

A SURVEY OF CLUSTERING AND MOTIF FINDING
FOR MICROARRAY TECHNOLOGIES

By

MEGUMI IGARASHI

Bachelor of Science

Oklahoma State University

Stillwater, Oklahoma

2002

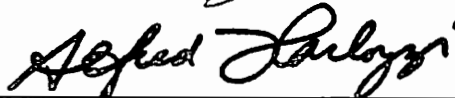
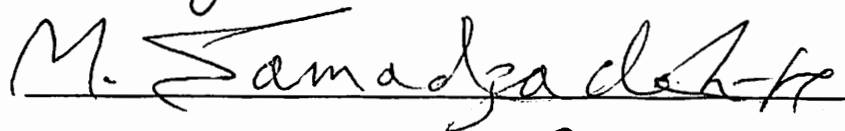
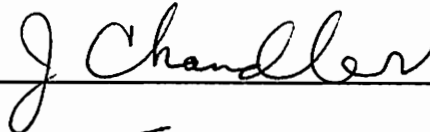
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July 2004

A SURVEY OF CLUSTERING AND MOTIF FINDING
FOR MICROARRAY TECHNOLOGIES

Thesis Approved:



Thesis Adviser



Dean of the Graduate College

PREFACE

Microarray technologies are widely used experimental techniques in functional genomic research. The experiments generate a large amount of data and efficient computational methods are required. Experimental data are clustered into similar biological functional groups and common subsequences (motifs) are detected among them to find subsequences that affect functions of the genomes.

This thesis is a survey of commonly used algorithms in the two processes.

Clustering algorithms: nearest neighbor clustering (*K*-means clustering, SOM (self-organizing map) and model-based clustering), agglomerative clustering (hierarchical clustering and quality-based clustering), divisive clustering (SOTA (self-organizing tree algorithm) and adaptive quality-based clustering).

Motif finding algorithms: string-based method, greedy algorithm, EM (expectation-maximization) algorithm, Gibbs sampling and Gibbs motif sampling.

Each algorithm's advantages, disadvantages and experimental results are discussed in each section. Finally, comparisons of the algorithms are made in a conclusion.

ACKNOWLEDGEMENT

I would like to express my appreciation to my thesis adviser, Dr. H. K. Dai for his patience, guidance and assistance. I would like to extend my appreciation to committee members Dr. J. P. Chandler and Dr. M. H. Samadzadeh.

Also, I am thankful to my family and friends for their encouragement.

TABLE OF CONTENTS

1. INTRODUCTION.....	1
2. MICROARRAYS	
2.1 Molecular Biology	4
2.2 DNA to Proteins.....	4
2.3 Microarray Experimental Method.....	6
2.3.1 cDNA Microarrays.....	7
2.3.2 Oligonucleotide Arrays	8
3. CLUSTERING	
3.1 Clustering Algorithms	9
3.2 General Description of EM Algorithm.....	10
3.3 Nearest Neighbor Clustering	
3.3.1 K-means Clustering	12
3.3.2 SOM (Self-organizing Map)	16
3.3.3 Model-Based Clusteing.....	19
3.4 Agglomerative Clustering	
3.4.1 Hierarchical Clustering	25
3.4.2 Quality-Based Clustering.....	28
3.5 Divisive Clustering	
3.5.1 SOTA (Self-Organizing Tree Algorithm).....	33
3.5.2 Adaptive Quality-Based Clustering	38

4. MOTIF FINDING	
4.1 Information Contents	45
4.2 String Method	47
4.3 Probablistic Method	
4.3.1 Greedy Algorithm	51
4.3.2 EM Algorithm	53
4.3.3 Gibbs Sampling	57
4.3.4 Gibbs Motif Sampling	60
5. CONCLUSION	66
6. REFERENCES	71
APPENDIX: Pseudocode Listings	74

LIST OF TABLES

Table 1	Currently Available Parameters in MCLUST	22
Table 2	Comparison of Clustering Algorithms	69
Table 3	Comparison of Motif Finding Algorithm	70

LIST OF FIGURES

Figure 1	Biological Process from DNA to Proteins	5
Figure 2	Overview of cDNA Microarray	7
Figure 3	<i>K</i> -means Clustering	15
Figure 4	Principle of SOMs	19
Figure 5	Example Result from the Experiment	21
Figure 6	Example of Hierarchical Clustering	28
Figure 7	Example of SOTA	38
Figure 8	Relocation of Cluster in Adaptive Quality-based Clustering	44
Figure 9	Example of Motif Candidates of Width 3	46
Figure 10	Transformation from M to M'	50
Figure 11	Example of Greedy Algorithm	53
Figure 12	Example of EM Algorithm	56
Figure 13	Example of Gibbs Sampling	59
Figure 14	Example of Gibbs Motif Sampling	63

1. INTRODUCTION

Studies of information science, statistics and computer science are applied to molecular biology to help biologists process large and complex data sets in their experiments and databases. This study is known as bioinformatics and its goal is a knowledge acquisition from genomic data: gene sequences, protein interactions and protein folding. This paper is a survey on some representative algorithms in microarray technologies that are commonly used for knowledge acquisition from gene sequences.

DNA (deoxyribonucleic acid) sequencing technology has been improving greatly and the entire human genome was sequenced in 2001. The number of known gene sequences in public databases has been increasing exponentially. However, knowledge about the genes grows at a slow rate [Kohane *et al.* 03][Ermolaeva *et al.* 98]. There is a high demand on effective manipulation and analysis of large data sets in a functional genomic research.

One of the main goals of current genomic research is identifying gene functions. Genes are specific regions in DNA sequences and they have necessary information to produce proteins that develop cells in organisms. Unraveling and modifying genes enhance our ability in scientific fields. For example, we may be able to cure a currently incurable serious disease by detecting the disease in gene sequences in early stage and rewriting a gene sequence to make malfunction proteins work properly rather than maintenance complex protein sequences [NOVA 01].

Microarrays are widely used experimental techniques to identify gene functions because they can compare tens of thousands of genes at a time. Microarrays generate gene expression-levels of many different genes to show similarity of the gene activities in different stages of the biological process. Similarities of the gene activities imply their biological functional similarities. We classify genes into the same functional groups and analyze sequence patterns to find statistically significant segments (motifs) that commonly appear in groups of genes and control gene activities. However, efficient computational and statistical techniques are necessary to analyze a large amount of data and to understand complex data patterns [Moreau *et al.* 02].

There are mainly two computational processes in microarray technologies: clustering and motif finding. Clustering classifies genes into certain biological functional groups by observing gene expression-levels. Using the result from the clustering, motif finding analyzes a group of the gene sequences and identifies motifs among the same group of genes. There are many different types of algorithms in the computational processes, because each microarray experimental result has a different nature of similarities and available prior information. Therefore, choosing the appropriate algorithms for analysis is a crucial element of the experimental design [Quackenbush 01]. This paper will introduce some representative algorithms that are employed in microarray technologies with their advantages, disadvantages, experimental results and examples.

In Chapter two, microarray technologies are briefly described. Its experimental methods and applications are introduced with some molecular biology basics.

In Chapter three, representative clustering algorithms such as hierarchical clustering [Eisen *et al.* 98], *K*-means clustering [Tavzoie *et al.* 99], SOM (self-organizing

map) [Tamayo *et al.* 99], SOTA (self-organizing tree algorithm) [Herrero *et al.* 01], model-based clustering [Yeung *et al.* 01], quality-based clustering [Heyer *et al.* 99] and adaptive quality-based clustering [De Smet *et al.* 02] are described. Also, this chapter presents general description of the EM algorithm that is widely used in both clustering and motif finding algorithms.

In Chapter four, motif finding algorithms that use a likelihood or information content measurements are discussed. A string-based method [Tompa 99], greedy algorithm [Hertz *et al.* 90], EM algorithm [Lawrence *et al.* 90], Gibbs sampling [Lawrence *et al.* 93] [Liu *et al.* 95] and Gibbs motif sampling [Neuwald *et al.* 95], are introduced as well as a concept of information content measurement.

2. MICROARRAYS

2.1 Molecular Biology

Molecular biology is a study of cells that compose all living-organisms in molecular level. Nucleic acids (that contains DNA and RNA (ribonucleic acid)) and proteins manage functions of the all organisms; nucleic acids encode and convey information to produce proteins and proteins are in charge of the physical activities. DNA is transcribed to RNA that is translated to proteins and proteins do all cell activities with enzymes.

2.2 DNA to Proteins

DNA is a blue print of the organism activities. DNA has double strands (chains) of DNA molecule and each DNA molecule is called nucleotide. The nucleotide consists of a sugar, a phosphate, and a base. To identify each DNA molecule, we call it with one of four bases: adenine (A), thymine (T), cytosine (C) and guanine (G) in computational molecular biology. Each base in a single strand pairs with its own complement; adenine always pairs with thymine and cytosine always pairs with guanine. Therefore, if there is a single DNA sequence, it will anneal to a complementary sequence and be able to form a double-strand DNA sequence.

A complete set of DNA sequences is called a genome. Genomes in any organisms are long; *E. Coli* (bacterium) has 600,000 bp (base pairs) and the human genome has 3 billion bp approximately. Genomes are found in chromosomes in any cells in human

except for mature red blood cells. According to [HGP 04], about 99.9% of human genomes are identical among all people and only 2% of genomes are encoding regions that are called genes.

Human has 30,000-40,000 genes with length of 3,000 bp in average. Genes are tagged with 3' and 5' at each end and there are upstream region and downstream region in a gene. Upstream region is where motifs are usually found and it is also called control region. Downstream region is composed by a sequence of codons that are groups of three nucleotides. Each codon will be transcribed into corresponding mRNA (messenger RNA).

mRNA is a single stranded and the structure is very similar to DNA. It is possible to obtain DNA from mRNA. DNA transcription splices out intron that are regions of genes that are not necessary in further biological process and the rest of genes are called exon. A reverse transcribed DNA is called cDNA (complementary DNA) and it only contains exon. As DNA is transcribed into mRNA, mRNA goes out from nucleus and

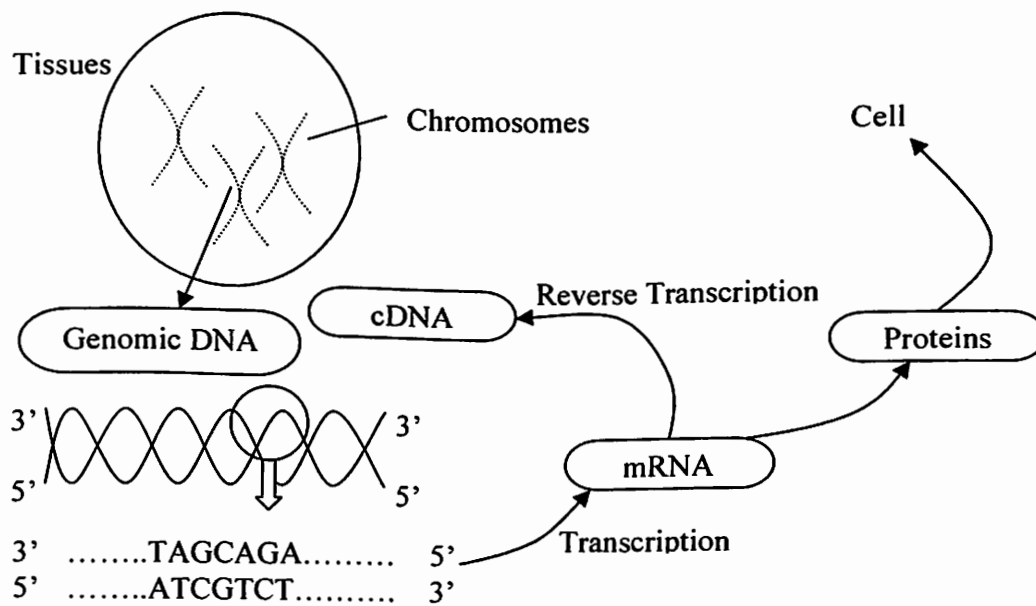


Figure 1. Biological Process from DNA to Proteins.

moves to certain proteins called ribosomes. mRNA downstream is translated into proteins in ribosomes. Figure 1 shows a summary of biological process from DNA to proteins.

2.3 Microarray Experimental Methods

One of advantages of microarray technologies is that microarrays are able to directly measure gene activities that are involved in a particular mechanism or system. Examples of such gene activities are transcription and translation; microarrays that use transcription and translation for its expression-level measurement are called RNA detection microarrays [Kohane *et al.* 03].

RNA detection microarrays work as following. A targeted tissues or cells are chosen and its mRNA is extracted from their gene sequences. cDNA is reversely transcribed by the mRNA and stored in each grid on a slide. These cDNA are called probes. Transcription can be initiated externally by heat shock or stress. By sampling produced mRNA in different time phase, we can measure how far the transcription has done in each different gene sequences. Also, we can measure how far the translation has done by checking how much RNA left in the produced protein in ribosomes.

Sample probes are fluorescently labeled and hybridized with probes on the slides. The sample probe only hybridizes with its complementary probe and rest of them will be washed away. The brightness of the probes are measured by a laser scanner, converted to quantitative numbers and recorded as an expression-level in a table that has genes in its row and samples in its column. Each row is referred as a gene profile.

cDNA microarrays and oligonucleotide arrays are two main technologies of microarrays. Following subsections describe the both technologies briefly with emphasis

on cDNA microarrays, because most of papers we refer in this thesis employed cDNA microarrays.

2.3.1 cDNA Microarrays

cDNA microarrays are also called robotically spotted microarrays [Kohane *et al.* 03]. DNA sequences are amplified by PCR (polymerase chain reaction) so that more information is available from the sequences. The gene profiles are derived from a certain DNA of interest and mechanically cDNA is spotted on a slide glass. Two different probes are used in each expression: test probes and reference probes (controlled probes). The experiment data takes ratio of these two expression-levels. In RNA detection microarray, reference probes will be complete mRNA sequences and test probes will be sampled DNA sequences from incomplete transcription stage or reference probes will be complete mRNA sequences and test probes will be sampled mRNA sequences from incomplete

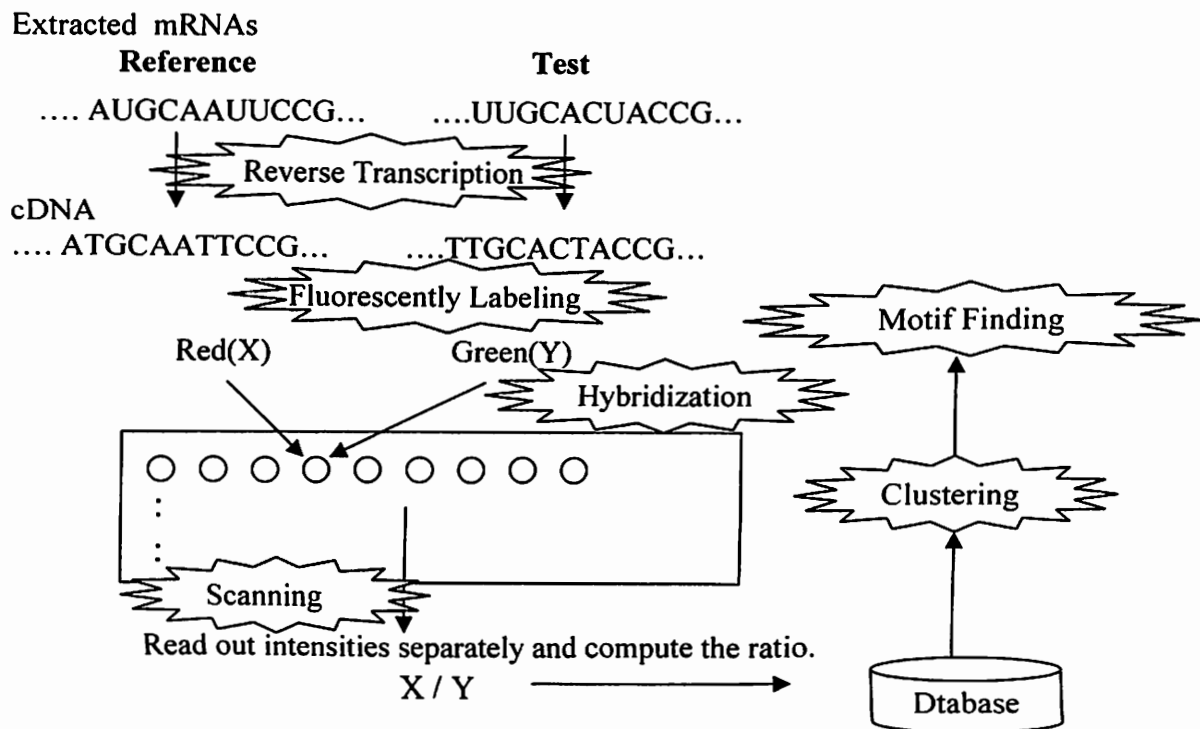


Figure 2. Overview of cDNA Microarray.

translation stage.

Advantage of this method is that we can customize target gene profiles for each slide. We can put a larger piece of cDNAs or entire cDNAs on the chip. Disadvantage is that we cannot obtain absolute quantities from the measurement since experiment result is shown as a ratio of test probes versus reference probe.

2.3.2 Oligonucleotide Arrays

Oligonucleotide arrays are mass-produced and distributed by manufactures [Kohane *et al.* 03]. Gene profiles are provided according to characteristics of experiments: yeast genes, mouse genes, etc. The most popular product is the GeneChip[®] (Affymetrix Inc., Santa Clara, CA) that allows us to compare more genes in a single experiment than cDNA microarray because of its high density. Basic experimental procedure is still the same, however, a pair of probes: mismatch and perfect match is read, computed and intensity is recorded in absolute value.

Advantages of this technology are an availability of absolute value measurement and its large capacity. However, this technology is more expensive than cDNA microarray. Especially if target gene profiles are on two different slides, we have to do the experiment twice, because current technology allows us to use one slide at a time in a single experiment.

3. CLUSTERING

3.1 Clustering Algorithms

Once experiments give us gene-expression profiles, we can group them with respect to their behaviors. Each gene profile is represented as a vector; i th gene profile with n observations is represented as $x_i = [x_{i,1}, x_{i,2}, \dots, x_{i,n}]$. The vector is called a gene-expression vector. Usually obtained expression-levels in each vector are normalized over different observations and comparison between the gene-expression vectors is made by Euclidean distance most commonly [Quackenbush 01].

All clustering algorithms that we discuss in this thesis are considered to be an unsupervised analysis, since they look for characterization of the components of the dataset without a priori input such as particular priori patterns [Kohane *et al.* 03]. They are technically classified into three groups: nearest-neighbor clustering, agglomerative clustering and divisive clustering.

Nearest-neighbor clustering decides the number of clusters and cluster centers initially, and gene-expression vectors are assigned into each cluster (K -means clustering, SOM and model-based clustering). Agglomerative clustering is a bottom-up method; a

cluster is initially empty and genes are added into a cluster (hierarchical clustering and quality-based clustering). Divisive clustering is a top-down method; initially a large cluster is created and it will be divided into small clusters (SOTA and adaptive quality-based clustering).

Many of the algorithms in this thesis involve a likelihood analysis and EM (Expectation-Maximization) algorithm [Dempster *et al.* 77], and they help algorithms attain global optima from a large set of gene-expression vectors.

3.2 General Description of EM algorithm

The EM algorithm is a general method of finding a maximum-likelihood estimate of parameters of an underlying distribution from a given data set when the data is incomplete or has missing values [Bilmes 98].

Let X be a data set size of n such as $X = \{x_1, x_2, \dots, x_n\}$, the probability of drawing X given by a set of parameters θ is $\prod_{i=1}^n \Pr(x_i|\theta)$. Most of the time, a logarithmic-likelihood is used for its computational simplicity and written as $\log(\Pr(X|\theta))$. The likelihood function is represented as $L(\theta|X)$. To maximize the likelihood L , we need to find θ^* such that,

$$\theta^* = \underset{\theta}{\operatorname{argmax}} L(\theta|X).$$

Next, we consider a hidden (or missing) data set Y with an observable data set X .

A complete-data will be $Z = (X, Y)$. Since there is no observable distribution in Y , Y is dependent on X and θ . Then $\Pr(z|\theta) = \Pr(x,y|\theta) = \Pr(y|x,\theta) \Pr(x|\theta)$. Its likelihood function will be $L(\theta |Z) = L(\theta|X,Y) = \log(\Pr(X,Y|\theta))$.

We define a two-place function $Q(\theta, \theta')$ that maximizes the likelihood of set Z with its parameter θ , where θ is an optimized variable set that maximizes the likelihood of set Z and θ' is a temporary variable set that is used to estimate a likelihood in the current random variable set Y . An EM algorithm obtains θ , using two steps: Expectation step and Maximization step. The algorithm alternates the two steps until the algorithm converges.

Expectation step: we estimate a likelihood $\log(\Pr(X,Y|\theta))$ from the current random variable Y and the current parameter $\theta^{(i-1)}$:

$$\begin{aligned} Q(\theta, \theta^{(i-1)}) &= E[\log(\Pr(X, Y | \theta)) | X, \theta^{(i-1)}] \\ &= \int_{y \in Y} \log(\Pr(X, y | \theta)) f(y|X, \theta^{(i-1)}) dy, \end{aligned}$$

where $f(y|X, \theta^{(i-1)})$ is a probability distribution function that governs the random variable Y . By taking average of such y with the current parameter $\theta^{(i-1)}$, we can estimate the complete-data likelihood, $\log(\Pr(X,Y | \theta))$.

