Analysis of Gene Expression Data **Spring Semester, 2007**

Lecture 4: March 22, 2007

Lecturer: Ron Shamir *Scribe: Gil Hoffer and Tal Peleg*[1]

4.1 The CLICK algorithm

4.1.1 Introduction

CLICK (CLuster Identification via Connectivity Kernels) is a graph-based algorithm for clustering [16]. The input for CLICK is the gene expression matrix. Each row of this matrix is an "expression fingerprint" for a single gene. The columns are specific conditions under which gene expression is measured.

The CLICK algorithm attempts to find a partitioning of the set of elements into clusters, so that two criteria are satisfied: *homogeneity* - pairs of elements from the same cluster, called *mates*, are highly similar to each other; and *separation* - pairs of elements from different clusters, called *non-mates*, have low similarity to each other.

The goal is to identify highly homogeneous sets of elements - *connectivity kernels*, which are subsets of very similar elements. The remaining elements are added to the kernels by the similarity to average kernel fingerprints.

We will make use of tools from graph theory and use probabilistic considerations.

4.1.2 Probabilistic Model

We'll define *mates* as pairs of genes that belong to the same true cluster.

The CLICK algorithm makes the following assumptions:

1. Similarity values between mates are normally distributed with parameters μ_T, σ_T.
2. Similarity values between non-mates are normally distributed with parameters μ_F, σ_F.
3. We expect that $\mu_T > \mu_F$, and that σ_T, σ_F are small enough such that the clusters are distinguishable.

[1]Based in part on a scribe by David Shafrir, May 2005, and on a scribe by Orly Stettiner and Ron Gabor, December 2001

4. We assume that similarity values are mutually independent, and not dependent on specific genes and/or clusters.

These assumptions are justified empirically by simulations, and in some cases theoretically (by the Central Limit Theorem).

Parameters for the algorithm can be learned in two ways: from partially known solutions, or estimated using the EM (Expectation-Maximization) algorithm [13].

4.1.3 The Basic CLICK Algorithm

The CLICK algorithm represents the input data as a weighted *similarity graph* $G = (V, E)$. In this graph, vertices correspond to elements and edge weights are derived from the similarity values (Figure 4.1). The weight w_{ij} of an edge (i, j) reflects the probability that i and j are mates, and is set to be:

$$w_{ij} = \ln \frac{p\, f^M(S_{ij})}{(1-p)\, f^N(S_{ij})}$$

where p is the probability of two genes being mates, and $f^M(S_{ij})$ ($f^N(S_{ij})$) is the value of the probability density function for mates (non-mates) for S_{ij}:

$$f^M(S_{ij}) = \frac{1}{\sqrt{2\pi}\sigma_T} e^{-\frac{(S_{ij}-\mu_T)^2}{2\sigma_T^2}} \quad , \quad f^N(S_{ij}) = \frac{1}{\sqrt{2\pi}\sigma_F} e^{-\frac{(S_{ij}-\mu_F)^2}{2\sigma_F^2}}$$

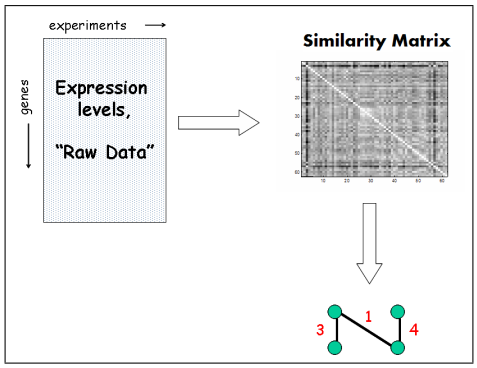

Figure 4.1: The raw data is translated into a matrix of similarity values. This matrix is then translated into a Similarity Graph.

The basic CLICK algorithm is described in Figure 4.2 and exemplified in Figure 4.3. The main concepts of the algorithm are as follows: given a connected graph G, we would like to decide whether $V(G)$ is a subset of some true cluster, or $V(G)$ contains elements from at least two true clusters. In the first case we say that G is *pure*. In order to make this decision, we test the following two hypotheses for each cut C in G:

- H_0^C: C contains only edges between non-mates.
- H_1^C: C contains only edges between mates.

G is declared a *kernel* if H_1 is more probable for all cuts. The decision of whether G is a kernel relies on the following theorem:

Theorem 4.1 *G is a kernel iff the weight of $MinCut(G) > 0$.*

Proof: First, we'll use the assumption that the $S_{i,j}$-s are independent, and that the mate relations are also independent.

Then, using Bayes' Theorem, it can be shown that for any cut C in G

$$\ln \frac{Pr(H_1|C)}{Pr(H_0|C)} = \ln \frac{Pr(H_1)f(C|H_1)}{Pr(H_0)f(C|H_0)} = \ln \frac{p^{|C|}\prod_{i,j\in C} f^M(S_{ij})}{(1-p)^{|C|}\prod_{i,j\in C} f^N(S_{i,j})}$$
$$= \sum_{i,j\in C} \ln \frac{pf^M(S_{i,j})}{(1-p)f^N(S_{i,j})} = \sum_{i,j\in C} w_{ij} = W(C)$$

Obviously, $W(C) > 0$ **iff** $Pr(H_1^C|C) > Pr(H_0^C|C)$. If the minimum cut is positive, then obviously so are all other cuts. Conversely, if the minimum cut is non-positive, then for that cut $Pr(H_1^C|C) \leq Pr(H_0^C|C)$, therefore G is not a kernel. ■

```
Basic-CLICK(G(V, E))
    if (V(G) = {v}) then
        move v to the singleton set R
    elseif (G is a kernel) then
        Output V(G)
    else
        (H,H̄, cut) ← MinWeightCut(G)
        Basic-CLICK(H)
        Basic-CLICK(H̄)
    end if
end
```

Figure 4.2: The Basic-CLICK algorithm

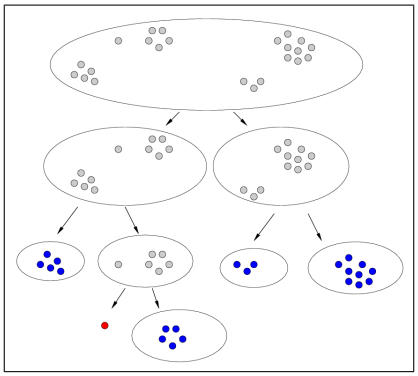

Figure 4.3: Basic scheme of the CLICK algorithm. Split subsets of G, that contain elements from two kernels.

