4	1
Abiot	91	10	1
Cyto	397	10	15
CH Met	93	10	8
CC	207	15	18
AA Met	85	36	70
Pro Met	670	31	70

C

Figure 8.8: Source: [18]. Yeast functional annotation. A: Annotation specificity. The table depicts the annotation accuracy measured using 70:30 cross-validation. Rows represent classes assigned using our method and columns represent SGD GO classes. Cell (x, y) contains the percentage of genes annotated x that belong to GO class y. Higher percentages are darker. B: Annotation sensitivity calculated w.r.t. annotated genes only. Cell (x, y) contains the percentage of SAMBA annotated genes that belong to GO class y and were annotated x. C: Annotation of unknown genes. The table shows for each functional class its size in the SGD GO annotation, the number of genes that belong to this class and were annotated by SAMBA, and the number of unknown genes assigned to this class by SAMBA. Abbreviations for functional classes: Mating - mating (sensu Saccharomyces, Fungi); Lipid - lipid metabolism; Ene - energy pathway; Biotic - response to biotic stimulus; Abiot - response to abiotic stimulus; Cyb - cytoplasm organization and biogenesis; CH Met - carbohydrate metabolism; CC - mitotic cell cycle; AA Met - amino acid and derivative metabolism; Pro Met - protein metabolism and modification.

tionally enriched (p-value below 10^{-4}) were used. The unannotated genes in those biclusters were now assigned to the most abundant class. Note that each gene may be annotated more than once, as is the case for the curated GO annotations. For cross validation, 100 runs were performed and in each one 30% of the annotations were hidden. The overall average success rate in annotating the hidden genes was calculated.

The results of these runs are summarized in Figure 8.8(A,B). Overall, 81.5% of the test set annotations matched those known from SGD, demonstrating tha we can extrapolate functional annotation using biclusters.

8.4 Plaid Models

We shall next describe a different methodology for gene expression analysis which in some aspects can be thought of as biclustering. The idea originates from statistics. We are introducing a linear model for the data, namely we assume that the matrix is a sum of "plaids" - signals which dominate some submatrix - and we are interested in finding a small set of such plaids such that the difference between their sum and the observed signal is only noise. Plaid models are general statistical tools for analysing of multivariate data. In this section, we will introduce these models, and then describe Lazzaroni and Owen's [14] algorithm for fitting an expression matrix to a plaid model. Finally, we will present some applications of plaid models to yeast expression data.

8.4.1 The Plaid Model

We already denote the input matrix of expression data as $A = (A_{ij})$. AS we already defined, $i = 1, ..., n$ is the set of genes and $j = 1, ..., p$ is the set of conditions. A_{ij} describes the level of expression of gene i in sample j. The number np of data points is very large, and meaningfull visualization of such a volume of data is very important. A natural starting point is to form a color image of the data on a n by p grid, with each cell colored according to the value of $A_{i,j}$. Figure 1 in [1] shows one such image for yeast data. The ordering of the rows and the columns in such images is usually arbitrary. It is natural then to consider ways of reordering the rows and columns in order to group together similar rows and similar columns, thus forming an image with blocks of similar color, which creates the 'plaid' pattern (for example, see figure 2 in [14]). A set of genes behaving similarly in a set of samples, define what is called a "layer", which is a synonym to "bicluster", but in the plaid model context.

An ideal reordering of the array would produce an image with some number K of rectangular blocks on the diagonal. Each block would be uniformly colored, and the part of the image outside of these diagonal blocks would be of a neutral background color. This ideal corresponds to the existence of K mutually exclusive clusters of genes and a corresponding of K-partition of the samples. Every gene in tht k'th gene-block is expressed within, and only within, its sample block k. Mathematically, the corresponding representation is

$$A_{i,j} = \mu_0 + \sum_{k=1}^{K} \mu_k \rho_{ik} \kappa_{jk} \tag{8.4}$$

where μ_0 is a background color, μ_k describes the color in block k, ρ_{ik} is a gene-block membership indicator variable, i.e., ρ_{ik} is 1 iff gene i belongs to the k'th gene-block and κ_{jk}, similarly, is a sample-block membership indicator variable. If overlapping layers are not allowed, we must add a constraint $\sum_k \kappa_{jk} = 1$ for all j, $\sum_k \rho_{ik} = 1$ for all i. The general model represents the data as a sum of possibly overlapping layers that do not have to cover the whole array and assigns several layes to cover the same gene.