Maximization step: X and $\theta^{(i-1)}$ are constant but Y is random variable, so adjust θ to

obtain the maximized expectation. Then we will compute $\Theta^{(i)}$,

$$\Theta^{(i)} = \underset{\Theta}{\operatorname{argmax}} Q(\Theta, \Theta^{(i-1)}).$$

The more accurate estimation we have, the more maximization we can make; once we capture correct gene-expression vectors (or motif candidates in motif finding) in the estimation step, right vectors (motif candidates) are obtained in the further maximization step. Examples of EM algorithms are shown in sections 3.3.1, 3.3.3, 3.5.2, 4.3.2, 4.3.3, and 4.3.4.

3.3 Nearest-Neighbor Clustering

3.3.1 *K*-means Clustering

K-means clustering [Tavzoie *et al.* 99] partitions an n -dimensional space into K clusters by EM algorithm, where K is the expected number of clusters and n is the number of observations of gene-expression. All gene-expression vectors are assigned to one of the clusters.

Advantages of this algorithm are that biologists can easily see the clusters in the n -dimensional space and simplicity of computation; however, this algorithm requires the number of clusters as a priori knowledge.

An experiment [Tavzoie *et al.* 99] shows that estimating the number of clusters

is difficult and suggests independent analyses: functional category enrichment and motif searching to validate clustering. In the experiment, 3,000 genes from an oligonucleotide array were categorized into 30 clusters and it took 200 iterations to converge by the statistical software package SYSTAT 7.0 (SPSS). Each cluster had 49 to 186 genes and successfully obtained some meaningful clusters, however, not all clusters showed significant enrichment for functions. They concluded that the number of clusters overestimated the diversity of biological expression classes in the dataset.

K-means Clustering Algorithm:

The algorithm employs the EM algorithm; where X is a set of gene-expression vectors, Y is a set of cluster centers, and a set of parameters is partitions of clusters.

Input: a set of gene-expression vectors $X = \{x_1, x_2, \dots, x_i, \dots, x_m\}$, where $1 \leq i \leq m$ and the number of expected clusters, K .

1. Randomly, partitions are created in an n -dimensional space to form K clusters, where n is the number of observations in a gene-expression. Map the set of vectors X into the space.
2. Its cluster centers $Y = \{y_1, y_2, \dots, y_k, \dots, y_K\}$, where $1 \leq k \leq K$, are calculated by taking an average of the vectors in each cluster. (Estimation Step)

$$y_k = [(\sum_{i=1}^u x_{i,1})/u, (\sum_{i=1}^u x_{i,2})/u, \dots, (\sum_{i=1}^u x_{i,n})/u],$$

where a cluster k has u vectors.

3. Reassign the set of vectors X into each cluster according to the new cluster center; a vector x_i will belong to a cluster that has the nearest cluster center, $y_{i,p}$ from the vector,

$$D(x_i, y_k) = \|x_i - y_k\| = [(x_{i,1} - y_{k,1})^2 + (x_{i,2} - y_{k,2})^2 + \dots + (x_{i,n} - y_{k,n})^2]^{1/2},$$

where n is the number of observations in a vector x . A function D calculates a distance between x_i and an arbitrary chosen cluster center y_k by Euclidean distance measurement,

$$y_{i,p} = \underset{y}{\operatorname{arg\,min}} (D(x_i, y_k)).$$

Vector x_i is assigned into the cluster that the nearest cluster center $y_{i,p}$ belongs.

After all the vectors are reassigned into the clusters, repartition the space into clusters, according to the vectors' locations that belong to the same cluster.

(Maximization Step)

4. Repeat the step 2 and 3 until the cluster centers Y become stationary.

Example of *K*-means Clustering Algorithm:

Figure 3 shows alternation of the expectation-maximization steps: continuous changes of cluster centers and partitions. Gray dots represent gene-expression vectors and black circles are cluster centers. The sequence of alternation demonstrates that some vectors move from cluster to cluster as the process goes on.

First, assign a set of vectors in an *n*-dimensional space and partition the space into four (Figure 3 (a)). The same numbers implies that they belong to the same cluster. Cluster centers are calculated (Figure 3 (b)). The partition is removed. Assign the set of vectors again into the space (Figure 3 (c)). Partition the space according to the vectors that belong to the same clusters. Bold numbers imply that the vectors belong to different

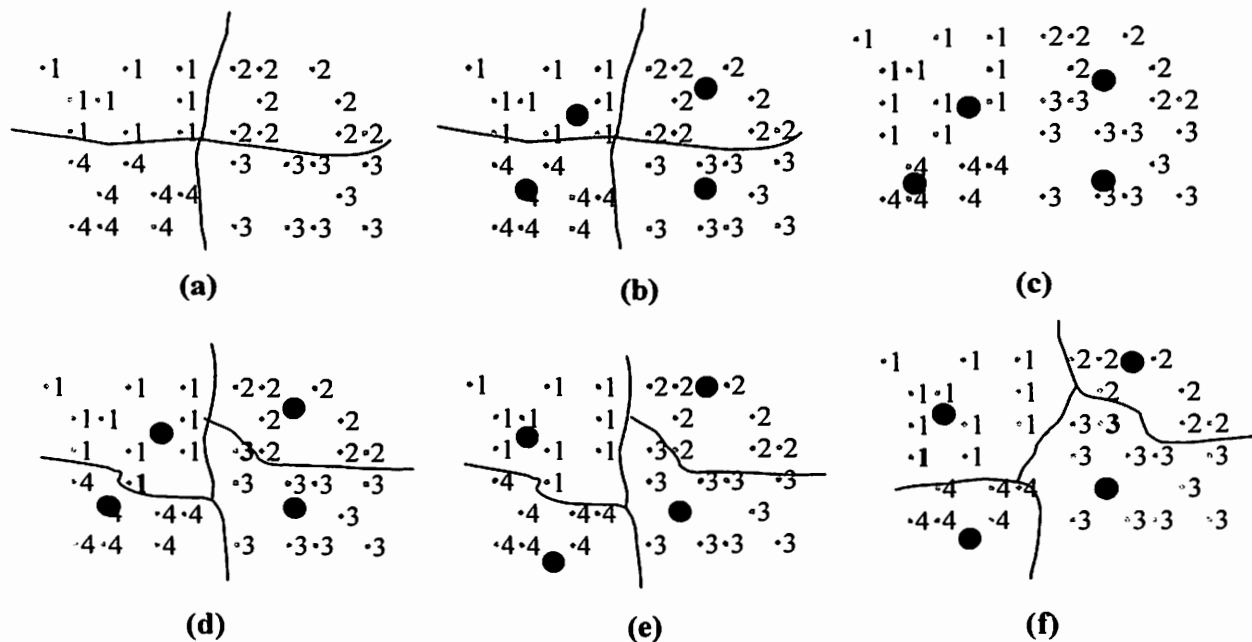


Figure 3. *K*-means Clustering.

clusters from the last time (Figure 3 (d)). Recalculate and locate the cluster centers again (Figure 3 (e)). The set of vectors are reassigned to the clusters and the space is repartitioned accordingly (Figure 3 (f)).

3.3.2 SOM (Self-Organizing Map)

SOM [Tamayo *et al.* 99] is an application of neural network; cluster centers are considered to be cells and vectors are used as inputs. In an n -dimensional space, SOM initially locates a certain simple dimensional shape (such as a two-dimensional rectangle) of grids into the space, where n is the number of observations of gene-expression vector. The grids represent cluster centers. All gene-expression vectors are mapped into the space. In each iteration, a vector is randomly selected and a distance between the selected vector and each cluster centers are calculated. The nearest cluster center is moved toward the selected vector. The neighboring cluster centers are also moved proportional to the distance between each cluster center and the vector. The process continues until it converges.

Advantage of the algorithm is that initial clusters are assigned with certain geometrical shape of grids and biologists can estimate the shape from their observation of plotted vectors in a space. Unlike K -means clustering, initial clusters are not assigned

randomly, therefore, the convergence is attained more efficiently. Disadvantage of this algorithm is that the number of clusters has to be known.

The experiments [Tamayo *et al.* 99] were done on yeast (828 genes into 30 clusters) and hematopoietic differentiation (567 genes into 12 clusters and 1,036 genes into 24 clusters). A similar yeast experiment had been presented earlier [Cho *et al.* 98] with hierarchical clustering and [Tamayo *et al.* 99] used the same data sets to see an accuracy of SOM. They stated that their result matches to hierarchical clustering result very close. The hematopoietic differentiation experiment showed one of clusters contained 32 genes with four duplicates and 18 genes were expected genes. A Web-based software package GENECLUSTER was used in the experiments and it is available at <http://www.broad.mit.edu/cancer/software/software.html>.

SOM Algorithm:

Input: a set of gene-expression vectors $X = \{x_1, x_2, \dots, x_i, \dots, x_m\}$, the number of expected cluster K and a geometrical shape of grids (cluster centers), $Y = \{y_1, y_2, \dots, y_k, \dots, y_K\}$, where $1 \leq i \leq m$ and $1 \leq k \leq K$.

1. Map the set of vectors X into an n -dimensional space, where n is the number of observations in gene-expressions. Then map the simple geometry shape of grids Y

as initial cluster centers.

2. Train a set of clusters Y for the certain numbers of iterations or until the cluster centers become stationary.

1. Randomly select a vector x_i and choose the nearest cluster $y_{i,p}$,

$$D(x_i, y_k) = \|x_i - y_k\| = [(x_{i,1} - y_{k,1})^2 + (x_{i,2} - y_{k,2})^2 + \dots + (x_{i,j} - y_{k,j})^2]^{1/2}.$$

A function D calculates distance between x_i and an arbitrary chosen cluster center y_k by Euclidean distance measurement,

$$y_{i,p} = \underset{y}{\operatorname{arg\,min}} (D(x_i, y_k)).$$

2. Move clusters according to a learning function,

$$F_s(y_k) = y_k + \tau(D(y_k, y_{i,p}), s)(D(x_i, y_k)).$$

Function $F_s(y_k)$ calculates position of cluster center y_k at s th iteration. τ is a learning rate that decreases with distance of cluster center y_k from $y_{i,p}$. $\tau(t,s) = 0.02T/(T+100s)$ for $t = \rho(s)$, where T is a maximum number of iteration, $\rho(s)$ decreases linearly with s and initially $t=3$.

Example of SOM:

Figure 4 shows a principle of SOM with initial geometry of clusters in rectangular grid. Each grid is numbered and arrows show their location after algorithm

converged. The gene-expression vectors are represented by black dots.

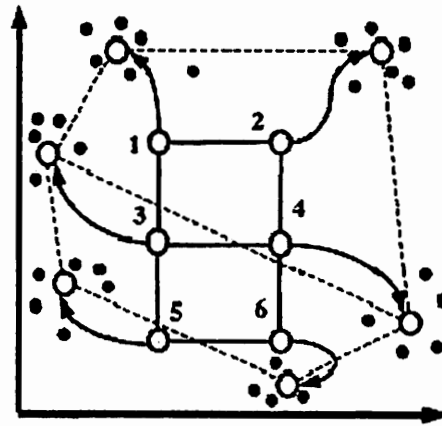


Figure 4. The Principle of SOMs [Tamayo *et al.* 99].

3.3.3 Model-Based Clustering

The model-based clustering [Yeung *et al.* 01] uses a statistical approach with assumption that there are finite number of multivariate normal distributions (and the number of distributions is the number of clusters) over gene-expression data. The clustering chooses a model (sets of parameters) and the correct number of clusters by EM algorithm (see Section 3.2). Models are a combination of parameters: volume, shape and orientation. Classical *K-means* clustering is an example of the equal volume spherical model that is the simplest model and does not have any three of parameters as variables.

The algorithm performs EM algorithm on every possible models of different number of clusters, compares the result by BIC (Bayesian Information Criterion)

[Schwarz 78] in different number of clusters and finds the best models with the number of clusters. The EM algorithm estimates parameters by calculating a conditional probability that each vector belongs to each cluster (Estimation Step), and adjusts parameters to maximize the likelihood that the vectors belong to each cluster (Maximization Step).

Advantage of this algorithm is its accuracy because a variety of combinations of variables are available (Table 1). The algorithm can choose models and the number of clusters that fit to each dataset. Also, the algorithm is totally data-driven and the number of clusters is not required as an input. However, the computational complexity is quadratic.

[Yeung *et al.* 01] reports successful experiment results in their paper. Their model-based clustering software MCLUST [Fraley 99] was compared to the leading heuristic-based clustering CAST (Cluster Affinity Search Technique) [Ben-Dor 99] in experiments of two different types of data sets: synthetic data and real microarray experimental data. Synthetic data was created as a mixture of normal distributions to verify the performance of the algorithm over different kinds of distributions.

In the experiment result of synthetic data, there were over-estimated number of clusters, however, they claimed that the clusters were strongly related each other to form

one cluster that was statistically correct. In real gene-expression experiment, MCLUST correctly chose the right number of clusters and model (384 genes into 5 clusters and 237 genes into 4 clusters; the model-based clustering usually reduces the number of genes by preprocessing a set of genes by hierarchical clustering, therefore, more genes were involved in the experiment than indicated above). In some data sets, MCLUST outperformed CAST.

Figure 5 shows an example of the experimental result of real gene-expression data. (Figure 5 (a)) shows that CAST predicted 6 clusters; however, (Figure 5 (b)) shows that MCLUST predicted 5 clusters correctly.

The MCLUST is available at <http://www.stat.washington.edu/fraley/mclust/>.

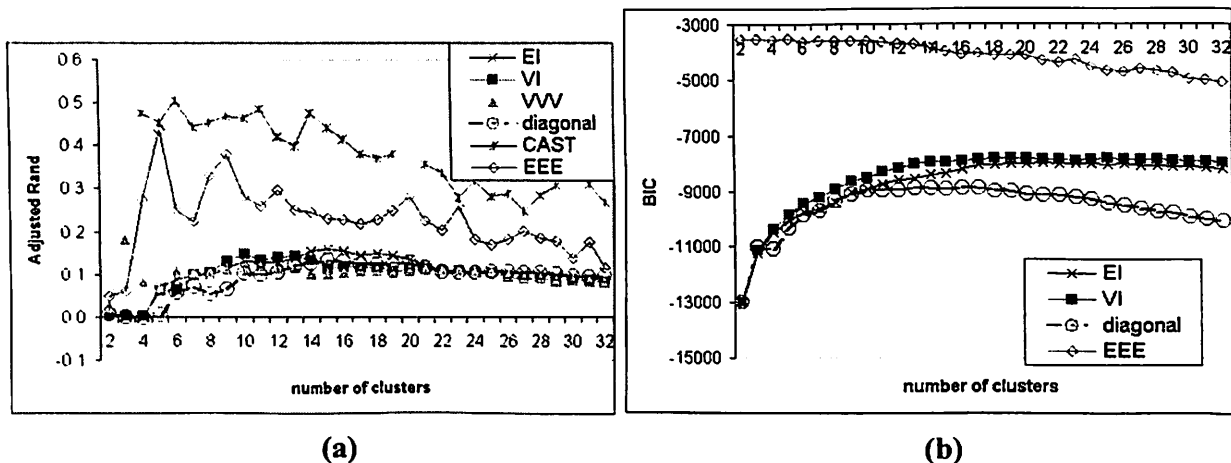


Figure 5. Example Result from the Experiment [Yeung *et al.* 01].

Modeling Datasets:

Our assumption is that some underlying probability distributions generate each component of gene-expression vectors X . A likelihood of mixture model is defined as:

$$L_{\text{mix}}(\theta_1, \dots, \theta_k | X) = \prod_{i=1}^m \sum_{k=1}^K \pi_k f_k(x_i | \theta_k),$$

where K is the number of clusters, k is an index of cluster, π_k is the probability of an gene-expression vector belongs to k th cluster and defined as $\sum_{k=1}^K \pi_k = 1$ and $\pi_k \geq 0$, f_k is a density function of vectors x in given parameter θ . The parameter θ is the Gaussian mixture model and it is decomposed into two: mean vectors (cluster center) μ_k and covariance matrix $\Sigma_k = \lambda_k D_k A_k D_k^T$ (parameters), where λ_k is a scalar and controls volumes, D_k is an orthogonal matrix of eigenvectors and controls orientation, and A_k is a diagonal matrix, its values are proportional to eigenvalues and it controls shape. Each model is represented by covariance matrix as shown in Table 1.

ID	Model	Volume	Shape	Orientation
EI	λI	equal	equal	N/A
VI	$\lambda_k I$	variable	equal	N/A
EEE	$\lambda D A D^T$	equal	equal	equal
VVV	$\lambda_k D_k A_k D_k^T$	variable	variable	variable
EFV	$\lambda D_k \hat{A} D_k^T$	equal	fixed	variable
EEV	$\lambda D_k A D_k^T$	equal	equal	variable
VFV	$\lambda D_k \hat{A} D_k^T$	variable	fixed	variable
VEV	$\lambda_k D_k A D_k^T$	variable	equal	variable

Table 1. Currently Available Parameters in MCLUST [Fraley 99].

For example, VI ($\lambda_k I$), where $I = D_k A_k D_k^T$, can take volume of each cluster into account and obtain tightly related group of vectors in one cluster. The function f_k is defined as

$$f_k(\mathbf{x}_i | \mu_k, \Sigma_k) = \exp\{-(1/2)(\mathbf{x}_i - \mu_k)^T \Sigma_k^{-1} (\mathbf{x}_i - \mu_k)\} / (\det(2\pi \Sigma_k))^{-1/2}.$$

Model-Based Clustering Algorithm:

Input: a set of gene-expression vectors $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_i, \dots, \mathbf{x}_m\}$, where $1 \leq i \leq m$.

[Ghosh 02] presents an actual implementation of the model-based algorithm.

Let X be a set of gene-expression vectors and Y be a set of variables $y_{i,k}$ that represents a membership of \mathbf{x}_i and $y_{i,k} = 1$ if \mathbf{x}_i belongs to cluster k or $y_{i,k} = 0$ otherwise. Let Z be the complete data set of (X, Y) . As we mentioned above, an EM algorithm is used to find the best model that fits to the dataset X , however, to enhance efficiency and accuracy of the EM algorithm, we usually use agglomerative hierarchical clustering (see Section 3.4.1) to create initial dataset. Unlike normal hierarchical clustering, the hierarchical clustering uses probability of genes that belongs to the same cluster instead of using Euclidian distance to compare a pair of vectors.

1. Hierarchical Clustering: merge clusters that increase logarithmic likelihood l^{CL} ,

$$l^{CL}(\theta_1, \dots, \theta_K, c_1, \dots, c_m | \mathbf{x}_1, \dots, \mathbf{x}_m) = \prod_{i=1}^m f_{l_i}(\mathbf{x}_i | \theta_{l_i}),$$

where c represents a membership of cluster, l_i indicates a known membership of

cluster i and there are K clusters all together.

2. The following EM algorithm is done on all potential models with the different estimated number of clusters.

Expectation Step: We estimate y_i that represents cluster assignment by

$$\hat{y}_{i,k} = \hat{\pi}_k f_k(x_i | \hat{\theta}_k) / (\sum_{p=1}^K \hat{\pi}_p f_p(x_i | \hat{\theta}_p)).$$

Maximization Step: We adjust π and θ to maximize a likelihood of the complete

data, that is $L_{cd} = \prod_{i=1}^m \prod_{k=1}^K (\pi_k f_k(x_i | \theta_k))^{y_{i,k}}$, or its logarithmic likelihood is

$$l_{cd} = \sum_{i=1}^m \sum_{k=1}^K y_{i,k} \log(\pi_k f_k(x_i | \theta_k)).$$

3. We select the best model with the suitable number of clusters by measuring integrated likelihood of the models.

Integrated likelihood over different models are taken as,

$$p(X|M_n) = \int p(X|\theta_n, M_n) p(\theta_n | M_n) d\theta_n,$$

where function p represents a probability that X is observed under model M_n , n is

a set of different models, θ_n is parameters in a set of models n . The estimate $\hat{\theta}_n$ is

the maximized likelihood estimate for parameter θ_n that is a clustered set of genes

we obtained from step 2. For example, if we have $n = \{1, 2\}$, $B_{12} = p(X | M_1) / p(X |$

$M_2)$ and if $B_{12} > 1$, M_1 is more favored than M_2 . BIC (Bayesian Information

Criterion) [Schwarz 78] is used for computational simplicity.

$$2\log p(X|M_n) \approx 2\log p(X|\hat{\theta}_n, M_n) - v_n \log m = \text{BIC}_n,$$

where v_n is the number of independent parameters obtained from covariance Σ_n , and m is the number of vectors. The first term symbolizes a maximized mixture likelihood model and the second term penalizes over-fitting from free variables.