Removing Negative Weight Edges The MIN-CUT problem for a weighted graph with both positive and negative edges is NP-Complete[2]. In order to use the efficient MIN-CUT algorithms we can use the following heuristic: We find the minimal cut efficiently while ignoring all negative edges. When we calculate the total weight of the cut, we fix it by adding the weight of the ignored edges which participate in it. We consider this weight as the minimal cut weight (although this is generally not true).

[2]MIN-CUT can be proved to be NP-Complete by reduction from MAX-CUT [5, page 210].

Refinements

The Basic-CLICK algorithm divides the graph into kernels and singletons. These kernels are expanded to the full clustering, using several refinements, while using both fingerprints and similarity values:

Adoption Step In practice, "true" clusters are usually larger than just the kernel. To accommodate this, in the refined algorithm, kernels "adopt" singletons to create larger clusters. This is done by searching for a singleton v and a kernel K, whose pairwise fingerprint similarity is maximum among all pairs of singletons and kernels. The refined algorithm iteratively applies the adoption step and then the Basic-CLICK algorithm on the remaining singletons, stopping when there are no more changes.

Cleaning Step In this step we remove nodes having a low degree. These nodes are usually not "interesting", since we are mostly interested in finding large kernels whose nodes have high degrees. By removing the low degree nodes, we ensure the algorithm will not waste time removing them one by one, thus making it more efficient.

Merge Step In this step we merge clusters whose fingerprints are similar (in practice, clusters can contain multiple kernels). The merging is done iteratively, each time merging two clusters whose fingerprint similarity is the highest (provided that the similarity exceeds a predefined threshold).

4.1.4 Quality Assessment

When the "correct" solution for the clustering problem is known, we can evaluate the algorithm's performance using comparison criteria . The criteria used in [16] are the Jaccard coefficient and the Minkowski coefficient. Let S, T be two clustering solutions. We mark by n_{11} the number of pairs of elements that are mates in both S and T, n_{01} is the number of pairs that are mates only in S, and n_{10} is the number of pairs that are mates only in T. The Jaccard coefficient is defined by:

$$J = \frac{n_{11}}{n_{11} + n_{10} + n_{10}}$$

and the Minkowski coefficient is defined by:

$$M = \sqrt{\frac{n_{01} + n_{10}}{n_{11} + n_{10}}}$$

One should note that the Jaccard coefficient denotes an optimal solution when its value is 1, while the quality of the solution improves as the value of the coefficient increases. On the other hand, the Minkowski coefficient denotes an optimal solution when its value is 0, while the quality of the solution improves as the value of the coefficient decreases.

Unfortunately, in most cases the "correct" solution for the clustering problems is unknown. In these cases we evaluate the quality of the solution by computing two figures of merit to measure the *homogeneity* and *separation* of the produced clusters. For fingerprint data, homogeneity is evaluated by the average and minimum correlation coefficient between the fingerprint of an element and the fingerprint of its corresponding cluster. Separation is evaluated by the weighted average and the maximum correlation coefficient between cluster fingerprints. Formally:

Definition Homogeneity and separation measures:
We define the fingerprint of a set of elements to be the mean vector of the fingerprints of the members of the set. Let $X_1, ..., X_t$ be clusters, $C(u)$ be the cluster of vertex u, $F(X)$ and $F(u)$ be the fingerprints of a cluster X and of element u respectively, and let $S(x, y)$ denote the similarity between fingerprints x and y, then:

Average Homogeneity

$$H_{Ave} = \frac{1}{|N|} \sum_{u \in N} S(F(u), F(C(u)))$$

Minimum Homogeneity

$$H_{Min} = \min_{u \in N} S(F(u), F(C(u)))$$

Average Separation

$$S_{Ave} = \frac{1}{\sum_{i \neq j} |X_i||X_j|} \sum_{i \neq j} |X_i||X_j| S(F(X_i), F(X_j))$$

Maximum Separation

$$S_{Max} = \max_{i \neq j} S(F(X_i), F(X_j))$$

Logically, a clustering improves when H_{Ave} and H_{Min} increase, and when S_{Ave} and S_{Max} decrease.

Another method of quality assessment is setting a certain similarity threshold and measuring the fraction of mates and non-mates above that threshold.

4.1.5 Algorithm Performance Comparisons

This section contains examples of comparisons between CLICK and other clustering algorithms, in various problems, including expression data, oligo-fingerprinting data and protein similarity data (Tables 4.1, 4.2, 4.3, 4.4, 4.5, 4.6 and Figures 4.4, 4.5). Analysis of the results (see Table 4.7) shows that CLICK outperforms all the compared algorithms in terms of quality. In addition, CLICK is very fast, allowing clustering of thousands of elements in minutes, and over 100,000 elements in a couple of hours on a regular workstation. Figure 4.6 shows the result of a comparison in which the authors of each clustering algorithm were allowed to run the test on their own. The graph shows a trade off between the homogeneity and separation scores; The further the algorithm is from the origin the "better" its overall performance.

In addition, CLICK was tested in simulations which included varying cluster structures and different distribution parameters. Similarity values for mates and non-mates were distributed normally: for each cluster structure, standard deviation σ was set at 5 for both mates and non-mates, while the difference between the means of mates μ_T and non-mates μ_F was set at $t \times \sigma$ for $t = 2, 1, 0.8, 0.6$. Results are shown in Table 4.8, evaluated using the Jaccard coefficient. As expected, the larger the distance between the means of mates and non-mates, the better the performance of the algorithm. It also seems that better results are obtained when cluster sizes are larger.

Program (algorithm)	# Clusters	Homogeneity		Separation	
		H_{Ave}	H_{Min}	S_{Ave}	S_{Max}
CLICK	30	0.8	-0.19	-0.07	0.65
GENECLUSTER	30	0.74	-0.88	-0.02	0.97

Table 4.1: A comparison between CLICK and GENECLUSTER [18] on the yeast cell-cycle dataset [3]. Expression levels of 6,218 *S. cerevisiae* genes, measured at 17 time points over two cell cycles.