The model in (8.4) describes a response μ_k that is shared by all genes in it for all samples in it. It would also be biologically interesting to identify a set of genes that had an identical, though not constant, response to a set of samples. Conversely a set of samples with a common expression pattern for a set of genes would be interesting. The following models support these notions:

$$A_{i,j} = \mu_0 + \sum_{k=1}^{K} (\mu_k + \alpha_{ik})\rho_{ik}\kappa_{jk} \tag{8.5}$$

$$A_{i,j} = \mu_0 + \sum_{k=1}^{K} (\mu_k + \beta_{jk})\rho_{ik}\kappa_{jk} \tag{8.6}$$

$$A_{i,j} = \mu_0 + \sum_{k=1}^{K} (\mu_k + \alpha_{ik} + \beta_{jk})\rho_{ik}\kappa_{jk} \tag{8.7}$$

where each $\rho_{ik} \in \{0, 1\}$ and each $\kappa_{jk} \in \{0, 1\}$. If α_{ik} or β_{jk} is used, then $\sum_i \rho_{ik}\alpha_{ik} = 0$, $\sum_j \kappa_{jk}\beta_{jk} = 0$ to avoid over parameterization. Here we can get a deeper understanding of the plaid model image pattern, which is a plot of $\mu_k + \alpha_{ik} + \beta_{jk}$.

Each of the models (8.4) to (8.7) approximates the image by a sum of layers. We may use the notation θ_{ijk} to represent either μ_k, $\mu_k + \alpha_{ik}$, or $\mu_k + \alpha_{ik} + \beta_{jk}$, as needed. We will get a little more generality by mixing layer types, so that α_{ik} or β_{ik} might appear in some but not all θ_{ijk}. The model then may be written as a sum of layers:

$$A_{i,j} = \mu_0 + \sum_{k=1}^{K} \theta_{ijk}\rho_{ik}\kappa_{jk} \tag{8.8}$$

Note that the resulting model is a superposition of the two-way analysis of variance models (see [16]), each defined over a subset of genes and samples.

8.4.2 Estimating Parameters

In the plaid model regression problem, we seek a plaid model with a small value of

$$\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{p}(A_{ij} - \theta_{ij0} - \sum_{k=1}^{K} \theta_{ijk}\rho_{ij}\kappa_{jk})^2 \tag{8.9}$$

For each layer k, there are $(2^n - 1)(2^p - 1)$ ways to select the participating genes and conditions, and the problem in general is NP-hard (Lazzaroni and Owen do not prove it) so we revert to numerical heuristics. The heuristic solution of Lazzaroni and Owen is the following: New layers will be added to the model one at a time. Suppose we have fixed the first $K-1$ layers and we are seeking for the K'th layer to minimize the sum of squared error. Let

$$Q = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{p}(Z_{ij}^{(K-1)} - \theta_{ijK}\rho_{iK}\kappa_{jK})^2 \tag{8.10}$$

where

$$Z_{ij}^{(K-1)} = A_{ij} - \theta_{ij0} - \sum_{k=1}^{K-1} \theta_{ijk}\rho_{ij}\kappa_{jk} \tag{8.11}$$

is the residual from the first $K-1$ layers.

The regression algorithm uses an iterative approach in which each step optimize θ values, ρ values or κ values while fixing the two other parameter families. It is also instrumental to consider ρ and κ values in a continuous range, only forcing them to take values 0 and 1 in the last one or several iterations. At intermediate stages, the values of θ_{ijk} describe a "fuzzy analysis of variance" in which ρ_{iK} and κ_{jK} are not necessarily 0 and 1. Let $\theta^{(s)}$ denote all of the θ_{ijK} values at iteration s. Similarly, let $\rho^{(s)}$ and $\kappa^{(s)}$ represent all of the ρ_{iK} and κ_{jK} values at iteration s. The starting values for s=0 are randomly generated near 0.5 (more detailes about initialization can be found in [14]).