3.4 Agglomerative Clustering

3.4.1 Hierarchical Clustering

Hierarchical clustering is one of the most widely used clustering algorithms, because the clustering process is graphically represented by a tree and it is easy to observe clusters and clustering process for biologists. There are bottom-up and top-down approaches in hierarchical clustering. In this thesis, we discuss bottom-up hierarchical clustering [Eisen *et al.* 98], since it is more popular than top-down approach.

Initially, gene-expression vectors are ordered by simple methods of weighting genes, such as an average expression level, maximum time taken to observe certain gene expression level, or chromosomal positions, and we place the element with the lower average weight earlier in the ordering. Then we calculate every pair of Euclidean distance and create a table and merge the closest pair of gene-expression vectors with the average-linkage method [Sokal 58].

This algorithm's advantage is that it is simple, easy to implement and easy to visualize the clustering process. However, it suffers from space complexity of data tables and distortion of clusters by averaging original clusters.

The paper [Eisen *et al.* 98] shows that the algorithm is more successful on properly ordered wide variety of observations than repeated observations. Experiments were done on a growth response in human cell (8,600 genes) and a budding yeast *S. cerevisiae* (6,200 genes). They also mentioned that a single noise in the observation did not affect in its neighboring genes. However, randomized ordered data sets were also tested and they could not obtain the same result. They concluded that gene-expression vectors' order is significantly important for accuracy.

Hierarchical Clustering Algorithm:

Input: a set of gene-expression vectors $X = \{x_1, \dots, x_i, \dots, x_j, \dots, x_n\}$ and $1 \leq i < j \leq n$.

1. Initially, every vector is considered to be a cluster. Create an ordered data table, which has biologically similar gene-expression vectors adjacently. Develop Euclidean distance table of every couple of clusters. The distance is obtained by function D that is defined by:

$$D(x_i, x_j) = \|x_i - x_j\| = [(x_{i,1} - x_{j,1})^2 + (x_{i,2} - x_{j,2})^2 + \dots + (x_{i,n} - x_{j,n})^2]^{1/2},$$

where each vector has n observations.

2. Comparing distance between every couple of clusters, the algorithm merges the shortest-distance pair by taking average of two vectors. Continue this process until there is only one cluster left. An average-linkage method of vectors x_i and x_j is defined as,

$$\text{ave}(x_i, x_j) = [(x_{i,1} + x_{j,1})/2, (x_{i,2} + x_{j,2})/2, \dots, (x_{i,n} + x_{j,n})/2].$$

3. Create a tree, which has initial clusters as its leaves and the last cluster as a root.

Final clusters are obtained by cutting the tree at the certain level.

Example of Hierarchical Algorithm:

Figure 6 shows an example of hierarchical clustering. Data table is created (Figure 6 (a)). Euclidean distance table is created (Figure 6 (b)). According to the Euclidean distance table, gene2 and gene3 is merged and the data table is updated (Figure 6 (c)). Figure 6 (d) shows the actual process of entire clustering process by tree.

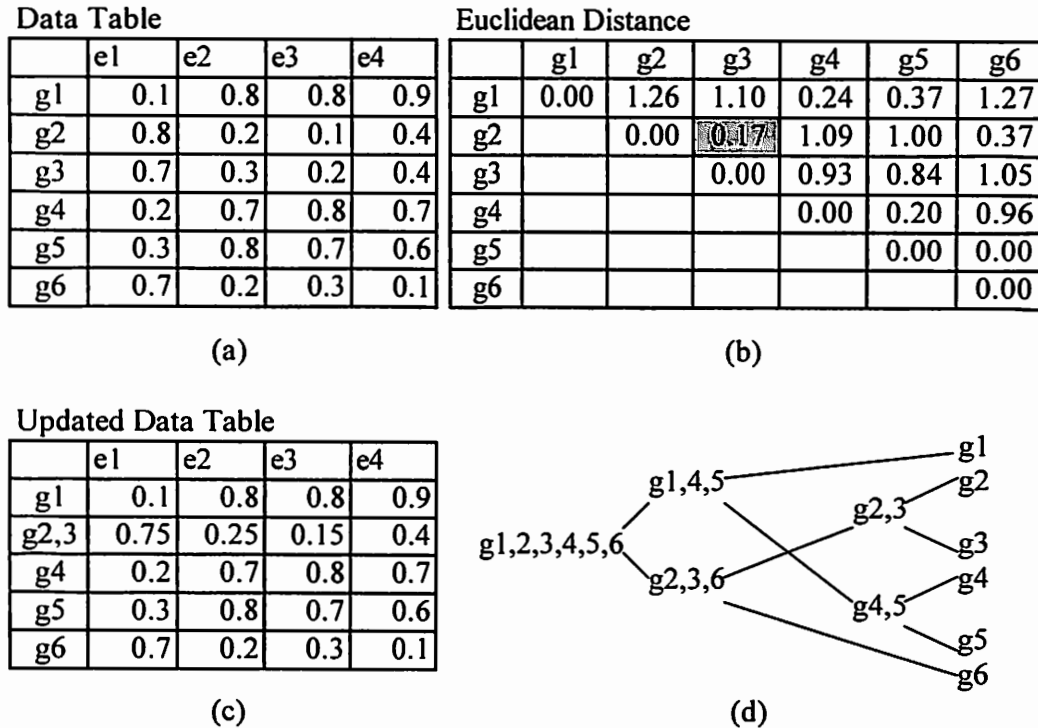


Figure 6. Example of Hierarchical Clustering.

3.4.2 Quality-Based Clustering

Quality-based clustering [Heyer *et al.* 99] takes into account of errors from the experiments and instead of using all expression-levels, uses expression-levels that satisfy qualities: similarity of gene-expressions and minimum population of cluster. When we use Euclidean distance measurement to compare gene-expression vectors similarity, sometimes one or two very similar expression-levels bias the clustering analysis and give us higher correlation of two vectors as a result; however, this will generate false positive genes that are not biologically correlated but they are correlated in the analysis.

The quality-based clustering standardizes expression levels and uses jackknife

correlation [Efron 82] to measure correlation of gene-expression vectors, instead of plain Euclidean distance measurement. A vector is selected as a cluster candidate and is located in a space. Then vectors that have the larger jackknife correlation with the cluster are added. Also, other vectors that minimize the increment of the cluster diameter are added and the process is repeated until the diameter reaches user-defined maximum diameter size. Once the process has been saturated, next cluster candidate is selected. The algorithm continues until doing the same procedures for all vectors. Finally, the largest cluster is selected and all vectors that belong to the cluster will be removed. The same iteration continues until algorithm cannot select a cluster that meets the minimum population that is defined by user.

The algorithm can eliminate bad gene-expression vectors, improve accuracy and capture clusters in small range. However, there are disadvantages; its computational complexity is high ($O(N^2)$), additional input is required (minimum number of vectors in a cluster and diameter of circle) and diameter is fixed locally. [Heyer *et al.* 99] consider only single observation for outlier. When we need to consider more than one observation such as size s , there are $(s!-1)$ of combinations and an effective computational technique is necessary to calculate jackknife correlation for every possible set.

The experiments are done on 4169 ORFs (open reading frames; downstream of

genes) from yeast cell cycle. Twenty-four clusters that maintain high quality are reported in [Heyer *et al.* 99]. They state that alternating jackknife correlation threshold does not make significant difference in the experiment but alternating a diameter of circle and the number of vectors in a cluster does.

They suggest two applications of quality-based clustering. Taking a median of representative pattern from the clusters, the algorithm can group gene-expression vectors that satisfy quality of the cluster. Also, we can generate only clusters of genes with interest. Instead of clustering on all gene-expression vectors, we can select a vector with special interest and group the vectors that satisfy quality of the cluster.

Quality-based Algorithm:

Input: a set of gene-expression vectors $X = \{x_1, x_2, \dots, x_i, \dots, x_m\}$, where $1 \leq i \leq m$, the minimum number of vectors in a cluster and the maximum cluster diameter d .

1. Standardize gene-expression levels by taking mean μ over observation in each profile, subtracting mean from the values and divide it by its standard deviation σ :

$$x_{ij} = (x_{ij} - \mu_i) / \sigma_i \text{ and } \sigma_i = (\sum_{j=1}^n (x_{ij} - \mu_i)^2 / (n-1))^{1/2},$$

where $0 < j \leq n$ and n is the number of observations in gene-expression vectors.

2. Calculate jackknife correlations $J_{p,q} = \min(\rho_{p,q}^{(1)}, \rho_{p,q}^{(2)}, \dots, \rho_{p,q}^{(n)}, \rho_{p,q})$, where

$0 < p < q \leq m$, $\rho_{p,q}$ is a correlation of the pair of vectors \mathbf{x}_p and \mathbf{x}_q . $\rho_{p,q}^{(t)}$ denotes that a correlation without expression-level sample from column t . We use Euclidean distance D for our correlation measurement:

$$D(\mathbf{x}_p, \mathbf{x}_q) = \|\mathbf{x}_p - \mathbf{x}_q\| = [(x_{p,1} - x_{q,1})^2 + (x_{p,2} - x_{q,2})^2 + \dots + (x_{p,n} - x_{q,n})^2]^{1/2}.$$

3. Select a vector \mathbf{x}_i (initially \mathbf{x}_1) to make a cluster center of candidate cluster C_i , where $0 < i \leq m$ and group of clusters $C = \{C_1, \dots, C_i, \dots, C_m\}$. Select the nearest vector \mathbf{x}_p that its jackknife correlation $J_{i,p}$ meets threshold d . Add it into the cluster C_i . $\mathbf{x}_p \in C_i$ and $J_{i,p} = \min(\rho_{i,p}^{(1)}, \rho_{i,p}^{(2)}, \dots, \rho_{i,p}^{(n)}, \rho_{i,p}) > d$.

Take average of \mathbf{x}_i and \mathbf{x}_p , and update cluster center.

4. Within user-defined diameter d , find neighboring vectors \mathbf{x}_q that minimizes the diameter increase of cluster C_i and \mathbf{x}_q is added into the cluster C_i :

$$\mathbf{x}_q = \arg \min_{\mathbf{x}} (D(\mathbf{c}_i, \mathbf{x}_q)),$$

$$D(\mathbf{c}_i, \mathbf{x}_q) = \|\mathbf{c}_i - \mathbf{x}_q\| = [(c_{i,1} - x_{q,1})^2 + (c_{i,2} - x_{q,2})^2 + \dots + (c_{i,n} - x_{q,n})^2]^{1/2} < d$$

5. Repeat step 4 until there are no more vectors to be added. Create a candidate cluster with next vector and repeat the step 3 and 4. Continue the process until last vector \mathbf{x}_m .
6. Select the largest cluster C_{\max} and remove all vectors that belong to cluster C_{\max} from X , then move to step 2. Continue the whole process until there is no more

clusters that meet the minimum size that user specified.

Example of jackknife correlation:

Assuming that we obtained a result from microarray experiment and standardized the data as below, we demonstrate the algorithm from Step 2 to Step 4 as follows.

x_1	-0.2105	-0.36441	-0.92235	-0.92663
x_2	0.29574	0.93209	0.56843	0.23017
x_3	0.03307	-0.32469	-0.40135	0.79631
x_4	-0.80204	-0.22714	0.63205	0.3681
x_5	-0.38479	-0.03287	0.70618	-0.826
x_6	-0.35284	-0.66014	0.06172	-0.65343

Let $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$ and $d = 0.5$ be input data. Starting from x_1 , we consider C_1 .

We take a jackknife correlation to all other vectors versus x_1 and obtain the smallest correlation for each vector is underlined as follows,

(removed term) (0)	(1)	(2)	(3)	(4)	
x_2	2.34473	2.28943	1.95368	<u>1.80980</u>	2.03951
x_3	1.81682	1.80042	1.81639	1.74052	<u>0.57649</u>
x_4	2.11216	2.02764	2.10770	<u>1.43006</u>	1.66880
x_5	1.67407	1.66497	1.64091	<u>0.38784</u>	1.67104
x_6	1.07272	1.06324	1.03116	<u>0.42703</u>	1.03735

$$\begin{aligned} \text{i.e. } \rho_{1,2}^{(1)} &= \left(\cancel{(x_{1,1}-x_{2,1})^2} + (x_{1,2}-x_{2,2})^2 + (x_{1,3}-x_{2,3})^2 + (x_{1,4}-x_{2,4})^2 \right)^{1/2} \\ &= \left(\|-0.36441-0.93209\|^2 + \|-0.92235-0.56843\|^2 + \|-0.92663-0.23017\|^2 \right)^{1/2} \\ &= 2.289437376 \end{aligned}$$

$$J_{1,2} = \min(\rho_{1,2}^{(0)}, \rho_{1,2}^{(1)}, \rho_{1,2}^{(2)}, \rho_{1,2}^{(3)}, \rho_{1,2}^{(4)}) = \rho_{1,2}^{(3)} = 1.80980.$$

$$\text{Then, } J_{1,3} = \rho_{1,3}^{(4)} = 0.57649, J_{1,4} = \rho_{1,4}^{(3)} = 1.43006, J_{1,5} = \rho_{1,5}^{(3)} = 0.38784, J_{1,6} = \rho_{1,6}^{(3)} =$$

0.42703; and $x_p = x_5$.

Now, cluster center will be the average of x_1 and x_5 , that is c_1 $[-0.29764, -0.19864, -0.10808, -0.87631]$. Then add another gene vectors x_q into cluster C_1 , if the distance from c_1 to the vector x_q is the smallest and it is less than d .

(removed term) (0)	(1)	(2)	(3)	(4)	
x_2	1.82006	1.72061	<u>1.42621</u>	1.68966	1.44510
x_3	1.73462	1.70281	1.73004	1.70965	<u>0.45963</u>
x_4	1.53349	1.44816	1.53322	1.34305	<u>0.89611</u>
x_6	0.54271	0.53990	<u>0.28558</u>	0.51546	0.49483

$J_{c_1,2}=\rho_{c_1,2}^{(2)}= 1.42621$, $J_{c_1,3}=\rho_{c_1,3}^{(4)}= 0.45963$, $J_{c_1,4}=\rho_{c_1,4}^{(4)}= 0.89611$, $J_{c_1,6}=\rho_{c_1,6}^{(2)}=$

0.28558, and x_6 is added into C_1 . Cluster center is recalculated and repeat the same process.

3.5 Divisive Clustering

3.5.1 SOTA (Self-Organizing Tree Algorithm)

SOTA [Herrero *et al.* 01] is a hybrid algorithm of SOM and hierarchical algorithm. Like a hierarchical clustering, clusters are represented by a tree; however, the cluster updates are done by SOM algorithm. By hybridizing two algorithms, SOTA is able to overcome disadvantages of both algorithms and produce high quality clusters.

Initially, a root has an average vector of all gene-expression vectors. Its child

nodes are created and the parent node vector is copied into them. The child nodes are called terminals and considered to be clusters. In each iteration, distance between a randomly selected gene-expression vector and all terminals are compared. The closest terminal is selected and updated, and its sibling and its parent are updated as well. Upon updates, an error rate is calculated and the terminal with the largest error creates its child nodes and the tree grows. The iteration lasts until clusters do not satisfy the threshold any longer. The threshold can be the maximum or average distance between cluster centers and vectors.

An advantage of this algorithm is overcoming drawbacks of hierarchical algorithm and SOM. By adopting an update method from SOM, the algorithm avoids losing identity of genes during the tree growth. A tree structure enables us to obtain clusters without knowing the number of clusters and to observe a hierarchical relationship of clusters. The algorithm compares purely heterogeneity of vectors rather than the number of similar vectors. A disadvantage of this algorithm is that defining the threshold to terminate the algorithm is time consuming.

[Herrero *et al.* 01] reports that SOTA can produce high quality clusters; cluster centers obtained by SOTA were very close to an average values of each sample from gene-expression vectors in the clusters (less than 0.3% discrepancy). An experiment was

done on a yeast cellular cycle data with 800 genes and obtained 40 clusters (cluster set A) and 174 clusters (cluster set B). The cluster set A had threshold of distance 0.75 and the cluster set B had threshold of confidence level 5%. The cluster set B was at about four times higher resolution than the cluster set A, because clusters in A were spliced into approximately four clusters.

Self-Organizing Tree Algorithm:

Input: a set of gene-expression vectors $X = \{x_1, x_2, \dots, x_i, \dots, x_m\}$, where $1 \leq i \leq m$,

threshold θ , where θ could be a minimum distance to update terminals or confidence level.

1. Create a root of tree by taking an average of all n observations of all gene-expression vectors in X ,

$$x_{\text{avg}} = ((\sum_{s=1}^m x_{1,s})/m, (\sum_{s=1}^m x_{2,s})/m, \dots, (\sum_{s=1}^m x_{n,s})/m).$$

2. Create child nodes and copy x_{avg} into the nodes as initial cluster centers. The child nodes are called terminals.
3. Randomly select a gene-expression vector from X and compare distance between the vector and terminals. Select the nearest terminal that is called a winning cell.
Update the winning cell, its sibling and parent nodes,

$$\mathbf{x}_i(\tau+1) = \mathbf{x}_i(\tau) + \eta \cdot (\mathbf{x}_i - \mathbf{x}_i(\tau))$$

where on t th node vector at τ th iteration, selected vector \mathbf{x}_i chooses $\mathbf{x}_i(\tau)$ as either winning cell or sibling node or parent node, then updates as $\mathbf{x}_i(\tau+1)$. η represents magnitude of update and decreases winning cell (η_w), ancestor node (η_a) and sibling node (η_s) respectively. Typically, the values are $\eta_w=0.01$, $\eta_a=0.005$ and $\eta_s=0.001$ [Dopazo 97].

4. The step 3 lasts until every vector in X is selected and this cycle is counted as one epoch. In each iteration, heterogeneity of each cell is calculated by:

$$R_k = (\sum_{i=1}^u D(\mathbf{x}_i, \mathbf{y}_k)) / u.$$

where $D(\mathbf{x}_i, \mathbf{y}_k) = \|\mathbf{x}_i - \mathbf{y}_k\| = [(x_{i,1} - y_{k,1})^2 + (x_{i,2} - y_{k,2})^2 + \dots + (x_{i,j} - y_{k,j})^2]^{1/2}$, u is the number of vectors in cluster k . R_k is called a resource value and the total error ε at v epoch is obtained by $\varepsilon_v = \sum_{k=1}^K R_k$. Relative increase of the error is measured and when it is below a given threshold θ , the iteration stops:

$$|(\varepsilon_v - \varepsilon_{v-1}) / \varepsilon_{v-1}| < \theta$$

When it does not converges, the highest resource value R_k of the cell \mathbf{x}_k is going to split into two child nodes. The child nodes have parent node vector as a default.

Then iteration is resumed from step 3.

The threshold using resource value R_k measures a quality of cluster. Another

threshold measures confidence level of clusters. A variability V is used instead of resource value R . V is obtained by $V = \max_k (d_k)$, where $d_k = \max_{i,l} (D(x_i, x_l))$, where $1 \leq i < l \leq m$, and x_i and x_l belong to the same cluster.

Example of SOTA:

Assuming that the following subset of standardized microarray data are obtained from an experiment,

x_1	-0.2105	-0.36441	-0.92235	-0.92663
x_2	2.29574	1.93209	1.56843	0.23017
x_3	0.03307	-0.32469	-0.40135	1.79631
x_4	-0.80204	-0.22714	2.63205	0.3681
x_5	-1.38479	-0.03287	0.70618	-0.826
x_6	-0.35284	-0.66014	0.06172	-0.65343

we take average of each column and we have the root = $[-0.06986, 0.05368, 0.606407, -0.003681]$. The root is copied into initial child nodes and a tree is constructed (Figure 7 (a)). A gene is randomly picked and updates either one of terminals (child nodes). The terminal 1 is updated by x_2 , based on formula in algorithm explained above. Its sibling (terminal 2) is also updated (Figure 7 (b)). (The root will not be updated, however, usually parent node is updated in further generation.) Figure 7 (c) shows updates of tree after one epoch. We measure error to determine either to grow tree or to terminate the

iteration. Since this is the only the first epoch, we continue the iteration and decide which terminal will grow first. $R_1 = (D(x_2, y_1) + D(x_1, y_1) + D(x_5, y_1))/3 = (3.17853 + 1.82371 + 1.53517)/3 = 2.17914$ and $R_2 = (D(x_4, y_1) + D(x_6, y_1) + D(x_3, y_1))/3 = (2.20383 + 1.12484 + 2.11060)/3 = 1.81309$. $R_1 > R_2$. So R_1 will grow as Figure 7 (d). Error rate for the first epoch is $E_1 = R_1 + R_2 = 3.99223$.