Program	#Clusters	Homogeneity		Separation	
		H_{Ave}	H_{Min}	S_{Ave}	S_{Max}
CLICK	10	0.88	0.13	-0.34	0.65
Hierarchical	10	0.87	-0.75	-0.13	0.9

Table 4.2: A comparison between CLICK and Hierarchical clustering [4] on the dataset of response of human fibroblasts to serum [10]. Human fibroblast cells starved for 48 hours, then stimulated by serum. Expression levels of 8,613 genes were measured at 13 time points.

Program	#Clusters	#Singletons	Minkowski	Jaccard	Time(min)
CLICK	31	46	0.57	0.7	0.8
HCS	16	206	0.71	0.55	43

Table 4.3: A comparison between CLICK and HCS on the blood monocytes cDNA dataset [8]. 2,329 cDNAs purified from peripheral blood monocytes, fingerprinted with 139 oligos. Correct clustering is known from back hybridization with long oligos.

Program	#Clusters	#Singletons	Minkowski	Jaccard	Time(min)
CLICK	2,952	1,295	0.59	0.69	32.5
K-Means	3,486	2,473	0.79	0.4	–

Table 4.4: A comparison between CLICK and K-means [9] on the sea urchin cDNA dataset. $20,275$ cDNAs purified from sea urchin eggs, and fingerprinted with 217 oligos. Correct clustering of 1,811 cDNAs is known from back hybridizations.

	Minkowski	Jaccard
CLICK	0.88	0.39
ProtoMap	0.89	0.39

Table 4.5: A comparison between CLICK and ProtoMap [20] on a dataset of 72,623 proteins. Correct clustering of 17,244 single domain proteins is known.

Program	#Clusters	#Singletons	Homogeneity	Separation
CLICK	9,429	17,119	0.24	0.03
SYSTERS	10,891	28,300	0.14	0.03

Table 4.6: A comparison between CLICK and SYSTERS on a dataset of 117,835 proteins [11]. Measures are based on similarity when no correct solution is known: For a fixed threshold t, homogeneity is the fraction of mates with similarity above t, and separation is the fraction of non-mates with similarity above t.

Elements	Problem	Compared to	Time(min)	Improvement
517	Gene Expression Fibroblasts	Cluster [4]	0.5	Yes
826	Gene Expression Yeast cell cycle	GeneCluster [18]	0.2	Yes
2,329	cDNA OFP Blood Monocytes	HCS [8]	0.8	Yes
20,275	cDNA OFP Sea urchin eggs	K-Means [9]	32.5	Yes
72,623	Protein similarity	ProtoMap [20]	53	Minor
117,835	Protein similarity	SYSTERS [11]	126.3	Yes

Table 4.7: A Summary of the time performance of CLICK on the above mentioned datasets. CLICK was executed on an SGI ORIGIN200 machine utilizing one IP27 processor. The time does not include preprocessing time. The "Improvement" column describes whether the solution of the CLICK algorithm was better than the compared algorithm.

Structure	2σ	1σ	0.8σ	0.6σ
6×50	1	1	0.98	0.85
10×30	1	0.96	0.71	0.1
$10, ..., 80$	1	1	0.97	0.83

Table 4.8: CLICK simulation results (mean Jaccard score over 20 runs). The test included various cluster structures (rows) and distances between μ_T and μ_F (in each column, the distance appearing in the title was used as a factor of the standard deviation σ. The first column denotes a distance of 2σ, etc.).