Two questions are to be answered:

- How to perform the θ, ρ or κ optimization at each iteration?
- What is the desired total number of layers?

We will answer these questions in the next subsections.

Updating θ_{ijK}, κ_{jK}, ρ_{iK}

To update the θ_{ijK} for each K, we have to minimize:

$$Q^{(K)} = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{p}(Z_{ij}^{(K-1)} - (\mu_K + \alpha_{iK} + \beta_{jK})\rho_{iK}\kappa_{jK})^2 \tag{8.12}$$

subject to the conditions: $\sum_{i=1}^{n}\rho_{iK}^2\alpha_{iK} = \sum_{j=1}^{p}\kappa_{jK}^2\beta_{jK} = 0$. Straightforward Langrange multiplier arguments show that

$$\mu_K = \frac{\sum_i\sum_j \rho_{iK}\kappa_{jK}Z_{ij}^{K-1}}{(\sum_i \rho_{iK}^2)(\sum_j \kappa_{jK}^2)} \tag{8.13}$$

$$\alpha_{iK} = \frac{\sum_j(Z_{ij}^{(K-1)} - \mu_K\rho_{iK}\kappa_{jK})\kappa_{jK}}{\rho_{iK}\sum_{jK}\kappa_{jK}^2} \tag{8.14}$$

$$\beta_{jK} = \frac{\sum_i(Z_{ij}^{(K-1)} - \mu_K\rho_{iK}\kappa_{jK})\rho_{iK}}{\kappa_{jK}\sum_{iK}\rho_{iK}^2} \tag{8.15}$$

So, in iteration s, we use these equations to update $\theta^{(s)}$ from $\rho^{(s-1)}$ and $\kappa^{(s-1)}$. Note that the update for μ_K is the same whether or not the K'th layer include α_{iK} or β_{iK}. The

updated α_{iK} and β_{iK} are the same whether or not the other is included in the layer. The values for ρ_{iK} and κ_{jK} that minimize Q are:

$$\rho_{iK} = \frac{\sum_j \theta_{ijK}\kappa_{jK}Z_{ij}^{K-1}}{\sum_j \theta_{ijK}^2\kappa_{jK}^2} \tag{8.16}$$

$$\kappa_{jK} = \frac{\sum_i \theta_{ijK}\rho_{iK}Z_{ij}^{K-1}}{\sum_i \theta_{ijK}^2\rho_{iK}^2} \tag{8.17}$$

At iteration s, we use these equations to update $\rho^{(s)}$ from $\theta^{(s)}$ and $\kappa^{(s-1)}$, and update $\kappa^{(s)}$ from $\theta^{(s)}$ and $\rho^{(s-1)}$. Note that genes and conditions are treated symmetrically in this iteration, and the updates will be the same if they were done in the opposite order.

After we update the parameters for the K'th layer in iterations $s = 1, ..., S$, (S is the number of iterations, and according to Lazzaroni and Owen, the algorithm does not appear to be very sensitive to the choice of S), we start evaluating the parameters of the $(K+1)$'th layer iteratively in the same way. For each K, we check whether a stopping criterion holds for the layer or not, as detailed below.