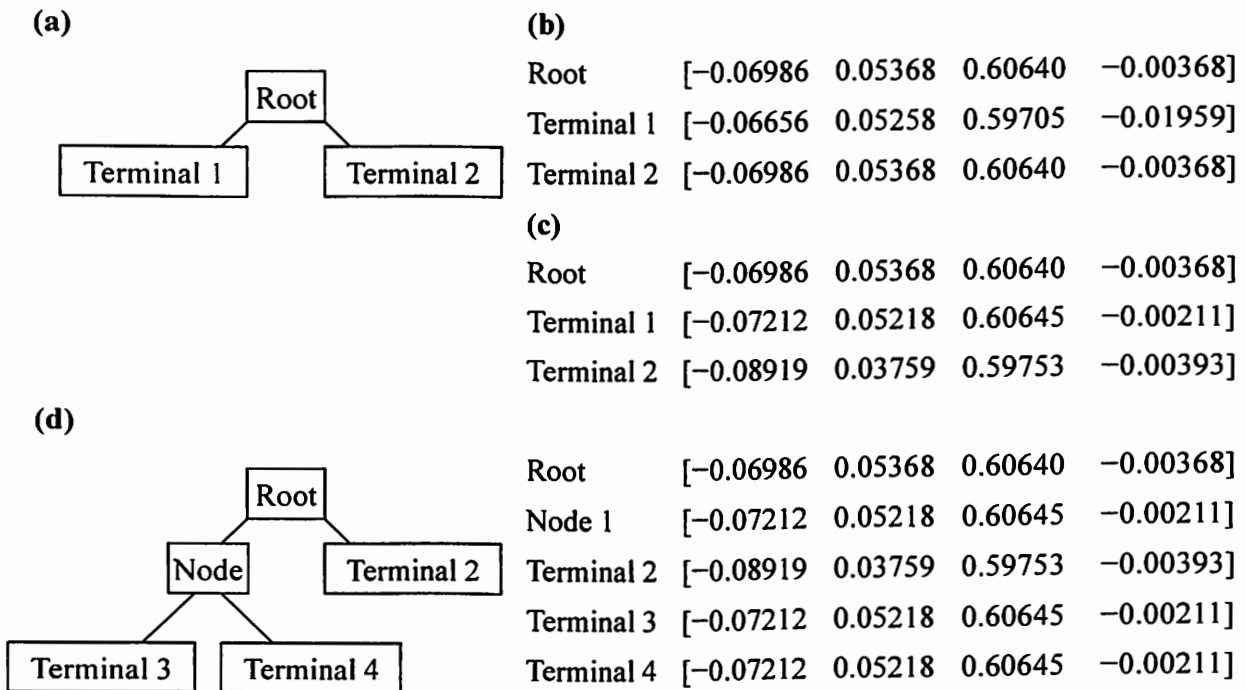


Figure 7. Example of SOTA.

3.5.2 Adaptive Quality-Base Clustering

Adaptive quality-based clustering [De Smet *et al.* 02] can adjust (adapt) the radius of cluster to obtain quality-guaranteed clusters for the each different cluster. This

algorithm does not use jackknife correlation, however, uses EM algorithm and likelihood ratio of gene-expression vectors in a cluster and successfully generate clusters with highly correlated vectors. Clusters are represented as a sphere in a multi-dimensional space; we assume that vectors in a cluster exist on the intersection of hyperplane and hypersphere. The algorithm has two steps: finding a location of cluster in high-dimensional space and deciding the radius of the sphere.

A sphere is located in a space that contains all vectors initially, then the center of sphere is considered to be a cluster center. The sphere center that is an average of the vectors in the cluster is calculated and the sphere is relocated according to obtained cluster center in each iteration. The radius of cluster decreases in each iteration until the sphere capture minimum number of vectors. Once the cluster captures the minimum number of vectors, a likelihood ratio of vectors in the sphere is calculated. Finally, the cluster only takes the vectors that have certain likelihood ratio (usually 95%), and the sphere and the vectors are removed from space. Next iteration starts with remaining vectors and iteration lasts until there are no more clusters to take.

Advantage of this algorithm is that highly quality-guaranteed clusters are obtained in linear running time that increases with the number of gene-expression vectors. A flexible radius size can capture clusters without extra computation on data processing

such as jackknife correlation. On the other hand, a convergence of algorithm has not been proved theoretically, therefore, there may be situations that never converge. Also, the algorithm can only apply for Euclidean distance measurement, since a likelihood analysis involves distance to cluster center.

Experiment result [De Smet *et al.* 02] shows that adaptive quality-based clustering can successfully generate biologically significant clusters and it outperforms the K -means clustering; the algorithm could detect some functional genes that the K -means clustering can not find and the algorithm produces more high density clusters than the K -means clustering. The software that is implemented by MATLAB and detail experiment results are available online at <http://www.esat.kuleuven.ac.be/~fdesmet/paper/adaptpaper.html>.

Adaptive Quality-Based Clustering Model:

Expression vectors are normalized as their mean is zero and variance is one. By definition, we have mean $\mu_i = 1/n (\sum_{s=1}^n x_{i,s}) = 0$, where n is the number of observations in gene-expression vector and the number of gene-expression vectors is m and $i \leq m$. Also, a standard deviation $\sigma_i = (1/(n-1) \sum_{s=1}^n (x_{i,s} - \mu_i)^2)^{1/2} = 1$. Solving for $\sum_{s=1}^n x_{i,s}$, we have $(n-1)^{1/2}$. [De Smet *et al.* 02] assumes that vectors in a cluster exist on the intersection of hyperplane μ_i and hypersphere with radius of $(n-1)^{1/2}$ in an n -dimensional

space and model the probability distribution as following.

The probability distribution of vectors with given cluster radius $p(r_c) = P_C \cdot p(r_c|C) + P_B \cdot p(r_c|B)$, where P_C is a prior cluster probability distribution and $p(r_c|C)$ is a current cluster probability distribution. P_B and $p(r_c|B)$ denote a prior and a current background probability distributions and $P_C + P_B = 1$:

$$p(r_c|C) = E_{n-2} / ((2\pi\sigma^2)^{(n-2)/2}) r_c^{n-3} \cdot \exp(-r_c^2 / (2\sigma^2)) \quad \text{--- (1)}$$

$$p(r_c|B) = E_{n-2} / (E_{n-1} (n-1)^{(n-2)/2}) r_c^{n-3} \quad \text{--- (2)}$$

where E_{n-2} is a surface area of a unit sphere in n -dimensions. A likelihood of vectors in a cluster with a given radius r_c is:

$$P(C| r_c) = (P_C \cdot p(r_c|C)) / (P_C \cdot p(r_c|C) + P_B \cdot p(r_c|B)). \quad \text{--- (3)}$$

We employ EM algorithm (see Section 3.2) to maximize $P(C| r_c)$ by adjusting r_c , where X is a set of gene-expression vectors, Y is a radius of cluster r_c and Θ is a prior standard deviation σ and probability distribution P_C (or P_B).

Adaptive Quality-Based Clustering Algorithm:

Input: a set of gene-expression vectors $X = \{x_1, x_2, \dots, x_i, \dots, x_m\}$, where $1 \leq i \leq m$, the minimum number of vectors in a cluster, the significance level S (95% by default).

1. [Locate Cluster] Map a set of gene-expression vector X and set an initial cluster C as a sphere in a n -dimensional space, where n is the number of observations in a vector and $C = \{x_1, x_2, \dots, x_i, \dots, x_m\}$ initially. Before iteration starts, initialize an estimated minimum radius $r_{\text{est}} = (n-1)^{1/2}/2$.

2. Relocate cluster by making the cluster center y to be an average of all vectors in the cluster, $y = x_{\text{avg}} = ((\sum_{s=1}^p x_{1,s})/p, (\sum_{s=1}^p x_{2,s})/p, \dots, (\sum_{s=1}^p x_{n,s})/p)$, where $|C| = p$.

If this is the first iteration for cluster C , calculate a delta for radius r_{Δ} .

$$r_{\Delta} = (D(x_z, y) - r_{\text{est}}) / r_{\text{fraction}}, \text{ where } r_{\text{fraction}} = 30 \text{ by default,}$$

$$D(x_i, y) = \|x_i - y\| = [(x_{i,1} - y)^2 + (x_{i,2} - y)^2 + \dots + (x_{i,n} - y)^2]^{1/2}, x_i \in C \text{ and}$$

$$x_z = \arg \max_x (D(x_c, y)).$$

Radius r_c is adjusted by $r_c = r_c - r_{\Delta}$.

3. Calculate a new cluster center and compare with the last cluster center. If it is the same, the algorithm converges, otherwise return to step 2.

4. Repeat step 2 and 3 until it reaches a maximum iteration times (50 times by default) or the radius is reduced into r_{est} . If the cluster center has not been fixed or the algorithm does not converge, then the algorithm will be terminated.

5. [Adjust Radius r] Maximize the likelihood of gene-expression vectors in a cluster by EM algorithm. Initially, $r_c = r_{\text{est}}$ and calculate σ and P_C by measuring distance

between a current cluster center and all vectors:

$$P_C = (\sum_{s=1}^p D(\mathbf{x}_s, \mathbf{y})) / (\sum_{t=1}^m D(\mathbf{x}_t, \mathbf{y})) \text{ and } P_B = 1 - P_C,$$

where $\mathbf{x}_s \in C$, p is the number of genes in a cluster and m is the number of all vectors in a space. $\sigma = ((\sum_{t=1}^m \mu - D(\mathbf{x}_t, \mathbf{y}))^2 / (m-1))^{1/2}$ and $\mu = (\sum_{t=1}^m D(\mathbf{x}_t, \mathbf{y})) / m$.

Using equations listed above, we can obtain $p(r_c|C)$ and $p(r_c|B)$ by plug-in σ , P_C and P_B into formula (1) and (2). (Estimation Step) Adjust r_c in formula (3) to maximizes likelihood of gene-expression vectors with given cluster $P(C| r_c)$.

(Maximization Step) Alternate two steps until it converges.

6. The iteration also stops when either relocating of a cluster or adjusting radius does not converge, or the cluster obtained does not meet the minimum number of vectors.

Example of Adaptive Quality-Based Clustering:

We demonstrate the relocation of cluster using algorithm Step 1 to Step 4.

Assuming that the following is a subset of standardized microarray data,

	$x_{m,1}$	$x_{m,2}$	$x_{m,3}$	$D(\mathbf{y}, \mathbf{x})$
\mathbf{x}_1	-0.2105	-0.36441	-0.92235	1.59212
\mathbf{x}_2	2.29574	1.93209	1.56843	3.17005
\mathbf{x}_3	0.03307	-0.32469	-0.40135	1.08241
\mathbf{x}_4	-0.80204	-0.22714	2.63205	2.17106
\mathbf{x}_5	-1.38479	-0.03287	0.70618	1.32111
\mathbf{x}_6	-0.35284	-0.66014	0.06172	0.94202

we estimate the minimum radius as $r_{est} = (n-1)^{1/2}/2 = (3-1)^{1/2}/2 = 0.7071$ and the center of cluster is set to be $y = x_{avg} = [-0.07023, 0.05381, 0.60745]$. The furthest gene from cluster center $x_2 = x_2$. Then $r_{\Delta} = (D(x_2, y) - r_{est}) / r_{fraction} = (3.17005) - 0.7071) / 30 = 0.0821$. In the next step, we reduce radius by r_{Δ} and $r_c = 2.38085$. x_2 is discarded because $D(x_2, y_{(1)}) = 3.17005$. New center $y_{(2)} = [-0.45285, -0.26821, 0.34604]$ is recalculated without x_2 . Since $D(x_4, y_{(2)}) = 2.31289$ and $r_c = 2.29875$, x_4 is discarded. Then $y_{(3)} = [-0.31918, -0.23035, -0.09263]$. At iteration 4, no more genes are discarded from the cluster and radius is fixed, therefore, the algorithm converges. Figure 8 shows how cluster was relocated in the process.

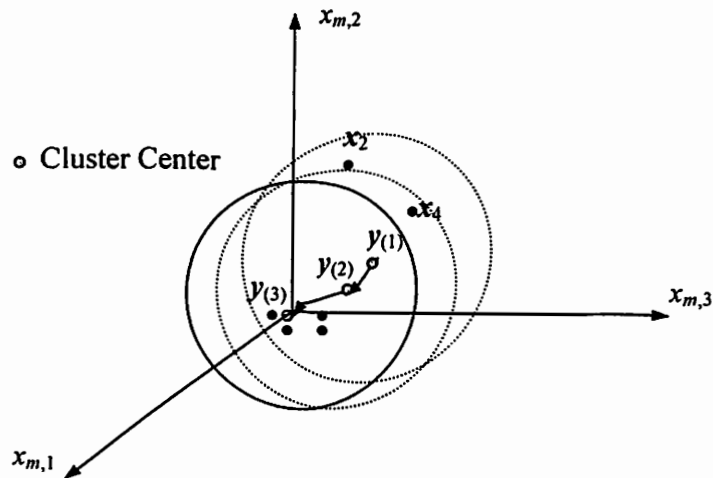


Figure 8. Relocation of Cluster in Adaptive Quality-Based Algorithm.

4. MOTIF FINDING

After clustering and its validation, we look for common subsequences in upstream of the gene sequences within the same clusters. The common subsequences are called motifs (or binding sites) that control the expression behaviors and have the same characteristics in functional genes. Each sequence may have zero or more motifs. Sometimes a certain pair or multiple motifs tend to occur together; therefore, there is a possibility that multiple motifs are detected simultaneously [Morequ *et al.* 02].

Detecting statistically significant motifs is a goal for motif finding. Finding such motifs from multiple sequences is shown to be NP-complete [Tompa 00][Akutsu 98][Akutsu *et al.* 00]. There are many algorithms to measure statistical significance of the motifs, however, this thesis focuses on methods that measure and compare a likelihood and information content of motifs.

4.1 Information Content

When we analyze motifs, it is important to consider not only the motif patterns but also the background frequencies. The background is a region in the sequences that the motifs exclude. If the most frequently occurring sequences are randomly selected from the set of sequences, they are heavily biased by base frequencies of the set. As a result, we will fail to detect significantly common motif patterns among the different sequences.

We want the motif pattern frequencies to be different from the background frequencies. The logarithmic-likelihood is used to measure how unlikely the motifs occur

in the multiple sequences. Instead of using the motif probability distribution, we use information content, that is a product of the motif probability distribution and the motif likelihood, to adjust the motif probability with its background frequencies. The more the motif contents are different from the background frequencies, the larger the information content will be.

Figure 9 shows an example of motif representation of a set of sequence S and $\{s_1, s_2, s_3, s_4\} \in S$. First, we count the number of each different base with respect to the position and create an alignment matrix. The number of base r in position i is represented as $n_{i,r}$ (Figure 9 (a)). Next, we compute the probability distribution $P(s_i, r)$, that is also called

		Position index	1	2	3
s_1ATCGTTCG <u>ATC</u> TTTGG.....		A	T	C
s_2GACC <u>CTG</u> TATATATTT.....		C	T	G
s_3GGCCGCGCAA <u>TTC</u> AA.....		T	T	C
s_4TATCAA <u>ATG</u> TCAAGT.....		A	T	G

A	$\begin{bmatrix} 2 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0.5 & 0 & 0 \end{bmatrix}$
T	$\begin{bmatrix} 1 & 4 & 0 \end{bmatrix}$	$\begin{bmatrix} 0.25 & 1 & 0 \end{bmatrix}$
C	$\begin{bmatrix} 1 & 0 & 2 \end{bmatrix}$	$\begin{bmatrix} 0.25 & 0 & 0.5 \end{bmatrix}$
G	$\begin{bmatrix} 0 & 0 & 2 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0.5 \end{bmatrix}$

(a)

(b)

$$\begin{bmatrix} 1 & - & - \\ 0 & 2 & 0 \\ 0 & - & 1 \\ - & - & 1 \end{bmatrix}$$

(c)

$$\begin{bmatrix} 0.5 & - & - \\ 0 & 2 & 0 \\ 0 & - & 0.5 \\ - & - & 0.5 \end{bmatrix}$$

(d)

(a) Alignment Matrix, (b) Profile, (c) Position Weight Matrix, (d) Relative Entropies of Motifs

Figure 9. Example of Motif Candidates of Width 3.

profile. $P(s_{i,r}) = n_{i,r} / k$, where k is the number of sequences in S (Figure 9 (b)).

We will have a position weight matrix of motif (Figure 9 (c)) by obtaining logarithmic-likelihood ratio on each element that is $\log_2 P(s_{i,r})/Q(s_r)$. The background model is represented by a weight matrix of position zero. In Figure 9, background model (Q) is assumed to be an equal distribution; the same number of bases exists in the background. So background model is represented as $Q(S) = [0.25, 0.25, 0.25, 0.25]^T$, where S represents a set of gene sequences.

Finally, information content (Figure 9 (d)) is a product of the probability distribution and the logarithmic likelihood ratio. We obtain the information content of motif I_{seq} by weighting likelihood ratio with probability of motif and summing result of all bases and the position within the motif as

$$I_{seq} = \sum_{i=1}^L \sum_{r \in \{A,T,C,G\}} P(s_{i,r}) \log_2 P(s_{i,r})/Q(s_r), \text{ where } L \text{ is length of motif.}$$

The motif finding is classified into two methods: a string-based method and a probabilistic method.

4.2 String-Based Methods

String-based method counts, compares a base frequencies and finds motif patterns. Tompa proposed a string-based method [Tompa 99], which counts all bases, selects the most frequently occurring strings and estimates their statistical significance with its background frequencies by z-score, $z = (N_t - Np_t) / (Np_t (1 - p_t))^{1/2}$, where t is the index of all possible candidate motifs, N_t is the number of sequences that have a motif candidate t with at most one substitution of base and p_t is the probability that a randomly selected sequence have at least one motif t . Then Np_t is the number of sequences that

have at least one occurrence of a motif candidate t . The z -score measure how unlikely it is to have N_t occurrences of a motif candidate t .

The string-based method demonstrates two advantages. It takes into account of the absolute number of occurrences. Under the uniform distribution, a perfectly conserved motif that occurs in only a few sequences will have a greater information content than an imperfectly conserved motif that occurs in nearly all the sequences. Since the algorithm uses exhaustive approach, it will not suffer from local optima like other heuristic methods do.

On the other hand, this method only works well on finding short and simple motifs. Because of the exhaustive approach, it is too expensive to apply for long sequences for complicated motifs.

This method is applied to the ribosome binding site problem and [Tompa 99] concluded this method successfully enumerates short motifs with their exact z -scores. The experiment is done on 14 prokaryotic genomes to find motifs of width 7. The result shows lists of 20 high z -score sequences from each sequence. The paper does not validate the result, however, suggests further analysis of the high z -score sequences and constructs a weight matrix from the 20 sequences to produce a single motif as an example of such analysis.

String-Based Algorithm:

Input: set of sequences $S = \{s_1, s_2, \dots, s_k\}$ and width of motif L

1. Count and compare strings in the set of sequences and count each string's number of occurrences N_t in the set by Staden's algorithm [Staden 89]; create a dictionary

that has all possible motifs of length L and count the number of each motif occurred in the set of sequence.

2. Calculate p_t , that is a probability that single random sequence contains at least one occurrence of candidate motif t of length L .
 1. Construct a deterministic finite automaton (DFA) M that accepts a sequence with a subsequence that matches with candidate motif t with at most one base substitution.
 2. Construct a Markov chain G that generates a randomly sampled sequence $X = \{x_1, x_2, \dots, x_j, \dots, x_n\}$ with degree of one. G satisfies $\Pr(x_j) = \Pr(x_j | x_{j-1}) = a_{j-1,j}$ and contains all possible transition probabilities of sequence X such that, $a_{1,2}, a_{2,3}, \dots, a_{j-1,j}, a_{j,j+1}, \dots, a_{n-1,n}$.
 3. Transform M into M' by mapping transition probabilities from Markov chain G on the edges of DFA M' and calculate the random sequence X 's probability p_t by tracing a start state to an accepting state.
3. Calculate the z -score of each candidate motif t and select the significantly large z -score strings as motifs.

Example of Computing p_t :

Consider a subsequence $t = \{ATC\}$ and a randomly sampled sequence X from a set of sequence S with $X = \{\underline{AA}ACCCG\underline{TTC}GAC\}$. Then X has three subsequences of t with one substitution as the underlines indicate, therefore, X satisfies the condition in step

2.

Create a DFA M as shown in the Figure 10. States q_u and q_v contain the prefix and suffix of matched subsequence in X . Then generate a first order Markov chain and compute the probability transitions $a_{j,j+1}$, where j is an index in the sequence X and there are $|X|-1$ pairs in X , $a_{j,j+1}$ = number of occurrence of pattern $(X_j, X_{j+1}) / (|X|-1)$.

Finally, using the probability transitions as a weight, we transform M into M' as shown in the figure. The bold arrows in M' indicate $\{CGTTCG\}$ of subsequence in X .