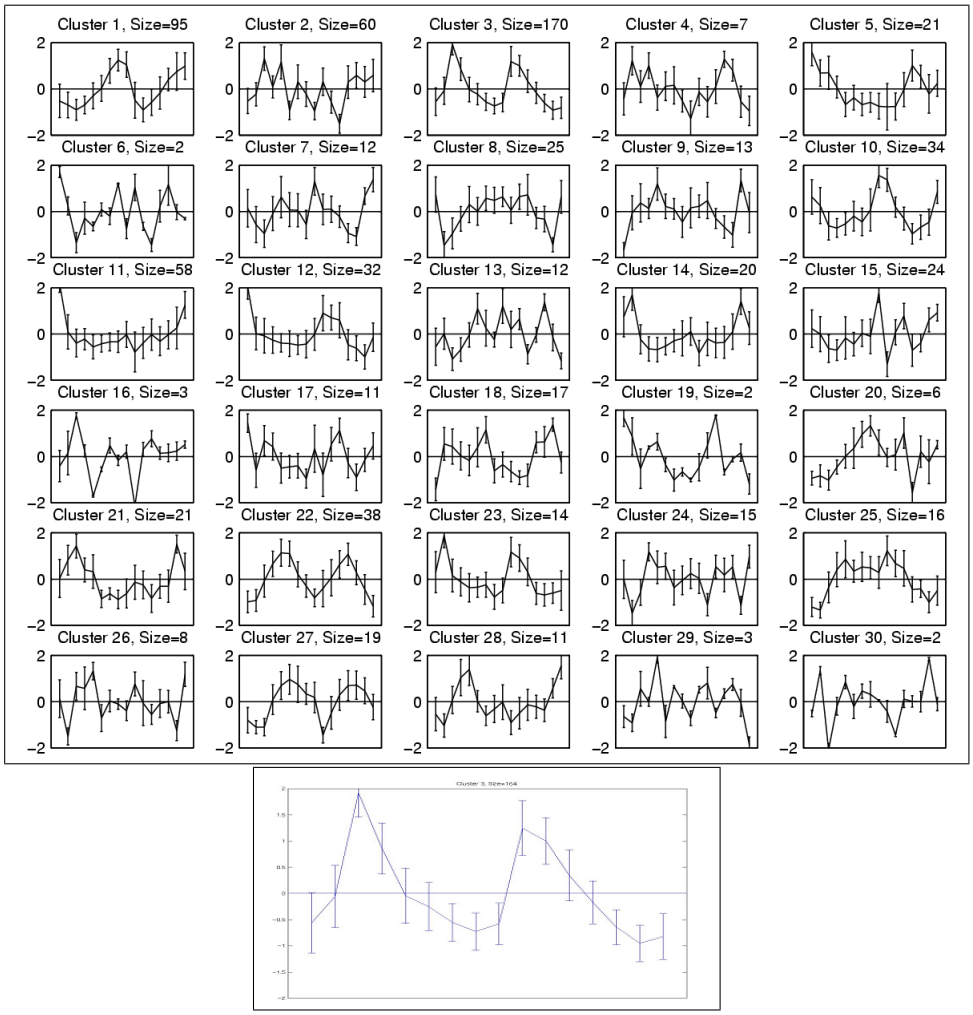

Figure 4.4: Source: [16]. CLICK's clustering of the yeast cell-cycle data [3]. x-axis: time points 0-80, 100-160 at 10-minute intervals. y-axis: normalized expression levels. The solid line in each sub-figure plots the average pattern for that cluster. Error bars display the measured standard deviation. The cluster size is printed above each plot. Cluster 3 (the late G1 Cluster) is shown in zoom in the lower image. The cluster found by CLICK contains 91% of the late G1-peaking genes. In contrast, in GeneCluster 87% are contained in 3 clusters.

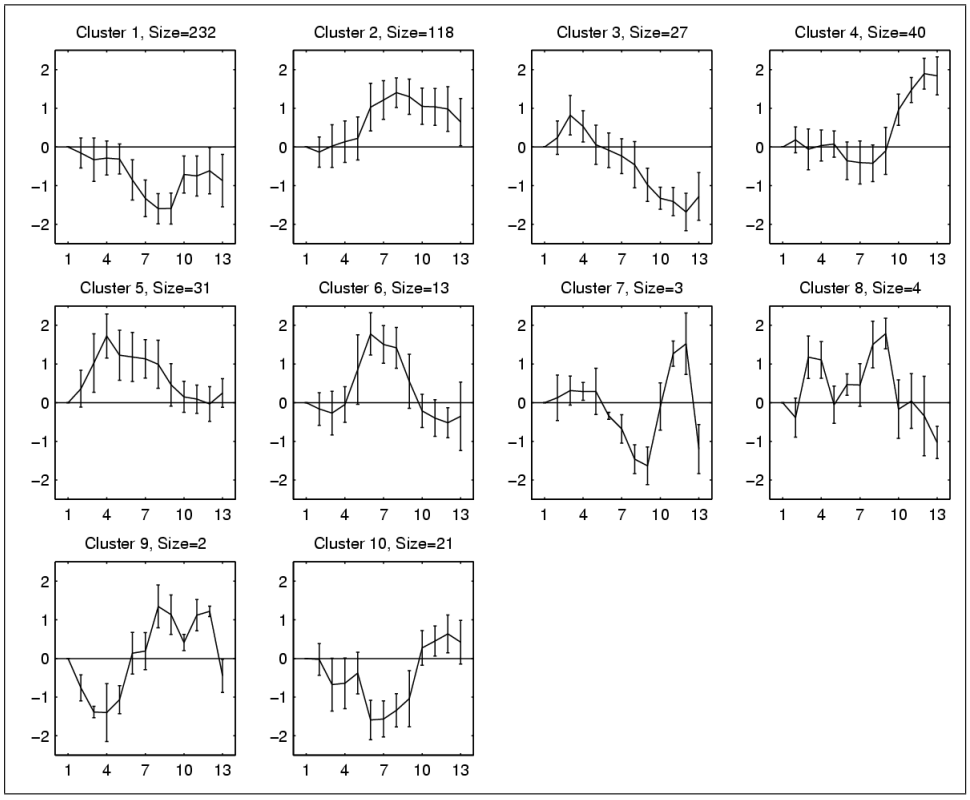

Figure 4.5: Source: [16]. CLICK's clustering of the fibroblasts serum response data [10]. x-axis: 1-12: synchronized time-points. 13: unsynchronized point. y-axis: normalized expression levels. The solid line in each sub-figure plots the average pattern for that cluster. Error bars display the measured standard deviation. The cluster size is printed above each plot.

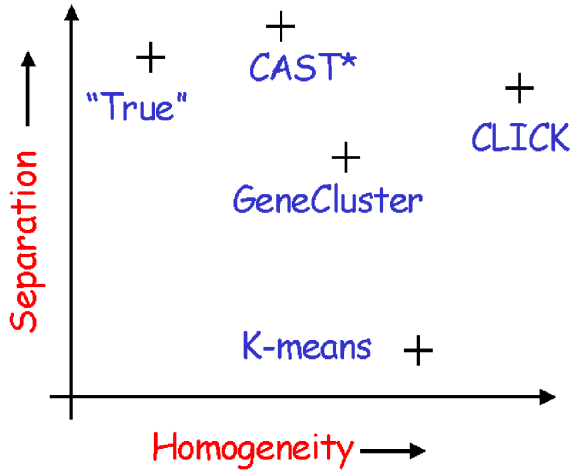

Figure 4.6: Comparison of clustering algorithms using homogeneity and separation criteria. The data consisted of 698 genes, 72 conditions [17]. Each algorithm was run by its authors in a "blind" test.

4.2 Hierarchical Clustering

An agglomerative approach attempts to place the input elements in a tree hierarchy structure in which the distance within the tree reflects element similarity. The elements are located at the leaves of the tree. Thus, the closer the elements are in the tree, the more similar they are.

Advantages of hierarchical methods :

1. A single coherent global picture.
2. Intuitive for biologists (similar representation is used in Phylogeny).

Disadvantages of hierarchical methods :

1. There is no explicit partition into clusters.
2. A human biologist with extensive knowledge might find it impossible to make sense of the data just by looking at the tree, due to the size of the data, and the number of errors.
3. The hierarchical structure is not natural for genes (as the basic rules of evolution do not apply to genes).
4. Forces all elements to fit a tree/dendrogram.