Stopping rule

A greedy algorithm that adds one layer at a time requires a stopping rule. We suppose that in higher K values, the residual becomes more and more similar to a noise. In order to avoid adding layers that contain unstructured noise, Lazzaroni and Owen (2000) used the following stopping criterion:

First they define the *importance* of layer k by $\sigma_k^2 = \sum_{i=1}^n \sum_{j=1}^p \rho_{ik}\kappa_{jk}\theta_{ijk}^2$. The algorithm accepts a layer if it has significantly larger *importance* than in noise. The distribution of σ_k^2 on noise is unknown. In order to evaluate σ_k^2 on noise, they randomly permute the data:

- Let Z_{ij} be the residual matrix in which we search for layer k.
- For each $r = 1, ..., R$, let $Z_{ij}^{(r)}$ be a matrix obtained by randomly permuting every row and column of Z_{ij}. All permutation are independent and uniformly distributed.
- Let $\sigma_k^{2,r}$ denote the *importance* of layer k found on the randomized data $Z_{ij}^{(r)}$.
- If $\sigma_k^2 < \max_{1 \le r \le R} \sigma_k^{2,r}$ and $k < K_{max}$ add the new layer k to the model.

8.4.3 Experiments

Web Figure 1 shows yeast gene expression data used by Eisen et al. (1998) [6]. The data are available at [1]. The columns represent timepoints within each of ten experimental series. Each of the 2467 rows represents a single probe on the microarray designed to detect

the expression level of particular gene. After 34 layers, the stopping criterion was met, and the algorithm was unable to find another meaningful layer. Overall, the 34 layers and background contained 5568 parameters, fewer than 3% of the number of observations. Web Figure 2 shows the complete fitted model. There was a little overlapping among the layers, with fewer the 1% of the data falling into more then one layer. As expected, the plaid model consistently put columns from the same experimental series together within the same layer. Moreover, certain gene classes concentrate in a single layer (for example, genes involve in cell cycle, glycolysis and ribosomal genes).

Bibliography

[1] http://www-stat.stanford.edu/~owen/reports/.

[2] http://genome-www.stanford.edu/saccharomyces/.

[3] A.A. Alizadeh et al. Distinct types of diffuse large B-cell lymphoma identified by gene expression profiling. *Nature*, 403(6769):503–511, 2000.

[4] Y. Cheng and G.M. Church. Biclustering of expression data. In *Proc. ISMB'00*, pages 93–103. AAAI Press, 2000.

[5] M.H. DeGroot. *Probability and Statistics*. Addison-Wesley, 1989.

[6] 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*, 95(25):14863–14868, 1998.

[7] A. P. Gasch et al. Genomic expression programs in the response of yeast cells to environmental changes. *Mol Biol Cell*, 11:4241–57, 2000.

[8] A.P. Gasch et al. Genomic expression responses to DNA-damaging agents and the regulatory role of the yeast ATR homolog mec1p. *Mol. Biol. Cell*, 12(10):2987–3003, 2001.

[9] Gene Ontology Consortium. Gene Ontology: tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

[10] J.A. Hartigan. *Clustering Algorithms*. John Wiley and Sons, 1975.

[11] D.S. Hochbaum. Approximating clique and biclique problems. *Journal of Algorithms*, 29(1):174–200, 1998.

[12] TR. Hughes et al. Functional discovery via a compendium of expression profiles. *Cell*, 102:109–26, 2000.

[13] T. Ideker et al. Integrated genomic and proteomic analyses of a systematically perturbed metabolic network. *Science*, 291:929–34, 2001.

[14] L. Lazzaroni and A. Owen. Plaid models for gene expression data. *Statistica Sinica*, 12(1):61–86, 2002.

[15] R. Peeters. The maximum edge biclique problem is NP-complete. citeseer.nj.nec.com/peeters00maximum.html.

[16] R.R. Sokal and F.J. Rohlf. *Biometry*. W. H. Freeman and company, New-York, third edition, 1995.

[17] P. T. Spellman, G. Sherlock, et al. Comprehensive identification of cell cycle-regulated genes of the yeast Saccharomyces cerevisiae by microarray hybridization. *Mol. Biol. Cell*, 9:3273–3297, 1998.

[18] A. Tanay, R. Sharan, and R. Shamir. Biclustering gene expresion data. Submitted for publication, 2002.

[19] S. Tavazoie, JD. Hughes, MJ. Campbell, RJ. Cho, and GM. Church. Systematic determination of genetic network architecture. *Nature Genetics*, 22:281–285, 1999.