Finally, we have $p_t = \Pr(A)a_{AA}a_{AA}a_{AC}a_{CC}a_{CC}a_{CG}a_{GT}a_{TT}a_{TC}a_{CG}a_{GA}a_{AC}$

$$= (5/13)(2/12)(2/12)(2/12)(2/12)(2/12)(2/12)(1/12)(1/12)(1/12)(2/12)(1/12)(2/12)$$

$$= 1.10431E-11.$$

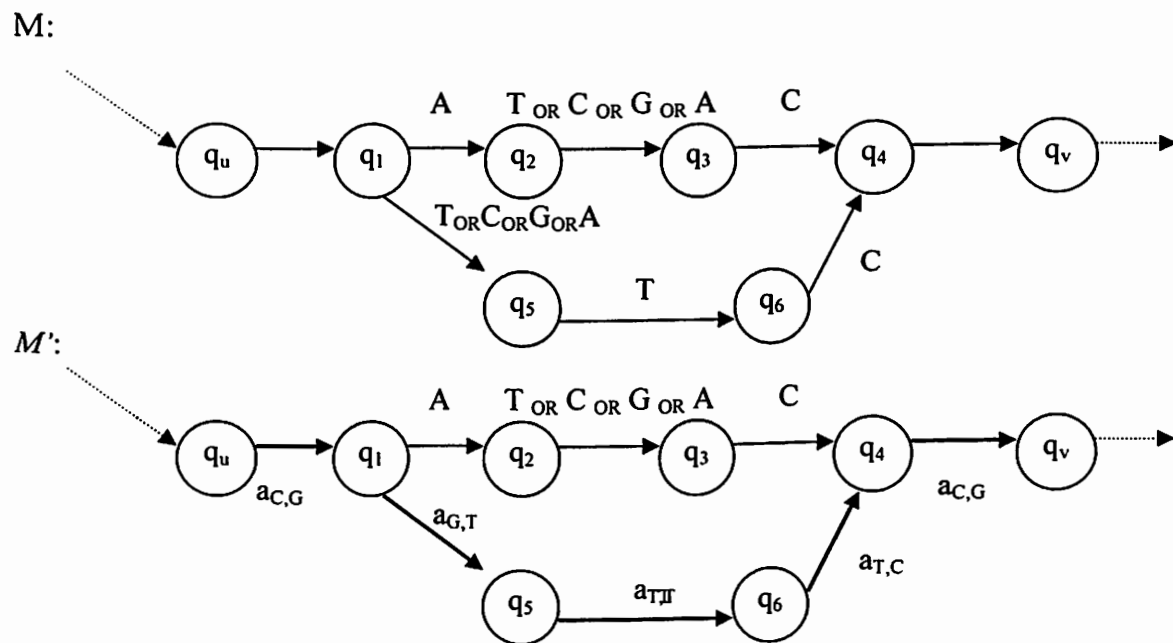


Figure 10. Transformation from M to M' .

4.3 Probabilistic Methods

4.3.1 Greedy Algorithm

In probabilistic methods, instead of comparing strings, the algorithms measure and compare the information content of candidate motifs with a greedy approach. Hertz [Hertz 99] implemented an algorithm to find the highest information contents with a pure greedy algorithm that tests gene sequences one by one and selects the alignment with the highest information content from each sequence. This algorithm assumes that exactly one motif exists in every sequence and the motif length is already known (mononucleotide model).

An advantage of this algorithm is its efficiency on finding motifs of high information content over string-based methods. A disadvantage is its lack of flexibility on finding motifs; a prior knowledge such as motif width is required and the algorithm cannot apply for any other models such as gapped motifs.

The greedy algorithm generated very successful result in [Hertz 99]. The experiment is validated and it has 19 out of 24 expected motifs with width of 22. The three out of five missing motifs are found to be overlapped with one of expected motifs.

Greedy Algorithm:

Input: set of sequences $S = \{s_1, \dots, s_m, \dots, s_k\}$, where $1 < m \leq k$ and width of motif L .

1. Calculate information content of all possible motif candidates with length of L in the set of sequences S including a set of motif S' (initially S' is empty):

$$I_{seq}^{mj} = \sum_{i=1}^L \sum_{r \in \{A,T,C,G\}} P(s_{m,j,i,r}) \log_2 P(s_{m,j,i,r}) / Q(s_r),$$

where j is a starting index of possible motif candidates in sequence s_m .

2. Select a motif that has the highest information content,

$$s_{m,j} = \arg \max_{m,j} (I_{\text{seq}}^{m,j}).$$

Remove the sequence s_m from the set S and add $s_{m,j}$ to S' as following,

$$S^{(t)} = S^{(t-1)} - \{s_m\} \text{ and } S'^{(t)} = S'^{(t-1)} + \{s_{m,j}\}.$$

3. Continue this step 1 and 2 until the set S becomes empty (k times).

Example of Greedy Algorithm:

Figure 11 shows the example of Greedy algorithm with three different sequences. Matrices in the figure are alignment matrices. In each cycle the largest information content motif is selected and added into the alignment matrix.

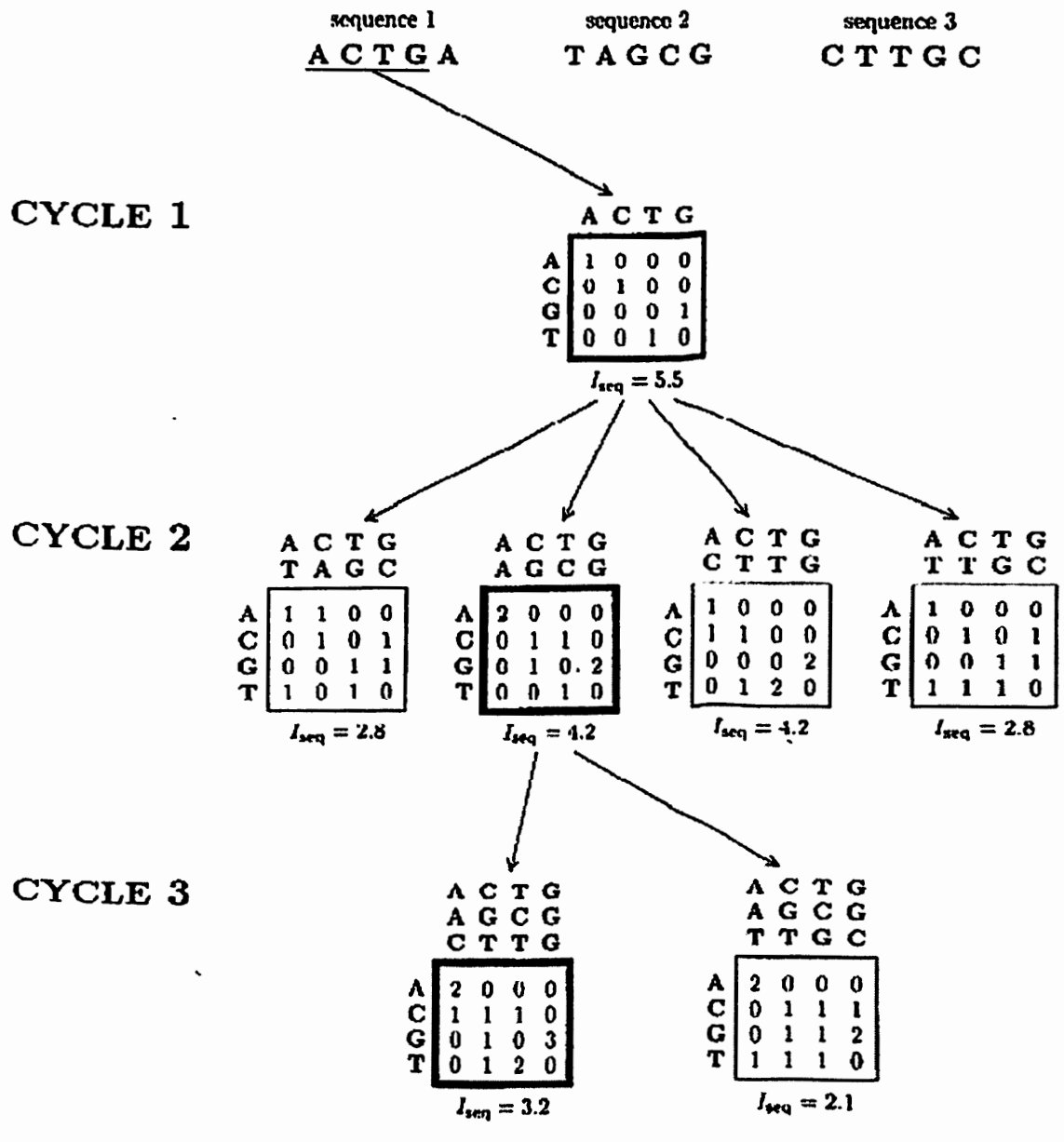


Figure 11. Example of Greedy Algorithm [Hertz 99].

4.3.2 EM Algorithm

EM algorithm [Lawrence *et al.* 90] optimized a greedy algorithm into two steps: estimating base frequencies of motif in each position with current set of motif candidates

and maximizing likelihood of set of motif by changing the starting position of each motif candidate so that its base population satisfies the estimated base frequencies. By considering a set of motif candidates in a group of sequences at once, EM algorithm can detect correlated motifs.

An advantage of the algorithm is its enhanced capabilities and computational simplicity that is linear in the number of sequences. EM algorithm (see Section 3.2) is applied to motif finding; where X is a set of sequences, Y is a starting position of motif and Θ is base frequency. This model enhances the motif finding capability by adding the second variable; adding variable-length gaps in the missing variable Y , we can obtain gapped motifs. On the other hand, this algorithm requires a prior knowledge such as a proposed motif set by statistical or biological analysis initially, because initial set of motif is very important to obtain accurate result.

An experiment [Lawrence *et al.* 90] shows that EM algorithm can detect motifs successfully. The experiment is done on 18 sequences with length of 105 and a motif was expected to have a width of 22. The result identified motifs correctly in 16 out of 18 sequences.

EM Algorithm:

Input: set of sequences $S = \{s_1, \dots, s_m, \dots, s_k\}$, where $1 < m \leq k$ and width of motif L .

1. Initially, set a motif starting position j for each sequence s_m in S , according to a proposed motif set.
2. Model the set of sequences with information measurement I_{seq} ,

$$I_{seq} = k \sum_{i=j}^L \sum_{r \in \{A,T,C,G\}} f(s_{i,r}) \log_e P(s_{i,r}) +$$

$$k(M-L) \sum_{r \in \{A,T,C,G\}} f(s_{0,r}) \log_e P(s_{0,r}),$$

and $f(s_{j,r}) = P(s_{j,r}) = n_{j,r} / k$, $f(s_{0,r}) = P(s_{0,r}) = n_{0,r} / k(M-L)$, where $j = 0$ denotes background, M is a length of sequence, $n_{j,r}$ is the number of bases r at position j , the function f is observed base frequency and the function P is our parameter. Calculate population frequencies of each position of motif candidates by summing probability that is obtained by Bayes formula shown below:

$$P(y_{m_j}=1 | P(s_{r,i})^{(q)}, s_m) = P(s_m | y_{m_j}=1, P(s_{r,i})^{(q)}) / (\sum_{j=1}^{M-L} P(s_m | y_{m_j}=1, P(s_{r,i})^{(q)})),$$

where $y_{m_j}=1$ if a motif starts at j , $y_{m_j}=0$ otherwise and

$$P(s_m | y_{m_j}=1, P(s_{r,i})^{(q)}) = \prod_{j=1}^L \prod_{r \in \{A,T,C,G\}} P(s_{r,i})^{v_{r,j+t,m}},$$

where $j \leq t \leq j+L$, therefore, t is relative index within the motif candidate.

The probability is used as weight and added across the positions to find the expected number of bases for each position. $\epsilon_{j,r}^{(q)} = E(n_{j,r} | P(s_{j,r})^{(q-1)}, S)$.

(Estimation Step)

3. Now, adjust the number of bases in a set of motif that we obtained from the last step by changing starting position j so that we can maximize I_{seq} . (Maximization Step) By changing the starting positions of motif, $P(s_{j,r})$ is updated. If $P(s_{j,r})^{(q+1)}$ is different from previous $P(s_{j,r})^{(q)}$, go back to Estimation Step.
4. Repeat step 2 and 3 until it converges.

Example of EM Algorithm:

Figure 12 (a) shows a model of motif alignment and background in EM algorithm with 10 sequences that have 40 bases in length. We look for a motif with width of 8. In

estimation step, count the number of each base and create alignment matrix. In maximization step, we sum the probability of each position as Figure 12 (b).

The formulas are given in step 3. As an example, obtain probability of having A at first sequence of first index at iteration q . The first motif starts from index 12, so we have,

$$P(s_1 | y_{1,12}=1, P(s_{12,A})^{(q)}) = \prod_{j=1}^L \prod_{r \in \{A,T,C,G\}} P(s_{j,r})^{v_{r,j+1,m}}$$

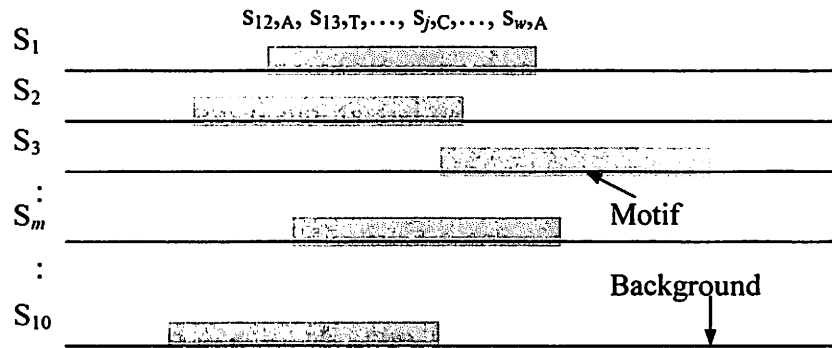
$$=(\# \text{ of A})^{(12)}/10(\# \text{ of T})^{(13)}/10 \dots (\# \text{ of C})^{(l)}/10 \dots (\# \text{ of A})^{(8)}/10, \text{ then}$$

$$P(y_{1,12}=1 | P(s_{12,A})^{(q)}, s_1) = P(s_1 | y_{1,12}=1, P(s_{12,A})^{(q)}) / (\sum_{j=1}^{40-8} P(s_1 | y_{1,12}=1, P(s_{r,i})^{(q)}).$$

By normalizing probability in Figure 12 (b), we will get expected number of base A at motif position 1.

$$\text{i.e. } n_{1,A} = k(A_j / (A_j + T_j + C_j + G_j)) = 10(A_1 / (A_1 + T_1 + C_1 + G_1)) = 10(.13 / (.13 + .33 + .12 + .59)) = 1.$$

Therefore, adjust starting position of motifs so that there is only one A in the first column of motif set.



(a)

	1	2	...	j	...	8
A	.13	.45	...			
T	.33	.33	...			
C	.12	.18	...			
G	.59	.22	...			

(b)

Figure 12. Example of EM Algorithm.

4.3.3 Gibbs Sampling

Gibbs sampling [Lawrence *et al.* 93] [Liu *et al.* 95] is similar to EM algorithm, however, Gibbs sampling only considers one element at a time, rather than summing all possibilities of each positions in a motif candidate like the EM algorithm does. This process reduced the time complexity. The algorithm improved performance of the EM algorithm by drawing a motif from all possible candidate motifs with weighted likelihood of generating motifs under current motif sets instead of taking a motif that maximizes the likelihood. As a result, Gibbs sampling is less likely to be trapped in local optima than is the EM algorithm. Also, the algorithm does not require a prior knowledge of the motif set as a default but only requires a set of sequences. The algorithm can also detect the width of motifs by running algorithm with different width on the same input. The algorithm can also apply to detect gapped motifs by taking account of relative positions with motif starting position within a sequence in likelihood analysis.

The experiment shows that Gibbs sampling is able to detect a set of motifs correctly and to handle gapped motifs and widely spread weakly conserved motifs. For instance, a set of 30 proteins is tested with different width and the algorithm detects motifs with width of 21 as the highest information content. The detected motifs are very close to the known motifs with width of 20.

Gibbs Sampling Algorithm:

Input: set of sequences $S = \{s_1, \dots, s_m, \dots, s_k\}$, where $1 < m \leq k$.

1. Initially, set a motif starting position j for each sequence s_m in S randomly.

2. Select a sequence s_z and exclude it from the set S . Obtain pattern description $P(s_{i,r})$ (also known as profiles in 4.1) and background $P(s_r)$ as,

$$P(s_{i,r}) = \sum_{i=j}^L (n_{i,r} + b_i) / (k-1+B) \text{ and } P(s_r) = \sum_{j=1}^M (n_{r,j} + b_j) / ((k-1)(M-L)+B),$$

where $n_{r,j}$ is the number of base r at position i , M is a length of sequence, L is a width of motif, i is index that is excluded from motif set, b_j is a pseudocounts (to avoid condition $\log_2 0 = \infty$) at position j and B is sum of b_j .

(Estimation Step)

3. Let x be a motif that is generated by s_z by all possible j . Draw a position j based on probabilities of generating motif x under the current motif set, $P_x = \Pr(x | s_m, \Theta_L)$ and probabilities of generating motif x by the background probability of Θ_0 , $Q_x = \Pr(x | s_m, \Theta_0)$. A weight A_x is calculated by $A_x = P_x / Q_x$ and normalize the probability distribution by $A'_x = A_x / \sum_{i=1}^{M-L+1} A_{x,i}$. We use information content, $I_{\text{seq}_x} = \sum_{i=j}^L \sum_{r \in \{A,T,C,G\}} n_{i,r} \log_2 P(s_{i,r}) / P(s_r)$, to obtain A_x . (Maximization Step)
4. Repeat step 2 and 3 until it converges.

Example of Gibbs Sampling:

In this example, we have 10 sequences with length 20 and look for motif with width of 5 (Figure 13 (a)). In estimation step, a sequence z is selected and we obtain the number of base at each position in a motif set and the background model (sequence z is excluded) as Figure 13 (b). In maximization step, we calculate all possible information content of motif set including a motif from z . As an example, we obtain information content that starts at index 3 in sequence z . Let z be sequence {GCTGTGAACCGT....}.

Then we have $P(s_{1,T}) = (4+1) \log_2(4+1/10) / .25 = 5$, $P(s_{2,G}) = (3+1) \log_2(3+1/10) / .25 = 2.75$,

$$P(s_{3,T})=(1+1)\log_2(1+1/10)/.25=-.64, P(s_{4,G})=(3+1)\log_2(3+1/10)/.25=2.75,$$

$$P(s_{3,A})=(0+1)\log_2(0+1/10)/.25=-1.32$$

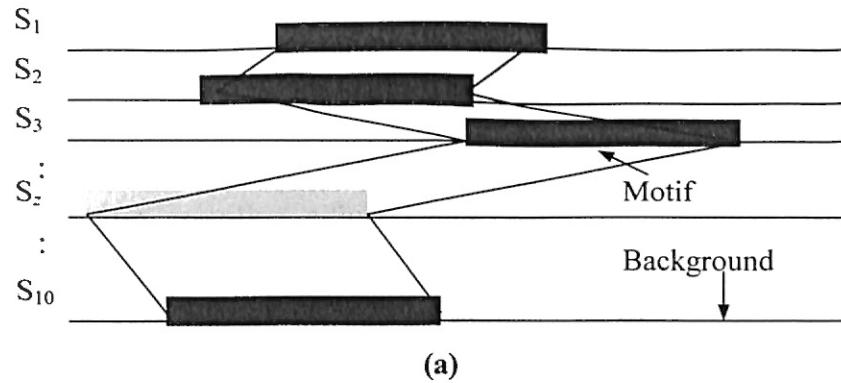
and information content for motif that includes z with motif starts at index 3 is:

$$IC_3 = \sum_{j=1}^5 P(s_j, r) = 8.54.$$

We normalize IC_3 over other starting index IC_j and obtain probability of having motif that starts at 3 in sequence s_z :

$$\Pr(IC_3) = IC_3 / \sum_{j=1}^{20} IC_j.$$

A sampled motif in s_z may not always maximize the over all information content, however, a motif with higher information content is more likely selected.



$r \setminus j$	1	2	3	4	5	Background Model
A	1	3	2	2	0	0.25
T	4	0	1	2	2	0.25
C	2	3	1	2	3	0.25
G	2	3	5	3	4	0.25

(b)

Figure 13. Example of Gibbs Sampling.

4.3.4 Gibbs Motif Sampling

Gibbs motif sampling [Neuwald *et al.* 95] can detect multiple motifs simultaneously by partitioning motif-encoding regions into different motifs with a prior knowledge of number of occurrences of each motif. There is a high demand on detecting widely spread weakly conserved motifs, because it often reveals important structural or functional roles of motif models. The paper [Neuwald *et al.* 95] mainly discusses protein sequences, however, the algorithm can be applied to DNA sequences as well. In this section, we define different group of motifs to be motif models since we discuss multiple groups of motifs in the algorithm. Also, we define motif candidates as sites.

The algorithm uses two samplings: motif sampling and column sampling. Alternatively applying these two sampling methods, we can obtain convergence more effectively. Motif sampling partitions sequences into motif models and background regions and column sampling adjust width of the site in motif sampling. After the algorithm converges, we will calculate significance of each motif in the motif models and rank them for future analysis.

Experiments are done on some protein sequences and show the effectiveness of algorithm. The algorithm detects highly significant motifs by converging three motif models with width of 12; 66, 35 and 63 motifs are obtained from 258 sequences. Also, the algorithm obtained 130 repetitive motifs with width of 11 varying in one to nine residues (same as bases in DNA sequences) by running on 32 bacterial iomps that past algorithm (BLAST [Altschul *et al.* 90]) could not detect any similarity among them.

Gibbs Motif Sampling Algorithm:

Input: set of sequences $S = \{s_1, \dots, s_m, \dots, s_k\}$, where $1 < m \leq k$, the number of motif models, the width of motifs L_i and $0 < j \leq L_i$, and the expected number of motifs in each motif model e_i .