4.2.1 Hierarchical Representations

As was explained, a hierarchical representation uses a tree structure, in which the actual data is represented at the leaves. The tree can be either rooted or non-rooted. A particular tree representation is a *dendrogram*. The algorithms for hierarchical clustering recursively merge similar clusters, and compute the new distances between the merged cluster and all other clusters. This hierarchical tree-like structure implies that if i is clustered with j, and both are not similar to r, then $D(i,r) = D(j,r)$ even though $D(i,j) > 0$. ($D(n,m)$ is the distance function) (See Figure 4.8).

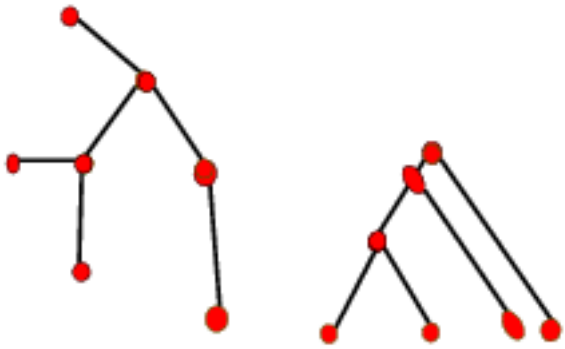

Figure 4.7: Hierarchical data represented as a rooted/non rooted tree

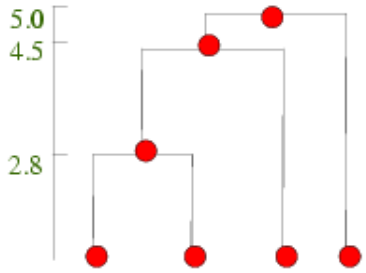

Figure 4.8: In a dendrogram, distances are represented on the y-axis. Denote the leaves a, b, c, d (from left to right). Then $D(a,b) = 2.8$, $D(a,c) = D(b,c) = 4.5$, $D(b,d) = D(c,d) = 5.0$

4.2.2 UPGMA Algorithm

UPGMA, or Unweighted Pair Group Method with Arithmetic mean [14], is a clustering procedure, which is simple and intuitive. It works by clustering the genes, at each stage merging two clusters, and at the same time creating a new node on a tree. The tree can be imagined as being assembled upwards, each node being added above the others and the edge lengths being determined by the difference in the heights of the nodes at the top and bottom of an edge.

UPGMA algorithm:

Let d be the distance function between gene expression fingerprints, we define the distance $D_{i,j}$ between two clusters of expressions C_i and C_j as follows:

$$D_{i,j} = \frac{1}{n_i + n_j} \sum_{p \in C_i} \sum_{q \in C_j} d(p, q)$$

where $n_i = |C_i|$ and $n_j = |C_j|$.

- Initialization:
  1. Initialize n clusters with the given expressions, one expression per cluster.
  2. Set the size of each cluster to 1: $n_i \leftarrow 1$.
  3. In the output tree T, assign a leaf for each expression.
- Iteration: combine two clusters to form a new cluster.
  1. Find the i and j that have the smallest distance D_{ij}.
  2. Create a new cluster – (ij), which has $n_{(ij)} = n_i + n_j$ members.
  3. Connect i and j on the tree to a new node, which corresponds to the new cluster (ij), and give the two branches connecting i and j to (ij) length $\frac{D_{i,j}}{2}$ each.
  4. Compute the distance from the new cluster to all other clusters (except for i and j, which are no longer relevant) as a weighted average of the distances from its components:

$$D_{(ij),k} = (\frac{n_i}{n_i + n_j}) D_{i,k} + (\frac{n_j}{n_i + n_j}) D_{j,k}$$

  5. Delete the columns and rows in D that correspond to clusters i and j, and add a column and row for cluster (ij), with $D_{(ij),k}$ computed as above.
  6. Return to 1 until there is only one cluster left.

Complexity: The time and space complexity of UPGMA is $O(n^2)$, since there are $n - 1$ iterations, with $O(n)$ work in each one.

Molecular Clocks and Ultrametric Property of distances

UPGMA produces a rooted tree of a special kind - a *clocklike*, or *ultrametric*, in which the total branch length from the root to any leaf is equal. In other words, there is a "molecular clock" that ticks in a constant pace, and all the observed expressions are at an equal number of ticks from the root. If there is a molecular clock (i.e., the solution is a clocklike tree), then UPGMA is guaranteed to return the optimal solution. Actually, UPGMA implicitly assumes the existence of an ultrametric tree, which explains why the new node, (ij), is the mean of the two nodes that were joined to create it, as shown in Figure 4.9. It is therefore not surprising that for substantially non-clocklike trees, the algorithm might give seriously misleading results.
As we do not assume that this attribute holds for gene-expression clustering data, the UPGMA algorithm is a problematic choice.

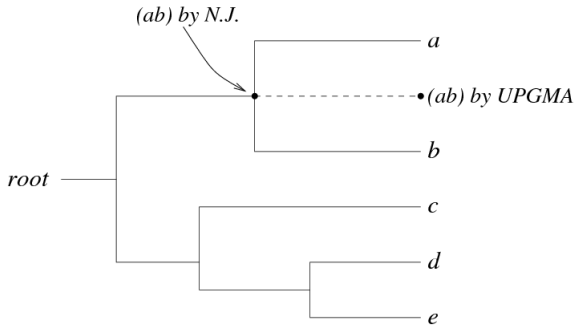

Figure 4.9: A clocklike tree, showing the clustering (ab) of the two nodes a and b by UPGMA and by the Neighbor-Joining algorithm.

4.2.3 Neighbor Joining Algorithm

This simple algorithm, based on neighbor merging, is due to Saitou and Nei [15]. The input matrix is the distance matrix between elements. Initially each element is a cluster. At each iteration we merge similar elements, and compute the new distances for the merged elements. When the algorithm finishes, we represent the results as a tree in which similar elements are near.

The Neighbor Joining Algorithm :

1. Input : The distance matrix D_{ij}.
2. Find elements r,s such that $D_{rs} = \min_{ij}(D_{ij})$.
3. Merge clusters r,s.
4. Delete elements r,s, and add a new element t with :
$$D_{it} = D_{ti} = \frac{D_{ir} + D_{is} - D_{rs}}{2}$$
5. Repeat, until one element is left.
6. Present the hierarchy as a tree with similar elements near each other.

The reason to define the new element t the way we define it is explained by the following argument: We look at the tree in Figure 4.10 in which i, s and r are leaves, and t is the parent of s and r. The distance between two nodes is defined to be the weight of the path between them. Therefore the distance between the nodes i and t is:

$$D_{it} = c + d = \frac{(c + d + a) + (c + d + b) - (a + b)}{2} = \frac{D_{ir} + D_{is} - D_{rs}}{2}$$

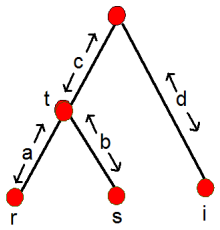

Figure 4.10: The graph that gives the rational behind the distance equation in the Neighbor Joining algorithm.

4.2.4 A General Framework

Lance and Williams, [12] also designed a general framework for hierarchical cluster merging algorithms. In their framework the distance calculating function is :

$$D_{it} = D_{ti} = \alpha_r D_{ir} + \alpha_s D_{is} + \gamma |D_{ir} - D_{is}|$$

In the Average Linkage algorithm :

$$\gamma = 0$$
$$\alpha_r = \frac{n_r}{n_r + n_s}$$
$$\alpha_s = \frac{n_s}{n_r + n_s}$$

4.2.5 Hierarchical clustering of gene expression data

A series of experiments were performed on real gene expression data, by Eisen et al. [4]. The goal was to check the growth response of starved human fibroblast cells, which were given serum. 8,600 gene levels were monitored over 13 time-points (about two cell cycles). The original data of test-to-reference ratios was first log-transformed, and then normalized, to have mean 0 and variance 1.
Formally:
t_{ij}– fluorescence levels of target gene i in condition j.
r_{ij}– same for reference.
$D_{ij} = \log(\frac{t_{ij}}{r_{ij}})$.
$D^*_{ij} = \frac{D_{ij} - E(D_i)}{std(D_i)}$ - the normalized levels of gene i in condition j. The similarity matrix was constructed from D^*_{ij} as follows :

$$S_{kl} = \frac{\sum_j D^*_{kj} \cdot D^*_{lj}}{N_{cond}}$$

Where N_{cond} is the number of conditions checked.

The Average Linkage method was applied on the similarity matrix, where genes with low activity (70%) were excluded from the analysis. The tree is presented by ordering the leaves according to increasing subtree weight (see Figure 4.11). The weight can be the average expression level, time of maximal induction, or any other.

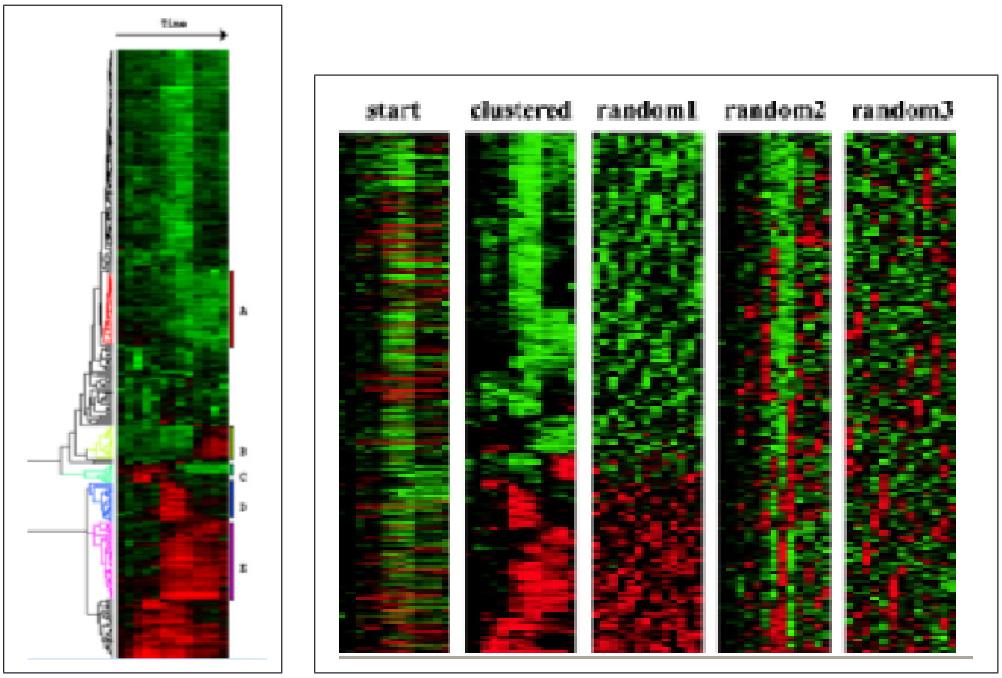

Figure 4.11: Source: [4]. The left image is the dendrogram resulting from the starved human fibroblast cells experiment. Five major clusters can be seen, along with many non clustered genes. The genes in the five groups serve similar functions : (A) cholesterol biosynthesis, (B) the cell cycle, (C) the immediate-early response, (D) signaling and angiogenesis, and (E) wound healing and tissue remodeling. In the right image, the same data is presented before and after random permutation of rows (random 1), of columns (random 2), and both (random 3). This should demonstrate the biological origins of patterns seen in the data. Indeed, the data is clustered visually, only when the "real" data was used (i.e. the "clustered" column).

4.3 Introduction to Biclustering

As we have seen in so far, the main method in the field of gene expression data analysis 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 have a common functionality 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, 6]) 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 types 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 identify 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 [7] and was applied to several domains before Cheng and Church [2] coined its usage in computational biology. Given a gene expression matrix, we search for sub-matrices that are tightly co-regulated according to some scoring criterion. We do not require the identified sub-matrices 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.

Before going into details, we state again the basic reasoning of using biclustering 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.

4.4 Cheng and Church's Algorithm

The first application of biclustering to gene expression data was the work of Cheng and Church [2]. 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. This process is repeated to produce a collection of biclusters.