1. Initially, set e_i of sites for each model M_i in the S and M_0 denotes background model.
2. [Motif Sampling] Target probabilities (also known as profiles in 4.1) $q_{j,r}$ for each model M_i and the background q_r are calculated based on current motif model alignments. $q_{j,r} = (n_{j,r} + b_r) / (c + B)$, where $n_{j,r}$ is a count of base r at position j , b_r is a pseudocounts of base r (to avoid condition $\log_2 0 = \infty$), c is the number of sites in the alignment and B is the total number of base pseudocounts.

A site x is selected randomly from S (or in a succeeding process, x is succeeding site in S), if the site is within one of motifs in the motif alignment, remove the site from the alignment and recalculate the target probability.

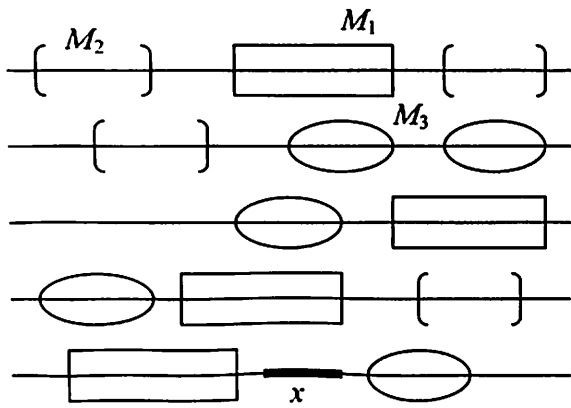
3. Sample one of motif models (including background model M_0) proportional to a likelihood, $L(x, M_i)$ that the selected site x is derived from the models and given as, $L(x, M_i) = (p_i / (1 - p_i)) \prod_{j=1}^W (q_{i,j,r_j} / q_{i,r_j})$, where $p_i / (1 - p_i)$ is a posterior probability that x belongs to the model M_i and p_i is obtained initially by e_i / C_i , where C_i is the total number of possible site in sequences, $C_i = \sum_{m=1}^k \max(0, u_m - L_i + 1)$ and u_m is a length of s_m . p_i is updated as the iteration goes on; $p'_i = (c_i + a_i) / (C_i + A_i)$, where c_i is the number of motifs in the motif models and a_i and A_i are pseudocounts and they are given as, $a_i = (e_i \cdot W) / (1 - W)$, $A_i = (C_i \cdot W) / (1 - W)$ and $W = 0.8$ by default. q_{i,j,r_j}

is a target probability of observed base r_j that is in selected site x at position j of model M_i and q_{i,r_j} is its background target probability.

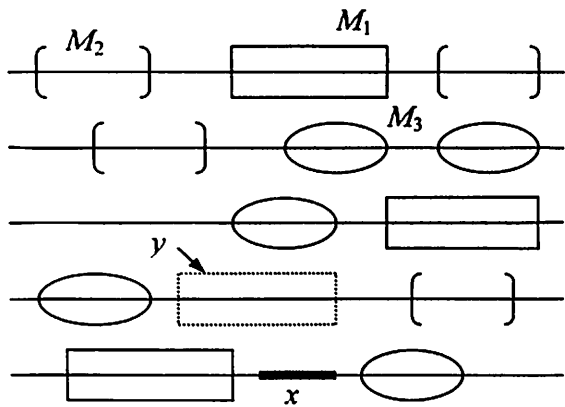
4. [Column Sampling] Select a motif y randomly or proportional to how poor the information content of the motif is in the previously selected motif model M_i . Remove the motif y temporary from the alignment.
5. Select a site z with the largest width that does not belong to any motif models. Calculate information contents with all possible $(L_z - L_i + 1)$ motif width within z and sample a site z' proportional to how rich the information content of motif is, $\prod_{j=1}^{L_i} [\Gamma(n_{i,r_j} + b_{r_j}) / q_{i,r_j}]$, Γ function is obtained from [Liu *et al.* 95].
6. If the selected site z' is subsequence (or the same) of x then replace y with z' , otherwise y is restored in Model i and z' is used as motif x in Motif Sampling in the next iteration. The step 2 to 6 is repeated until it converges.

Example of Gibbs Motif Sampling:

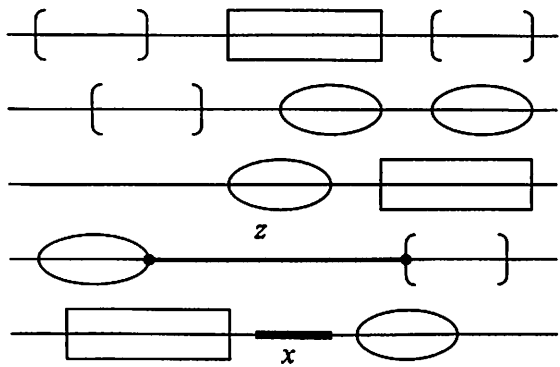
Figure 14 demonstrates Gibbs Motif Sampling step by step.



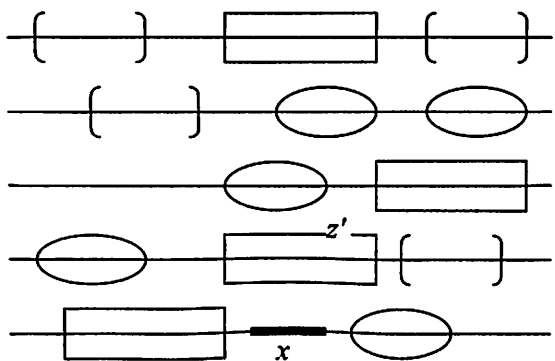
(1) Initially, motif sets are assigned in a group of sequences (Step 1). In this sample, rectangles represent M_1 , brackets represent M_2 and ovals represent M_3 and the background becomes M_0 . A site x is selected randomly (or succeeded from the last iteration) and a motif model that x belongs the most likely is selected by Motif Sampling (Step 2 and 3). In this example, M_1 is selected



(2) A site y is randomly selected from M_1 and y is temporary removed from the alignment (Step 4).

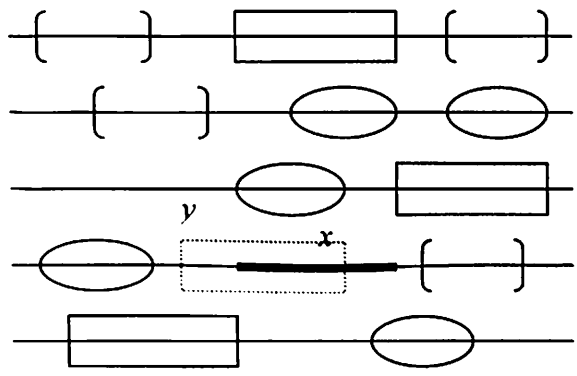


(3) The largest width site that belongs to background model M_0 is selected as z (Step 5).

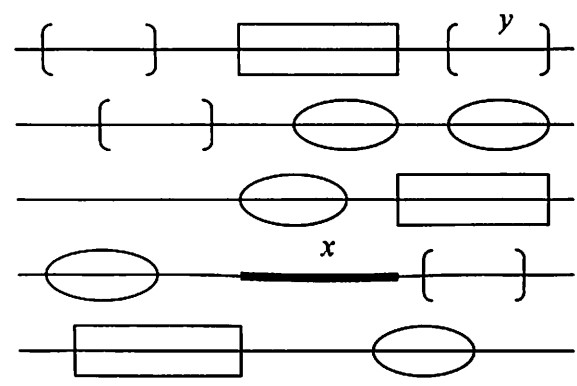


(4) Within the region z , a higher information content site with width of M_1 is selected as z' by Column Sampling (Step 5). However, the site z' is not in the same region with the site x .

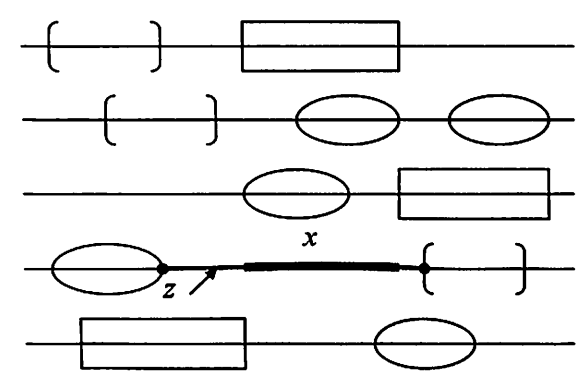
Figure 14. Example of Gibbs Motif Sampling (continue).



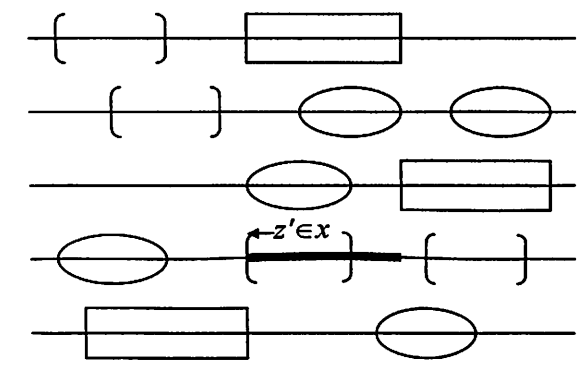
(5) The samplings failed. The site y is restored into M_1 and the site z' becomes the next target site x (Step 6). Since the site x overlaps with alignment motif model M_1 , the site y is removed from the alignment (Step 1).



(6) Motif sampling detected that x is the most likely in M_2 (Step 2 and 3) and randomly a site y is selected from M_2 (Step 4).



(7) Temporary remove the site y and a site z is selected as the largest width site from background model M_0 .



(8) By Column Sampling, a site z' is detected (Step 5) and the site z' shares the same region with the site x .

Figure 14. Example of Gibbs Motif Sampling (continue).

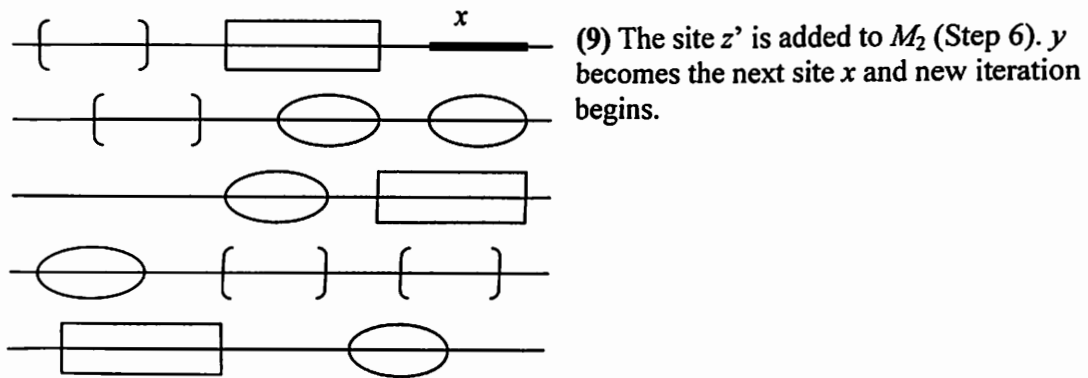


Figure 14. Example of Gibbs Motif Sampling.

5. CONCLUSION

To summarize characteristics and differences of the algorithm studied, two tables are developed as follows: comparisons of algorithms in clustering (Table 2) and motif finding (Table 3).

In general, the first-generation clustering algorithms: hierarchical clustering, *K*-means algorithm and SOM, are popular among erbiologists because they had been applied in some other fields' data mining before and the implementation is simple [Moreau *et al.* 02]. However there are some cases that they cannot handle, such that there are some vectors in a small range and quality guaranteed clustering is required, datasets involves some experimental errors etc. The second generation clustering algorithms are especially developed for clustering over gene expression profiles to manage these cases. The algorithms are tend to be more computationally expensive as they offer more flexibility on their performance; however, a priori knowledge on datasets may overcome the drawback. [Yeung *et al.* 01] states that the model based clustering can reduce computational time by customizing methods for well-known or common experiments.

Motif finding algorithms have to handle varieties of motif types: mononucleotide

model, multiple motif models, gapped motifs, palindrome motifs and widely spread weak motif etc.; algorithms were developed to manage each type of motif. [Helden *et al.* 98] states that probabilistic method such as Gibbs sampling is not always efficient. String method is considered to be naïve in terms of its computational efficiency; however, probabilistic methods may take about the same time, because probabilistic methods tend to be trapped by local optima and we need to run programs several time with different initial conditions to validate the result. In spite of its exhaustive approach, string-based method is preferred when we detects short motifs.

On the other hand, more and more powerful probabilistic methods are developed. EM algorithm gave us a prospective approach by alternating estimation and maximization of the likelihood of motif positions, once the algorithm captures a correct pattern in a set of motif, next iterations favor further correct patterns [Neuwald *et al.* 95]. As the iteration goes on, correct patterns dominate the set of motif and improve its likelihood. A maximization of likelihood is applied to different features of motif sets (not only positions of motifs but also width of motifs, etc.) and new algorithm such as motif sampling can detect similarities among sequences that past algorithms could not detect.

Every microarray experiment data has different nature of similarities and available prior knowledges, therefore, many different types of clusterings and motif

finding algorithms are applied. This thesis explored some representative algorithms used in microarray technology today.

	Nearest Neighbor Clustering			Agglomerative Clustering		Divisive Clusterings	
	K-means	SOM	Model-based	Hierarchical	Quality-based	SOTA	Adaptive Quality-based
User-defined Parameter	The number of expected clusters	The number of expected clusters and a simple geometric shape of grids for default	N/A	N/A	The number of expected clusters and maximum cluster diameter (threshold for jackknife correlation)	A threshold that indicate minimum cluster diameter	The minimum number of clusters
Data Structure	n -dimensional space (n=the number of observations)	n -dimensional space (n=the number of observations)	multi-dimensional space	Binary Tree	n -dimensional space (n=the number of observations)	Binary Tree	n -dimensional space (n=the number of observations)
Additional Required Information	Comparison of result from several different runnings.	Internal parameters and comparison of result from several different runnings.	Biological correlation of genes (to preprocess data with hierarchical clustering) and comparison of different models	Biological correlation of genes and where to cut the tree to obtain clusters.	Internal parameters	Where to cut the tree to obtain clusters.	Internal parameters
Statistical Definition of Clusters	N/A	N/A	Likelihood Analysis and Bayesian Information Criterion	N/A	Likelihood Analysis	N/A	Likelihood Analysis
Inclusion of All Genes	Yes	Yes	Yes	Yes	No	Yes	No
Is Algorithm Stable?	No	No	Yes	No	Yes	No	Yes
Missing Values Handling	No	No	Yes	Yes	Yes	No	Yes
Computational Complexity	Linear	Linear	Quadratic	Quadratic	Quadratic	Linear	Linear

Table 2. Comparison of Clustering Algorithms.

	String Method	Probabilistic Methods			
		Greedy Algorithm	EM Algorithm	Gibb's Sampling	Gibb's Motif Sampling
User-defined Parameter	Motif width	Motif width	Motif width and proposed motif models	N/A	Motif width, the number of motif models and the expected number of motifs in each model.
Available Type of Motifs	Short Motifs	Mononucleotide model (a sequence has only one motif and motifs are independent each other)	Validating proposed motif models (mononucleotide model, correlated palindromic or gapped motifs)	Mononucleotide model, gapped motifs, widely spread weak motifs	Detecting multiple motif models simultaneously (mononucleotide model, gapped motifs, widely spread weak motifs)
Computational Complexity	Quadratic	Quadratic	Linear	Linear	Linear

Table 3. Comparison of Motif Finding Algorithms.

6. REFERENCES

- [Akutsu 98] Akutsu Tetsuya, "Hardness results on gapless local multiple sequence alignment" Technical Report 98-MPS-24-2, *Information Processing Society of Japan*, 1998.
- [Akutsu *et al.* 00] T. Akutsu, H. Arimura and S. Shimozone, "On approximation algorithms for local multiple alignment" In *RECOMB00: Proceedings of the Fourth Annual International Conference on Computational Molecular Biology*, Tokyo, Japan, April 2000.
- [Altschul *et al.* 90] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, D. J. Lipman, "Basic Local Alignment Search Tool," *J. Mol. Biol.* 215 pp. 403-410, 1990.
- [Ben-Dor 99] A. Ben-Dor and Z Yakhini, "Clustering Gene Expression Patterns," In *RECOMB99: Proceedings of the Thrid Annual International Conference on Computational Molecular Biology*, Lyon, France, pp. 33-42 1999.
- [Bilmes 98] Bilmes, J. A., "A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models" International Computer Science Institute, 1998.
- [Cho *et al.* 98] R. J. Cho, J. J. Campbell, E. A. Winzeler, L. Steinmetz, A. Conway, L. Wodicka, T. G. Wolfsberg, A. E. Gabrielian, D. Landsman, D. J. Lockart, *et al.*, *Mol. Cell* 2, 65-73, 1998.
- [De Smet *et al.* 02] F. De Smet, J. Mathys, K. Marchal, G. Thijs, B. De Moor, and Y. Moreau, "Adaptive quality-based clustering of gene expression profiles," *Bioinformatics*, vol. 18, no. 5, pp. 735-746, 2002.
- [Dempster *et al.* 77] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum-likelihood from incomplete data via the em algorithm," *J. Royal Statist. Soc. Ser. B.*, 39, 1977.
- [Dopazo 97] J. Dopazo and J. M. Carazo, "Phylogenetic reconstruction using a growing neural network that adopts the topology of a phylogenetic tree," *J. Mol. Evol.*, 44, pp. 226-233, 1997.
- [Efron 82] Efron, B. "The Jackknife, the Bootstrap, and Other Resampling Plans," CBMS-NSF Regional Conference Series in Applied Mathematics; 38. *Society for Industrial & Applied Mathematics*, 1982.

- [Eisen *et al.* 98] M.B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein, "Cluster analysis and display of genome-wide expression patterns," in *Proc. Nat. Acad. Sci. USA*, vol. 95, pp.14863-14868, 1998.
- [Ermolaeva *et al.* 98] O. Ermolaeva, M. Rastogi, K. D. Pruitt, G. D. Schuler, M. L. Bittner, Y. Chen, R. Simon, P. Meltzer, J. M. Trent, and M. S. Boguski. Data management and analysis for gene expression arrays. *Nature genetics*, vol. 20(1), pp. 19-23, 1998.
- [Fraley 99] C. Fraley and E. Rfery, "MCLUST: Software for model-based cluster analysis," *J. Classification*, vol. 16, pp. 297-306, 1999.
- [Gosh 02] D. Ghosh and A. M. Chinnaiyan, "Mixture modeling of gene expression data from microaaray experiments," *Bioinformatics*, vo. 18, pp. 275-286, 2002.
- [Helden *et al.* 98] J. van Helden, B. André, and L. Collado-Vides, "Extracting regulatory sites from upstream region of yeast genes by computational analysiss of oligobase frequencies," *J. Mol. Biol.*, vol. 281, pp. 827-842, 1998.
- [Herrero *et al.* 01] J. Herrero, A. Valencia, and J. Dopazo, "A hierarchical unsupervised growing neural network for clustering gene expression patterns," *Bioinformatics*, vol. 17, pp. 126-136, 2001.
- [Hertz 99] G. Z. Hertz and G. D. Stormo, "Identifying DNA and protein patterns with statistically significant alignments of multiple sequences," *Bioinformatics*, vol. 15, no. 7/8, pp. 563-577, 1999.
- [Heyer *et al.* 99] L. J. Heyer, S. Kruglyak, and S. Yooseph, "Exploring expression data: Identification and analysis of coexpressed genes," *Genome Res.*, vol. 9, pp. 1106-1115, 1999.
- [Hubert 85] L. Hubert and P. Arabie, "Comparing Partitions," *Journal of Classification*, pp. 193-218, 1985.
- [HGP 04] Human Genome Project Information "The science behind the human genome project," creation date: March 2004 (access date: March 2004) http://www.ornl.gov/sci/techresources/Human_Genome/project/info.shtml.
- [Kohane *et al.* 03] I. S. Kohane, A. T. Kho, and A. J. Butte, "Microarrays for an Integrative Genomics," The MIT Press, Cambridge, Massachusetts, 2003.
- [Lawrence *et al.* 90] C. E. Lawrence and A. A. Reilly, "An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences," *Proteins*, vol. 7, pp. 41-51, 1990.
- [Lawrence *et al.* 93] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton, "Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment," *Science*, vol. 262, pp. 208-214, 1993.