4.4.1 The Algorithm

Notations

- $A = (a_{ij})$ - The input matrix of expression data.
- R - The row set.
- C - The column set.
- $A_{IJ}(I \subseteq R, J \subseteq C)$ - Sub-matrix
- $a_{Ij} = \frac{\sum_{i \in I} a_{ij}}{|I|}$ - Sub column average
- $a_{iJ} = \frac{\sum_{j \in J} a_{ij}}{|J|}$ - Sub row average
- $a_{IJ} = \frac{\sum_{i \in I, j \in J} a_{ij}}{|I||J|}$ - Sub-matrix average
- $RS_{IJ}(i,j) = a_{ij} - a_{Ij} - a_{iJ} + a_{IJ}$ - The *Residue Score* of an element a_{ij} in a submatrix A_{IJ}
- $H(I,J) = \sum_{i \in I, j \in J} \frac{RS_{ij}^2}{|I||J|}$ - The *Mean Residue Score* of the submatrix

Observations

- 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.
- A random sub-matrix (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 row/column addition/deletion. We apply the best operation if it improves the score and terminate 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 4.2 *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 :*

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

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 4.12).

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 4.13):

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 4.12: Single node deletion algorithm.

Lemma 4.3 *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 :*

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

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 4.13: Node addition algorithm.

This heuristic is not necessarily optimal for any situation. For example, the algorithm presented here is tailored for cases where the number of rows is much greater than the number of columns.

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).

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. One should note that this masking scheme lowers the chances of finding overlapping matrices. The general biclustering scheme is outlined in Figure 4.14.

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 4.14: Cheng-Church biclustering algorithm.

4.4.2 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 [1] 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. As we can see in table 4.9 , 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 $\delta - \textit{biclusters}$ may be far from random. The use of $\delta = 300$ is justified by the negligible percentage of large random δ-biclusters with a score lower than 300.

The results of the yeast data experiment by Cheng and Church were compared to the results of Tavazoie *et al.* ([19]). As we can see in figure 4.15, 12 biclusters that were found by

rows	columns	low	high	peak	tail
3	6	10	6870	390	15.5%
3	17	20	6600	480	6.83%
10	6	110	4060	800	0.0624%
10	17	240	3470	870	0.002%
30	6	410	2460	960	$< 10^{-6}$
30	17	480	2310	1040	$< 10^{-6}$
100	6	630	1720	1020	$< 10^{-6}$
100	17	700	1630	1080	$< 10^{-6}$

Table 4.9: Score distribution on random submatrices. The rows correspond to the number of rows and columns of the submatrices, the lowest and highest score, the score that was given the most, and the percentage of submatrices with score below 300.

Cheng and Church subdivide Cluster 2, found by Tavazoie *et al.*, into similar but different sub-clusters. They also include 10 genes from Cluster 14 of Tavazoie *et al.* and 6 from other clusters. All but one of the twelve clusters contain all 17 conditions, indicating that they form a tight cluster, with respect to the genes included in the table.

In the right side of Figure 4.15 we can see biclusters with lower number of conditions. They are also related to the Clusters of Tavazoie *et al.*. For example, Bicluster 93 corresponds to Cluster 9 of Tavazoie *et al.*. Some biclusters have less than half of the 17 conditions and thus represent shared patterns in many clusters discovered by using all the conditions.

In the second Experiment, Cheng and Church worked with data of human B-cell lymphomas ([1]) , consisting of 4026 genes and 96 conditions. One of their main hopes was to be able to differentiate between different types of cancer by examining biclusters with a relatively low number of conditions. In the left side of figure 4.16 we can see 12 of the biclusters found. In order to compare their result with the hierarchical clustering of Alizadeh et al., they considered how the found biclusters are divided by the major bipartition of the hierarchical clustering. They defined the mix of a bicluster by the percentage of the genes misclassified with respect to the bipartition. As we can see in the right side of figure 4.16, biclusters with few conditions or small row variance contain finer information than the one corresponding to the main bipartition.

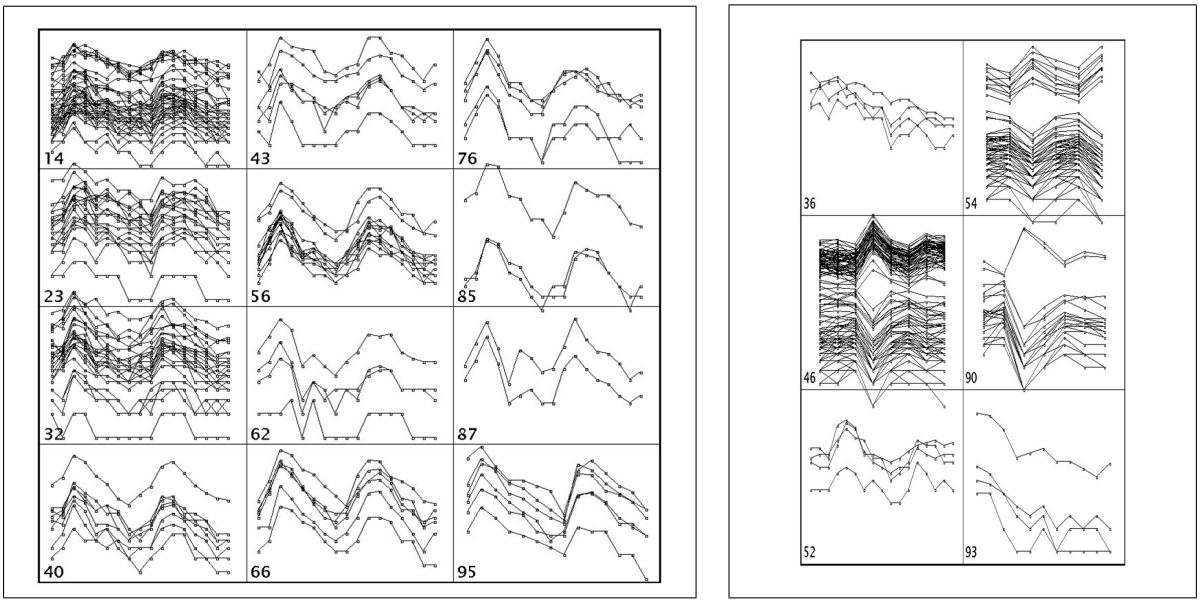

Figure 4.15: Source: [2]. In the left image are 12 biclusters with a large number of conditions, and in the right image there are 6 biclusters with a lower number of conditions. The biclusters numbers indicate the order in which they were discovered.

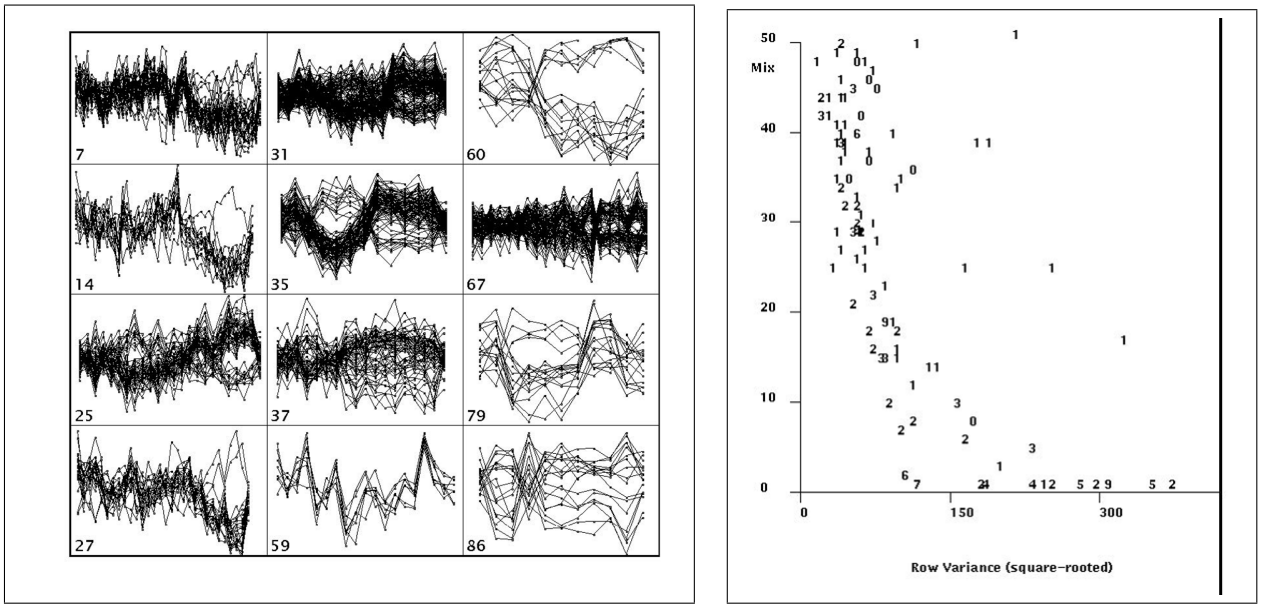

Figure 4.16: Source: [2]. In the left image are 12 biclusters discovered in the order labeled from the human expression data. Mirror images can be seen in most of these biclusters. In the right image is a plot of the first 100 biclusters in the human lymphoma data. Three measurement are involved. The horizontal axis is the square root of the row variance. The vertical axis is the mix. The digits used to represent the biclusters also indicate the size of the condition sets in the biclusters. The digit n indicates that the number of conditions is between $10n$ and $10(n+1)$ percent of the total number of conditions.

Bibliography

[1] A. A. Alizadeh, M. B. Eisen, R. E. Davis, C. Ma, I. S. Lossos, A. Rosenwald, J. C. Boldrick, H. Sabet, T. Tran, X. Yu, J. I. Powell, L. Yang, G. E. Marti, T. Moore, J. Hudson, L. Lu, D. B. Lewis, R. Tibshirani, G. Sherlock, W. C. Chan, T. C. Greiner, D. D. Weisenburger, J. O. Armitage, R. Warnke, and L. M. Staudt. Distinct types of diffuse large B-cell lymphoma identified by gene expression profiling. *Nature*, 403(6769):503–511, 2000.

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

[3] RJ. Cho et al. A genome-wide transcriptional analysis of the mitotic cell cycle. *Mol Cell*, 2:65–73, 1998.

[4] M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *PNAS*, 95:14863–14868, 1998.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., San Francisco, 1979.

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

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

[8] E. Hartuv, A. Schmitt, J. Lange, S. Meier-Ewert, H. Lehrach, and R. Shamir. An algorithm for clustering cDNA fingerprints. *Genomics*, 66(3):249–256, 2000.

[9] R. Herwig, A.J. Poustka, C. Muller, C. Bull, H. Lehrach, and J. O'Brien. Large-scale clustering of cDNA-fingerprinting data. *Genome Research*, 9:1093–1105, 1999.

[10] V.R. Iyer, M.B. Eisen, D.T. Ross, G. Schuler, T. Moore, J.C.F. Lee, J.M. Trent, L.M. Staudt, J. Hudson Jr., M.S. Boguski, D. Lashkari, D. Shalon, D. Botstein, and P.O. Brown. The transcriptional program in the response of human fibroblasts to serum. *Science*, 283 (1), 1999.

[11] A. Krause, J. Stoye, and M. Vingron. The systers protein sequence cluster set. *Nucleic Acids Research*, 28(1):270–272, 2000.

[12] G.N. Lance and W.T. Williams. A general theory of classification sorting strategies. 1. hierarchical systems. *The computer journal*, 9:373–380, 1967.

[13] G. J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. John Wiley and Sons, inc., 1997.

[14] C.D. Michener and R. R. Sokal. A quantitative approach to a problem in classifcation. *Evolution*, 11:136–162, 1957.

[15] N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987.

[16] R. Sharan and R. Shamir. CLICK: A clustering algorithm with applications to gene expression analysis. In *Proceedings of the 8th Annual International Conference on Intelligent Systems for Molecular Biology, (ISMB '00)*, pages 307–316. AAAI Press, 2000.

[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] P. Tamayo, D. Slonim, J. Mesirov, Q. Zhu, S. Kitareewan, E. Dmitrovsky, E. S. Lander, and T.R. Golub. Interpreting patterns of gene expression with self-organizing maps: Methods and application to hematopoietic differentiation. *PNAS*, 96:2907–2912, 1999.

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

[20] G. Yona, N. Linial, and M. Linial. Protomap: Automatic classification of protein sequences, a hierarchy of protein families, and local maps of the protein space. *Proteins: Structure, Function, and Genetics*, 37:360–378, 1999.