- [Liu *et al.* 95] J. S. Liu, A. F. Neuwald, and C. E. Lawrence, "Bayesian models for multiple local sequence alignment and Gibbs sampling strategies," *J. Amer. Statist. Assoc.*, vol. 90, no. 432, pp. 1156-1170, 1995.
- [Moreau *et al.* 02] Y. Moreau, F. D. Smet, G. Thijs, K. Marchal, B. D. Moor, "Functional bioinformatics of microarray data: from expression to regulation," *Proc. IEEE*, vol. 90, No. 11 November, 2002.
- [Neuwald *et al.* 95] A. Neuwald, J. S. Liu and C. E. Lawrence, "Gibbs motif sampling: Detection of bacterial outer membrane protein repeats," *Proteins Science*, vol. 4, pp. 1618-1632, 1995.
- [NOVA 01] *NOVA Online*, "Science programming on air and online," creation date: April 2001 (access date: March 2004), <http://www.pbs.org/wgbh/nova/genome/program.html>.
- [Quackenbush 01] Quackenbush, J. "Computational analysis of microarray data," *Nat. Rev. Genetics*, vol. 2, pp. 418-427, 2001.
- [Schwarz 78] G. Schwarz "Estimating the dimension of a model," *Ann. Stat.*, 6 pp. 461-464, 1978.
- [Staden 89] Staden, R. "Methods for discovering novel motifs in nucleic acid sequences," *Computer Applications in the Biosciences* 5(4), pp. 293-298, 1989.
- [Sokal 58] Sokal, R. R. and Michener, C. D. *Univ. Kans. Sci. Bull.* 38, 1409-1438, 1958.
- [Tamayo *et al.* 99] 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," *Proc. Nat. Acad. Sci. USA*, vol. 96, pp. 2907-2912, 1999.
- [Tavazoie *et al.* 99] S. Tavazoie, J. D Hughe, M.J. Campbell, R. J. Cho and G. M. Church, "Systematic determination of genetic network architecture," *Nature Genetics*, 22(3): 281-5, 1999, June 1999.
- [Tompa 99] M. Tompa, "An exact method for finding short motifs in sequences, with application to the ribosome binding site problem," in *Proc 7th Intl. Conf. Intelligent Syst. For Mol. Biol.*, Heidelberg, Germany, pp. 262-271, August 1999.
- [Tompa 00] M. Tompa, "Lecture Notes on Biological Sequence Analysis," creation year: 2000 (access date: February 2004), <http://www.cs.washington.edu/education/courses/527/00wi/lectures/roottr.pdf>.
- [Yeung *et al.* 01] K. Y. Yeung, C. Fraley, A. Murua, A. E. Raftery, and W. L. Ruzzo, "Model-based clustering and data transformations for gene expression data," *Bioinformatics*, vol. 17, pp. 309-318, 2001.

6. APPENDIX: PSEUDOCODES LISTINGS

K-means Clustering

```
K_means_Clustering(X, numOfClusters){
/* A set of vectors X and a set of cluster centers Y is represented by an object arrays that has
vectors. Clusters are represented by linked-list array: CLUSTER[ ]. */
/* Step 1*/
/* randomly partition n-dimensional space ---> randomly assign cluster centers */
for(i = 1; i <= numOfClusters ; i++)
    for(j = 1; j <= numOfObservations ; j++)
        Y[i].set(j, Random_number_generator());
do{ /* until the algorithm has converged */
/* copy and initialize the new cluster centers Y for the convergence test */
oldY = Y;
Y.initialize();
/* for all vectors, assign to the nearest cluster center's cluster */
for(i = 1; i <= numOfGenes ; i++)
    CLUSTER[MinEuclideanDistance(X, i, Y)].add(X[i]);
/* Step 2: [Estimation Step] recalculate the cluster */
ComputeClusterCenter (CLUSTER, Y);
/* Step 3: [Maximization Step] reassign the vectors X into clusters corresponding to the new
cluster centers. */
for(i = 1; i <= numOfGenes ; i++)
    CLUSTER[MinEuclideanDistance(X, i, Y)].add(X[i]);
}while(Compare(oldY, Y) == false); /* compare Y if it was the same as last time or not */
return CLUSTER;
} /* end function K_means_Clustering */

/* return the index of nearest cluster center's cluster Y[k] */
MinEuclideanDistance(X, i, Y){
for(k= 1; k <= numOfClusters ; k++)
    Distance[k]=EuclideanDistance(X[i],Y[k]);
if(Distance[min] > Distance[k])
    min = k;
return min;
}/* end function MinEuclideanDistance */

/* function EuclideanDistance returns the distance between two points */
EuclideanDistance(x, y){
for(j = 1; j <= numOfObservations ; j++)
    d =(x[i].get_x(j)+ y[k].get (j))2;
    Distance[k]=Distance[k]+d;
d =sqrt(Distance[k]);
return d;
}/* end function EuclideanDistance */
```



```

/* function Compares two arrays if they are identical or not, if identical return true */
Compare(oldY, Y){
  while(oldY[i]!=null)
    for(j = 1; j <= numOfObservation ; j++)
      if(oldY[i].get(j) == Y[i].get(j))
        i++;
      else
        return false;
  return true;
}/* end function Compare */

ComputeClusterCenter (CLUSTER, Y){
for(i = 1; i <= numOfClusters; i++)
  for(j = 1; j <= numOfObservation ; j++)
    while(CLUSTER[i].get_x() != null)
      sum_X = sum_X +CLUSTER[i].get_x(j);
      counter++; /* count the number of vector in a cluster */
      CLUSTER[i].get_next();
      Y[i].set(j, sum_X / counter); /* take an average and store in each observation */
      sum_X = 0; counter = 0; /* initialization for the next observation */
return Y;
} /* end function ComputeClusterCenter */

```

SOM (self-organizing map)

```

SOM(X, numOfClusters, Y){
/* A set of vectors X and a set of cluster centers initial geometric shape grids Y is represented
by an object array */
counter = 1;
/* Step 2-1: randomly select one of gene vectors and find the nearest cluster center.
indexOfX[ ] has flag to tell if the index X[i] has already been selected or not. */
do{ /* until iteration reaches the maximum number of times */
  /* copy and initialize the new cluster centers Y for the convergence test */
  oldY = Y;
  Y.initialize();
  i = Random_number_generator();
  while(indexOfX[i] == 1) /* X[i] has already been selected, draw another i */
    i = Random_number.generator();
  indexOfX[i] = 1/* set a flag for already drawn gene X[i] */
/* function MinEuclideanDistance returns index of Y that is the nearest cluster. */
p = MinEuclideanDistance(X, i, Y);
/* Step 2-2: update cluster centers Y proportional to the distance to x, */
/* functionP decreases (from 3) linearly with some function with respect to counter (the
number of iteration). Other coefficients are given above. */
learningRate=0.02*MaxNumIteration/ (MaxNumIteration+100*functionP(counter));
for(j= 1; j <= numOfObservation ; j++)
  newLocation=Y[k].get(j)+learningRate*EuclideanDistance(Y[p],Y[k])*
(EuclideanDistance(X[i], Y[k]));
  Y[k].set(j, newLocation);
  counter++;
}

```

```

    if(Compare(oldY, Y) == true) /* convergence test */
        return Y;
} while(count < MaxNumIteration);
return Y;
} /* end function SOM */

```

Model-based Clustering

```

Model_based_Clustering(X){
/* A set of vectors, X is represented by an object array. result[ ][ ] has clustered vectors in
linked list */
/* Step 1: preprocess the data with hierarchical clustering according to logarithmic likelihood
*/
Hierarchical_Clustering(X);
/* Step 2: try n possible models for different number of clusters 2 to k */
for(t = 2; t <=k; t++)
    result[1][t] = K_means_Clustering(X, k);
    result[2][t] = ...../* try other models */
    :
    result[n][t] = .....

/* Step 3: calculate BIC (Bayesian Information Criterion) and print them for further analysis
*/
/* BIC compares all possible n models in one time */
for(t = 2; t <=k; t++) /* for each number of clusters */
    Print(CalculateBIC(t, result[1][t], result[2][t], ..., result[n][t]));
} /* end function Model_based_Clustering */

```

Hierarchical Algorithm

```

Hierarchical_Clustering(X){
/* A set of vectors X is represented by an object array (size in  $2^{(\text{ceil}(\lg N)+1)-1}$ , where
NumOfGenes = N) and stored in biologically related order. Each object has pointers to a
dataset table (DistanceTable) and their child nodes, and a vector x. When the vectors are
merged and their parent node is created, the vectors' obsolete flag becomes true. */
do{ /* until there is only one gene in a data table */
/* Step 1: Construct Euclidean distance table */
for(i= 1; i <= numOfGenes ; i++)
    if(X [i].obsolete() != true) /* if the vector i is not obsolete, compare with other vectors */
        for(j= 1; j <= numOfGenes ; j++)
            if(X [j].obsolete() != true)
                if(i!=j)
                    distanceTable[i][j]= ∞ ;
                else
                    distanceTable[i][j]=EuclideanDistance(X[i], X [j]);
/* Step 2: Merge the shortest distance pair into one gene vector */
shortestDistance = DistanceTable[1][1]; /* initialize the shortest distance value */

```

```

for(i= 1; i <= numOfGenes ; i++)
    for(j= 1; j <= numOfGenes ; j++)
        if(distanceTable[i][j] < shortestDistance)
            shortestDistance = distanceTable[i][j];
/* merge X[i] and X[j] into new (parent) node X[numOfGenes +1] */
if(i == j) /* case there is only one gene in the distance table */
    return X;
else
    for(k= 1; k <= numOfObservation ; k++) /* for all observations take an average */
        X [numOfGenes].set_x((X[i].getX(k) + X[j].getX(k))/2, k);
numOfGenes++; /* new node created */
/* set pointers of parent node to a merged pair X[i] and X[j]. */
X[numOfGenes].setLCPointTo(X[i]);
X[numOfGenes].setRCPointTo(X[j]);
/* X[i] and X[j] are obsolete since they are merged.*/
    X[i].setObsolete();
    X[j].setObsolete();
}while(i != j);
} /* end function Hierarchical_Clustering */

```

Quality-based Clustering

```

Quality_based_Clustering(X, minNumGenes, diameter_d){
/* A set of vectors X is represented by an object array and cluster centers are stored in array Y.
Clusters are represented by object array CLUSTER[ ] and it has a gene index and vectors in
the linked-list array. Jackknife correlation is stored in two-dimensional array. When gene is
already selected, set ∞ to avoid being selected by the same cluster center. */
/* Step 1: preprocess the data and obtain standardized X */
for(i = 1; i <= numOfGenes; i++)
    sum = 0;
    for(j = 1; j <= numOfObservation; j++)
        sum = sum + X[i].get_x(j);
    mean = sum/ numOfObservation;
    sum = 0;
    for(j = 1; j <= numOfObservation; j++)
        sum = sum + (mean-X[i].get_x(j))^2;
    sigma = sqrt(sum/numOfObservation-1);
    /* obtain standard deviations */
    for(j = 1; j <= numOfObservation; j++)
        X[i].set_x(j, X[i].get_x(j)-mean);
        X[i].set_x(j, X[i].get_x(j) / sigma);
/* Step 2: Calculate jackknife correlation */
for(i = 1; i <= numOfGenes; i++)
    for(j = 1; j <= numOfGenes; j++)
        min = EuclideanDistance(X[i].get_x(j)); /* initialize minimum value */
        for(k = 1; k <= numOfGenes+1; k++)
            /* function EuclideanDistance has polymorphism functions and when it has the
            second argument, it eliminates term in index k when it calculates distance */
            if(i != j)
                d = EuclideanDistance(X[i].get_x(j), k);

```



```

        if(d < min)
            min = d;
        jackknife[i][j] = min;
    /* Step 3: find a greatest jackknife correlation gene to every one of genes */
    /* find a pair of genes  $x_i$  and  $x_j$  that have a minimum jackknife correlation and set the average
    as cluster center of cluster  $C_i$  */
    do { /* until there are no more clusters that meet minimum number of vectors */
        t++; /* increment an index of array CLUSTER */
        tempCLUSTER.empty(); /* initialize a temporary cluster set */
        for(i = 1; i <= numOfGenes; i++){ /* all genes can be a temporary cluster center  $i$  */
            /* use a temporary jackknife for temporary CLUSTER sets */
            /* the same genes can be selected by different index  $i$  in temporary CLUSTER[j] */
            tempJackknife = jackknife; /* reset tempJackknife to be original jackknife */
            min = tempJackknife[i][1]; /* initialize minimum correlation */
            if(tempJackknife[i][1] <= 1) /* if the gene does not belong to any cluster yet */
                for(p = 2; p <= numOfGenes; p++)
                    if (min > tempJackknife[i][p] && tempJackknife[p][1] !=  $\infty$  && diameter_d >
                        tempJackknife[i][p] && i != p)
                        min = tempJackknife[i][p];
            tempCLUSTER[i].add(X[i]); /* first, add the current cluster center  $i$  */
            if(diameter_d > tempJackknife[i][p])
                tempCLUSTER[i].add(X[p]); /* second, add selected vector */
            /* set null so that next time it is not selected in next step on current cluster center */
            tempJackknife[p][1] =  $\infty$ ;
            ComputeClusterCenter(tempCLUSTER, Y);
            cluster_center = Y[i];
            /* Step 4 */
            do { /* until there are no more vectors that meets threshold  $d$  */
                min = EuclideanDistance(tempJackknife[i][1], cluster_center);
                for(q = 2; q <= numOfGenes; q++)
                    if (i != q && tempJackknife[q][1] !=  $\infty$  && min >
                        EuclideanDistance(tempJackknife[i][q], cluster_center))
                        min = EuclideanDistance(jackknife[i][q], cluster_center);
                if(min < diameter_d)
                    tempCLUSTER[i].add(X[q]);
                    /* set null so that the next time it is not selected in this for loop */
                    tempJackknife[q][1] =  $\infty$ ;
                    ComputeClusterCenter(tempCLUSTER, Y);
                    cluster_center = Y[i];
            } while(min < diameter_d);
        } /* end for  $i$  */
    /* Step 6 */
        max = 0; maxNumOfGenes = 0;
        for(i = 1; i <= numOfGenes; i++){ /* find the largest cluster */
            if(maxNumOfGenes < tempCLUSTER[i].count()){
                maxNumOfGenes = tempCLUSTER[i].count();
                max = i;
            }
            if(maxNumOfGenes >= minNumGenes )
                CLUSTER[t] = tempCLUSTER[max];
        /* set to selected genes to be null so that the gene  $k$  is not selected in further iteration */
        do {

```

```

        jackknife[CLUSTER[t].get_index()][1] = 2;
    }while(CLUSTER[t].get_next() != null);
    tempJackknife = jackknife; /* renew jackknife correlation table */
/* if there still cluster that satisfies minimum number of genes, continue the loop */
}while(maxNumOfGenes >= minNumGenes);
return CLUSTER;
} /* end function Quality_based_Clustering */

EuclideanDistance(x, y, k){
    for(j = 1; j <= numOfObservations; j++)
        if(j!=k)
            d =(x[i].get_x(j)+ y[k].get (j))2;
            d =sqrt(Distance[k]);
    return d;
}/* end function EuclideanDistance */

```

SOTA (self-organizing tree algorithm)

```

SOTA(X, threshold_h){
/* A set of vectors X is represented by an object array and cluster centers are stored in array Y
that forms tree. Y is array object that has vectors X and the index of parent and sibling */
/* Step 1: create the root by taking an average of all vectors */
for(i=1; i<= numOfGenes; i++)
    CLUSTER[i].add(X[i]);
ComputeClusterCenter(CLUSTER, Y);
/* Step 2: create two child nodes from the root */
    root = Y[1];
    copy(Y[1], Y[2]);
    copy(Y[1], Y[3]);
    Y[2].setParentTo(1)
    Y[2].setSiblingTo(3)
    Y[3].setParentTo(1)
    Y[3].setSiblingTo(2);
/* Step 3 */
do{ /* until it converges */
    old_e = e; /* initialize error rate for the convergence test */
    counter = 1;
    do{ /* until all vectors are selected; one epoch */
        /* randomly select X[i] */
        i = Random_number_generator();
        while(indexOfX[ i ] == 1) /* i has already been selected, draw another i */
            i = Random_number.generator();
        indexOfX[ i ] = 1
        /* function MinEuclideanDistance returns index of Y that is the nearest cluster to X[i] */
        winning = MinEuclideanDistance(X, i, Y);
        /* Update the winning cluster, its parent and sibling with proportional to the distance to
        X[i], coefficients are obtained from step 3. */
        R[winning] = R[winning] + X[i].get_x(j) -Y[winning].get_x(j)];
        countVectors[winning] ++; /* increment counter of vectors in a cluster */
    }
}

```

```

Y[winning].set(j, Y[winning].get_x(j) + 0.01*(X[i].get_x(j) - Y[winning].get_x(j)));
node = Y[winning][j].getSibling();
node.set(j, node.get_x(j) + 0.001*(X[i].get_x(j) - node.get_x(j)));
node = Y[winning][j].getParent();
node.set(j, node.get_x(j) + 0.005*(X[i].get_x(j) - node.get_x(j)));
} while(counter != numOfGenes); /* one epoch */
/* Step 4 */
k = 1;
while(k <= numOfGenes && counter[k] > 0)
    R = R[k]/countVectors[k++];
k = 1;
while(k <= numOfGenes && counter[k] > 0)
    e = R[k++] + e;
---initialized array R and counter ---
} while((e - old_e) / old_e >= threshold_h);
return Y; /* process is converged */
} /* end function SOTA */

```

Adaptive Quality-based Clustering

```

Adaptive_Quality_Based_Clustering(X, minNumOfGenes, significance_level_h){
/* A set of vectors X and a set of cluster centers Y is represented by an object arrays that has
vectors. Clusters are represented by linked-list array: CLUSTER[ ]. */
/* Step 1 */
min_radius = sqrt(numOfObservation - 1) / 2;
/* Step 2 */
do{ /* until there is cluster that satisfies the minimum number of vectors */
    IterationCounter = 1;
    do{ /* until locate cluster into proper place */
        for(i=1; i <= numOfGenes; i++)
            if(X[i].obsolete() == false)
                CLUSTER[j].add(X[i]);
        ComputeClusterCenter(CLUSTER, Y);
        cluster_center = Y[j];
        if(IterationCounter == 1)
            delta_radius = (maxEuclideanDistance(Y, j, X) - min_radius) / fraction_radius;
        /* shrink the Cluster by delta_radius */
        radius = maxEuclideanDistance(Y, j, X) - delta_radius;
        /* clear cluster once and pick up vectors that are inside the cluster */
        CLUSTER[j].delete();
        for(i=1; i <= numOfGenes; i++)
            if(EuclideanDistance(cluster_center, X[i]) < radius)
                if(X[i].obsolete() == false)
                    CLUSTER[j].add(X[i]);
        /* Step 3 */
        old_cluster_center = cluster_center; /* keep last cluster center for convergence test */
        ComputeClusterCenter(CLUSTER, Y);
        cluster_center = Y[j];
        /* remove all genes in cluster j from the space */
    }
}

```

```

    if(Compare(old_cluster_center, cluster_center) == false)
        if(Distance(cluster_center, X[i]) < radius)
            X[i].set_obsolete();
} while(Compare(old_cluster_center, cluster_center) == true || IterationCounter <= 50 ||
        radius > min_radius);
if(CLUSTER[j].empty() == true) /* Step 6 */
    return CLUSTER; /* relocating cluster does not converge */
/* Step 5 */
radius = min_radius; newRadius=radius; /* initialize radius */
/* obtain Pc from distribution of distance between cluster center and vectors */
while(radius != lastRadius){
    /* Estimation Step */
    /* measure distance between vector i that belongs to the cluster and the cluster center
    and, and calculate likelihood. */
    for(i= 1; i<= numOfGenes; i++)
        if(Distance(cluster_center, X[i]) < radius)
            if(X[i].obsolete() == false) /* if vector is not taken by other cluster already */
                CenterToVectors = Distance(cluster_center, X[i]) + CenterToVectors;
                countVectors++;
            else
                if(X[i].obsolete() == false)
                    CenterToBackground=Distance(cluster_center,X[i])+
                                                CenterToBackground;

    if(countVector < minNumOfVectors) /* cluster does not have enough vectors */
        radius=lastRadius;
        break;
    else
        readus=newRadius;
        Pc = CenterToVectors/CenterToBackground;
        Pb = 1 - Pc;
        mean = Pc_b/numOfGenes;
        sum = 0;
        for(i= 1; i<= numOfGenes; i++)
            for(j = 1; j <= numOfObservation; j++)
                sum = sum + (mean-X[i].get_x(j))2;
        sigma = sqrt(sum/numOfObservation-1);
        PrC=Formula1(Pc,sigma);
        PrB= Formula2(Pb,sigma);
        /* Maximization Step */
        newRadius = Formula3(PrC,PrB,significance_level_h);
        lastRadius = newRadius; /* keep the last radius for convergence test */
} /* end while */
/* empty cluster once, pick up all vectors within the radius and store them in cluster */
CLUSTER[j].delete();
CountGenes = 0;
if(EuclidianDistance(cluster_center, X[i]) < radius)
    if(X[i].obsolete() == false)
        CLUSTER[j].add(X[i]);
        /* genes in the cluster are removed from the space for the next iteration */
        X[i].set_obsolete();
        countGenes++;

```

```

        j++; /* obtain next cluster */
    }while(countGenes >= minNumOfGenes);
    return CLUSTER; /* Step 6 */
}/* function Adaptive_Quality_Based_Clustering */
/* Refer to formulas (1), (2) and (3) above for subfunctions Formula1, Formula2 and
Formula3. */

```

String Method

```

String_Method(S, L){
/* A set of sequences S is stored in two-dimensional string array and significant motifs with
length of L is stored in an object array MOTIF that has motif sequences and z-scores */
/* Step 1: By Staden's algorithm count the occurrence of all possible 4L strings N_t [ ] and the
number of sequences that contain at least one motif candidate t of N_sq [ ]. Assume array
dictionary[ ] already contains all possible string of length L with {A,T,C,G} */
t=1;
while(t <= 4L){/* for each possible string t */
    for(m=1; m<=numOfSequence; m++){/* for all sequence in S */
        for(j=1; j <= lengthOfSequence-(L+1); j++){/* for all location in sequence s */
            for(k=1; k <= L; k++){/* for all index of string t */
                if(S[m][k+j] == dictionary[t][k])
                    matched++;
            }
            if(matched >= L-1) /* string t found within S[m] and at most one substitution */
                N_t[t]++;
            flag = true; /* there is at least one string t found in sequence S[m] */
            matched = 0;
        } /* end for j */
        if(flag == true)
            N_sq[t]++;
        flag = false;
    } /* end for m */
    t++; /* go to the next possible string t */
} /* end while */
/* Step 2: construct DNF and find probability p_t */
do{ /* do for all possible string t */
    /* Step 2-1: construct DNF that accepts string t with sequence m */
    M = ConstructDNF(dictionary, t, S, m);
    /* Step 2-2: construct object array A [ ] [ ] that has a pattern ai and ak and count of
occurrence, as a first order Markov Chain */
    m = Random_number_generator(); /* randomly select a sequence from S */
    /* count the number of occurrence of pair ai and ak */
    for(j=1; j <= lengthOfSequence-1; j++)
        for(i=1; i <= 4; i++)
            for(k=1; k <= 4; k++)
                if(A[i][k].get_a_i() == S[m][j])
                    if(A[i][k].get_a_j() == S[m][j+1])
                        A[i][k].increment();
}

```

```

/* to obtain probability divide the count by length of sequence-1 */
for(i=1; i <= 4; i++)
    for(k=1; k <= 4; k++)
        A[i][k].set_prob(A[i][k].get_count()/lengthOfSequence-1);
M = TransformM(M,A[i][k]); /* Step 2-3: transform M to M' */
/* traverse DNF and determine the probability of occurrence of string t in randomly selected
sequence S[m]. */
p_t[t]=Traverse(M);
/* Step 3: calculate z-score */
Z[t] = (N_t[t]-N_sq[t]) / (N_sq[t](1-p_t[t]))1/2;
t++;
}while(t <= 4L);
/* create an object array MOTIF for only significant strings */
s= 1;
for(t=1; t <= 4L; t++)
    if(Z[t] > 1)
        MOTIF[s++] = CreateMotif(Z[t], dictionary[t]);
return MOTIF;
}/* function String_Method */
/* pseudocode for subfunctions: ConstructDNF, TransformM and Traverse are omitted, please
refer to an example in the next section. */

```

Greedy Algorithm

```

Greedy_Algorithm(S,L){
/* A set of sequences S is stored in an object array that has a two-dimensional string array,
starting index and information content of motif candidate in the sequence. Motif is copied into
array MOTIF (S') as soon as it is considered to be significant. Array M has a motif profile of a
set that has already selected as MOTIF and array P has a temporary motif profile and array Q
has the background model. */
/* Step 1 and 2 */
for(k=1; k<= numOfSequences; k++)
    m= MaxInformationContent(S,L,P)
/* once the highest information content is found in sequence m, it is removed from S */
S[m].set_obsolete();
/* subsequence of length L in S[m] is copied into array MOTIF [ ] */
Copy(MOTIF[k], S[m].get_motif());
/* count the number of bases for next iteration */
for(i= S[m].get_startIndex(); i<= S[m].get_startIndex()+L; i++){
    if(S[m].getBase(i)=='A') M[a][i]++;
    if(S[m].getBase(i)=='T') M[t][i]++;
    if(S[m].getBase(i)=='C') M[c][i]++;
    if(S[m].getBase(i)=='G') M[g][i]++;
}/* function Greedy_Algorithm */

/* return array Q that has a background model */
CreateBackgroundModel(S,P){
for(m=1; m<=numOfSequence; m++)
    for(j=1; j<=lengthOfSequence; j++)

```

```

        if(S[m].getBase(j) == 'A') Q[a]++;
        if(S[m].getBase(j) == 'T') Q[t]++;
        if(S[m].getBase(j) == 'C') Q[c]++;
        if(S[m].getBase(j) == 'G') Q[g]++;
    m = Total(P)/L; /* calculate the number of sequence involved so far */
    /* subtract total number of bases in motif from the background */
    for(base=a; base ∈ (a,t,c,g); next base) /* for all different bases */
        Q[base] = (Q[base] - SubTotal(P,base)) / (m * (lengthOfSequences - L));
    return Q;
} /* end function CreateBackgroundModel */

/* returns object that has maximum information content with its starting index */
MaxInformationContent(S,L,P){
/* Array M has the number of bases in the motif set. Array P hold the number of each base
initially, then it has information content of each base */
for(m=1; m <= numOfSequence; m++)
    if(S[m].obsolete() == false)
        for(j=1; j <= lengthOfSequence; j++)
            Copy(P,M); /* initialize base counter array P */
            /* obtain alignment matrix */
            for(i=j; i <= j+L; i++) /* index in array P, i is relative index of j */
                if(S[m].getBase(i) == 'A') P[a][i-j+1]++;
                if(S[m].getBase(i) == 'T') P[t][i-j+1]++;
                if(S[m].getBase(i) == 'C') P[c][i-j+1]++;
                if(S[m].getBase(i) == 'G') P[g][i-j+1]++;
            Q = CreateBackgroundModel(S,P);
            /* obtain information contents */
            IC = 0;
            for(i=j; i <= j+L; i++){ /* index in array P, i is relative index of j */
                for(base=a; base ∈ (a,t,c,g); next base) /* for all different bases */
                    P[base][i-j+1] = (P[base][i-j+1] / (numOfMotifFound + 1)) * log2((P[base][i-j+1] / (numOfMotifFound + 1)) / Q[base]);
                IC = IC + P[a][i-j+1] + P[t][i-j+1] + P[c][i-j+1] + P[g][i-j+1];
            }
            if(max < IC)
                max = IC;
                maxSeq = m; maxStartIndex = j;
S[m].set_start(maxStartIndex);
return m;
} /* end function MaxInformationContent */

Total(P){
    for(base=a; base ∈ (a,t,c,g); next base)
        for(k=1; k <= 4; k++)
            n = n + P[base][k];
    return n;
} /* function Total */

SubTotal(P,base){
    for(k=1; k <= 4; k++)
        n = n + P[base][k];
}

```

```

return n;
}/* function Total */

```

EM Algorithm

```

EM_Algorithm(S,L){
/* A set of sequences S is stored in array of object that has a two-dimensional string array, start
index (that is already given initially) */
do{
/* Step 2: Estimation Step */
/* Count each different bases of each different position in motif candidates. Pr[ ][ ] contains
base frequency of each position of motif */
P = CalculateMotifProfile(S,L)
Pr = CalculateFrequency(S,L,P);
/* Step 3: Maximization Step */
newP = CalculateFunctionP(S,L);
---adjust j (starting positions) to satisfy estimated base frequency Pr ---
while(Compare(P, newP)==false); /* if the function P is not the same */
return S;
} /* function EM_Algorithm */

```

```

CalculateFrequency (S,L,P){
Q = CreateBackgroundModel(S,L);
for(m=1; m<= numOfSequence; m++)
for(i=S[m].get_startIndex(); i<=L; i++)
    base = S[m].getBase(i);
/* i-S[m].get_startIndex()+1 is the index that is considered to be starting point in Bayes
formula */
    Pr[base][j] = Pr[base][j] + Baye'sFormula(S,m,i-S[m].get_startIndex()+1,P);
return Pr;
}/* function CalculateFrequency */

```

```

CalculateMotifProfile(S,L){
for(m=1; m<=numOfSequence; m++)
for(j=1; j<=lengthOfSequence; j++)
    /* obtain alignment matrix (count the number of each base) */
for(i=j; i<=L; i++) /* index in array A, i is relative index with j */
        if(S[m].getBase(i)=='A') P[a][i-j+1]++;
        if(S[m].getBase(i)=='T') P[t][i-j+1]++;
        if(S[m].getBase(i)=='C') P[c][i-j+1]++;
        if(S[m].getBase(i)=='G') P[g][i-j+1]++;
for(base=a; base∈ (a,t,c,g); next base)
for(i=j; i<=L; i++)
    P[a][i-j+1]= P[a][i-j+1]/numOfSequence;
return P;
} /* function CalculateFunctionP */

```

```

BayesFormula(S,m,k,P){
ProbOfPositionK = 1;

```



```

for(i=k; i<=L+k; i++)
    probOfPositionK= probOfPositionK*P[S[m].getBase(i)][i];
ProbOfOtherPositions = 1;
for(i=1; i<=lengthOfSequence; i++)
    ProbOfOtherPositions = ProbOfOtherPositions*P[S[m].getBase(i)][i];
    sum = sum+ ProbOfOtherPositions;
return ProbOfPositionK/ProbOfOtherPositions;
} /* end function BayesFormula */

```

Gibbs Sampling

```

GibbsSampling(S){
/* A set of sequences S is represented by an object array that has a two-dimensional string
array. */
/* Step 1 */
for(m=1; m<=numOfSequence; m++)
    S[m].set_startIndex(Random_number_generator());
/* Step 2: Estimation Step */
do { /* until converges */
    iterationCounter++;
    oldS = S; /* store last motif set for convergence test */
    /* once all sequence has been selected, clear counter and index and start from beginning */
    if(iterationCounter ==numOfSeq+1)
        iterationCounter =1;
        for(q=1; q<=numOfSequence; q++)
            sequenceIndex[q]=0;
    /*select a sequence z that has not been selected randomly */
    z = Random_number_generator();
    while(sequenceIndex[z] == true)
        z = Random_number_generator();
    sequenceIndex[z]= true; /* set a flag to mark the sequence to be already selected */
    P = ObainProfiles(S,z);
    Q = ObtainBackgroundProfile(S,P);
    /* Step 3: Maximization Step */
    /* calculate a probability of starting position j by the information contents measurement
and set it in array IC[ ]*/
    for(j=1; j<=lengthOfSequence; j++)
        S[z].set_startIndex(j);
        IC[j] = CalculateIC(S,P,Q,z);
    --- draw j proportional to the probability calculated above ---
    S[m].set_startIndex(j);
} while(Compare(oldS, S));
} /* function GibbsSampling */

ObainProfiles(S,z){
for(m=1; m<=numOfSequence; m++){
    if(m!=z) /* exclude S[z] */
        /* obtain alignment matrix */
        j= S[m].get_startIndex();

```

```

    for(i=j; i<=j+L; i++){ /* index in array P, i is relative index to j */
        if(S[m].getBase(i)=='A') P[a][i-j+1]++;
        if(S[m].getBase(i)=='T') P[t][i-j+1]++;
        if(S[m].getBase(i)=='C') P[c][i-j+1]++;
        if(S[m].getBase(i)=='G') P[g][i-j+1]++;
    }
    for(i=j; i<= j+L; i++)
        for(base=a; base∈ (a,t,c,g); next base) /* for all different bases */
            P[base][i]=(P[base][i]+pseduoCount)/(numOfSequence-1);
    return P;
} /* function ObtainProfile */

ObtainBackgroundProfiles(S,z){
    for(m=1; m<=numOfSequence; m++)
        if(m!=z) /* exclude S[z] */
            /* obtain alignment matrix */
            j= S[m].get_startIndex();
            for(i=1; i<=lengthOfSequence; i++){
                if(i< S[m].get_startIndex() && i> S[m].get_startIndex()+L){
                    if(S[m].getBase(i)=='A') P[a][i]++;
                    if(S[m].getBase(i)=='T') P[t][i]++;
                    if(S[m].getBase(i)=='C') P[c][i]++;
                    if(S[m].getBase(i)=='G') P[g][i]++;
                }
            }
    for(i=1; i<= lengthOfSequences; i++)
        for(base=a; base∈ (a,t,c,g); next base) /* for all different bases */
            P[base][i]=(P[base][i]+pseduoCount)/lengthOfSequences;
    return Q;
} /* function ObtainBackgroundProfile */

CalculateIC(S,P,Q,z){
    /* let P be alightment matrix of motif set */
    for(i=1; i<= lengthOfSequences; i++)
        for(base=a; base∈ (a,t,c,g); next base) /* for all different bases */
            P[base][i]= P[base][i]*(numOfSequences-1)-pseudoCount;
    /* obtain alignment matrix including S[z] start at j */
    j= S[m].get_startIndex();
    for(i= j; i<= j+L; i++)
        if(S[z].getBase(i)=='A') P[a][i-j+1]++;
        if(S[z].getBase(i)=='T') P[t][i-j+1]++;
        if(S[z].getBase(i)=='C') P[c][i-j+1]++;
        if(S[z].getBase(i)=='G') P[g][i-j+1]++;
    for(i= 1; i<= L; i++)
        for(base=a; base∈ (a,t,c,g); next base) /* for all different bases */
            IC=IC + (P[base][i])log2((P[base][i]/numOfSequence)/Q[base]);
    return IC;
} /* end function CalculateIC */

```

Gibbs Motif Sampling

```

GibbsMotifSampling(S,numOfModel,width,expectedNumOfMotifs){
/* A set of sequences S is stored in an object array that has a two-dimensional string array and
sets of motif is in an object array MOTIF[ ]. width and expectedNumOfMotifs are stored in
array width[ ] and expectedNumOfMotifs[ ]. To avoid overlapping, sequence S has flag that
indicates whether it is included in motif or not by passing start index and its motif width. */
/* Step 1: initially e, numbers of set of motifs are located randomly */
for(i= 1; i<= numOfModels; i++)
    for(n= 1; n<= expectedNumOfMotifs[i]; n++)
        AssignRandomSites(S);
/* Step 2: [Motif Sampling] Estimation Step: obtain probability distributions */
/* define an initial sampled site x */
seqIndexX = Random_number_generator(numOfSequences);
startIndexX = Random_number_generator(lengthOfSequence);
do{ /* until converges */
    /* store current value for next convergence test */
    lastSeqX =seqIndexX; lastStartX=startIndexX;
    lastSeqZ =seqIndexZ; lastStartZ=startIndexZ;
    /* if site x is already in the alignment (selected as motif candidates), remove the motif
candidate that overlaps with x from the alignment.*/
    if(S[seqIndexX].get_flag(startIndexX,widthX) == true)
        S[seqIndexX].reset_flag(startIndexX,widthX);
    for(i= 1; i<= numOfModels; i++)
        P[i] = ObtainProfiles(S,0);
    Q = ObtainBackgroundProfile(S,P);
/* Step 3: [Motif Sampling] Maximization Step: draw a motif model */
/* initialize a posterior probability and the total number of possible sites */
    for(i= 1; i<= numOfModels; i++){
        for(j= 1; j<= numOfSequences; j++)
            totalNumOfPossibleSites=lengthOfSequence[j]-width[i]+1;
            if(max< totalNumOfPossibleSites)
                max=totalNumOfPossibleSites;
        totalNumOfPossibleSites[i]=max;
        if(posterirProb[i] ==0)
            for(i= 1; i<= numOfModels; i++)
                posterirProb[i] = expectedNumOfMotifs[i]
                    /totalNumOfPossibleSites[i];

        for(i= 1; i<= numOfModels; i++)
            L[i]=CalculateLikelihood(S,P,seqIndexX,startIndexX,totalNumOfPossibleSites,
                posterirProb);
        --- draw a model i proportional to likelihood calculated above ---
        /* update a posterior probability for given selected model i */
        posterirProb[i] = (numOfMotifs[i]+ pseudocount)/totalNumOfPossibleSites +
            sumOfPseudocount)
/* Step 4: [Column Sampling] Estimation Step: select one of motif candidates in model i */
seqIndexY=Random_number_generator(numOfMotifs[i]);
siteY=MOTIF[i][seqIndexY];
/* remove site y temporary from alignment */
numOfMotifs[i] ---;
MOTIF[i][numOfMotifs[i]].remove_site(seqIndexY,startIndexY);

```

```

/* Step 5: [Column Sampling] Maximization Step: select a highest information content site
from the largest width site that belongs to background. */
    for(j= 1; j<= numOfMotifs[0]; j++)
        if(max < MOTIF[0][j].get_width())
            max = MOTIF[0][j].get_width();
            z=j; /* select largest width segment z */
seqIndexZ=MOTIF[0][z].get_index();
startIndexZ=MOTIF[0][z].get_startIndex();
    for(n = 0; n<= widthZ; n++)
        IC=CalculateInformationContent(S,P,widthZ,seqIndexZ,startIndexZ+n);
        if(max < IC)
            max =IC;
            maxStartIndexZ= startIndexZ+n;
/* Step 6: merge two samplings; if two samplings selected the same region, replace site x
with site y, otherwise, restore y that previously removed from alignment and use site z as
next site x */
    if(seqIndexX== seqIndexZ)
        if(startIndexX==maxStartIndexZ)
            startIndexX = startIndexY;
            seqIndexX== seqIndexY;
        else
            startIndexX = startIndexZ; seqIndexX== seqIndexZ;
            numOfMotifs[i]++;
            MOTIF[i][numOfMotifs[i]].set_site(seqIndexY,startIndexY);
}while(seqIndexX!= lastSeqX || startIndexX!= lastStartX || seqIndexZ!= lastSeqZ ||
startIndexZ!= lastStartZ);
return MOTIF;
} /* function GibbsMotifSampling */

AssignRandomSites(S){
m = Random_number_generator(numOfSequences);
startIndex = Random_number_generator(lengthOfSequence);
while(S[m].get_flag(startIndex,width[i]) == true)
    m = Random_number_generator(numOfSequences);
    startIndex = Random_number_generator(lengthOfSequence);
S[m].set_flag(startIndex,width[i]);
S[m].set_MotifModelId(i);
numOfMotifs[i]++;
MOTIF[i][numOfMotifs[i]].set_site(m,startIndex);
}/* AssignRandomSites */

CalculateLikelihood(S,P,L,m,startIndex,totalNumOfPossibleSites,posteriorProb){
/* let P be alignment matrix of motif set */
for(i=1; i<= lengthOfSequences; i++)
    for(base=a; base ∈ (a,t,c,g); next base) /* for all different bases */
        P[base][i]= P[base][i]*(numOfSequences-1)-pseudoCount;
/* obtain alignment matrix with site x start */
for(i= startIndex(); i<= startIndex +L; i++)
    if(S[m].getBase(i)=='A') P[a][i-startIndex +1]++;
    if(S[m].getBase(i)=='T') P[t][i-startIndex +1]++;
    if(S[m].getBase(i)=='C') P[c][i-startIndex +1]++;

```

```

    if(S[m].getBase(i)=='G') P[g][i-startIndex+1]++;
for(i= startIndex; i<= L; i++)
    likelihood=likelihood*(posteriorProb[m]/(1-posteriorProb[m]))*((P[S[m].getBase(i)][i]/m)/Q
    [S[m].getBase(i)]);
return likelihood;
} /* end function CalculateLikelihood */

CalculateInformationContent(S,P,L,seqIndexZ,startIndexZ){
/* let P be alignment matrix of motif set */
for(i=1; i<= lengthOfSequences; i++)
    for(base=a; base∈ (a,t,c,g); next base) /* for all different bases */
        P[base][i]= P[base][i]*(numOfSequences-1)-pseudoCount;
/* obtain alignment matrix with site z */
for(i= S[seqIndexZ].get_startIndex(); i<= startIndexZ +L; i++)
    if(S[seqIndexZ].getBase(i)=='A') P[a][i-startIndexZ+1]++;
    if(S[seqIndexZ].getBase(i)=='T') P[t][i-startIndexZ+1]++;
    if(S[seqIndexZ].getBase(i)=='C') P[c][i-startIndexZ+1]++;
    if(S[seqIndexZ].getBase(i)=='G') P[g][i-startIndexZ+1]++;
for(i= 1; i<= L; i++)
    for(base=a; base∈ (a,t,c,g); next base) /* for all different bases */
        IC=IC + (P[base][i])log2((P[base][i]/m)/Q[base]);
} /* end function CalculateInformationContent */

```

VITA



Megumi Igarashi

Candidate for the Degree of

Master of Science

Thesis: A SURVEY OF CLUSTERING AND MOTIF FINDING FOR
MICROARRAY TECHNOLOGIES

Major Field: Computer Science

Biographical:

Personal Data: Born in Tochigi, Japan, On January 23, 1973, daughter of Takao Igarashi and Yoko Igarashi.

Education: Graduated from Omiya Koryo High School, Saitama, Japan in 1991; received Associates of Science from Miles Community College, Montana in May 1998; received Bachelor of Science from Oklahoma State University, Oklahoma in May 2002. Completed the requirements for the Master of Science degree at the Computer Science Department at Oklahoma State University in July 2004.

Experience: Employed as a desktop publisher at Kurashiki Printing Company, Tokyo Japan April 1993 through December 1996 and May 1998 through January 1999.