2005

Optimizing parameters in fuzzy k-means for clustering microarray data.

Wei Yang
University of Windsor

Optimizing Parameters in Fuzzy k-Means for Clustering Microarray Data

By
Wei Yang

A Thesis
submitted to the Faculty of Graduate Studies and Research
through the School of Computer Sciences
in Partial Fulfillment of the Requirements for the
Degree of Master of Science at the University of Windsor

Windsor, Ontario, Canada
2005

Your file Votre référence
ISBN: 0-494-09840-6
Our file Notre référence
ISBN: 0-494-09840-6

Abstract

Rapid advances of microarray technologies are making it possible to analyze and manipulate large amounts of gene expression data. Clustering algorithms, such as hierarchical clustering, self-organizing maps, k-means clustering and fuzzy k-means clustering, have become important tools for expression analysis of microarray data. However, the need of prior knowledge of the number of clusters, k, and the fuzziness parameter, b, limits the usage of fuzzy clustering. Few approaches have been proposed for assigning best possible values for such parameters.

In this thesis, we use simulated annealing and fuzzy k-means clustering to determine the optimal parameters, namely the number of clusters, k, and the fuzziness parameter, b. To assess the performance of our method, we have used synthetic and real gene experiment data sets.

To improve our approach, two methods, searching with Tabu List and Shrinking the scope of randomization, are applied. Our results show that a *nearly-optimal* pair of k and b can be obtained without exploring the entire search space.

Acknowledgments

I take this opportunity to thank my two advisors: Dr. Luis Rueda and Dr. Alioune Ngom for their continuous encouragement and intensive support throughout my graduate studies. This work would not have been achieved without their support and advice. I am grateful to Dr. Dan Wu and Dr. Behnam Shahrrava for their appreciated suggestions. I want to express my gratitude to my wife and parents for their support and love over these years and thanks to my friend, YuanQuan Zhang, for his help.

Finally, I want to express my thanks to the institutioins which supported this research work: School of Computer Science, University of Windsor, NSERC, the Natural Sciences and Engineering Research Council of Canada, CFI, the Canadian Foundation for Innovation, and OIT, the Ontario Innovation Trust.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

1.1.1 DNA, RNA and cDNA

The genetic information of every organism is stored in the biopolymeric molecule known as Deoxyribonucleic acid (DNA) .

DNA is composed of a long string of nucleotides, each of which contains one of four based (A, G, C or T), a deoxyribose sugar and a phosphate group. Naturally occurring DNA molecules are double-stranded in which two DNA strands are held together by specific interactions between bases A and T and bases C and G, each base pairing occurs exclusively [29]. The structure of a DNA molecule is illustrated by a double helix (Fig. 1.1 left shows a segment of DNA double helix).

We can regard DNA as a book, or even a library, storing all the genetic information for synthesizing protein or RNA. In the course of synthesizing proteins based on the genetic information on DNA, RNA is a molecular intermediary.

RNA is formed by a single strand, while the DNA consists of two complementary strands attached to one another, forming a double helix. The four bases in the DNA nucleotides are adenine (A), guanine (G), thymidine (T) and cytosine (C). In RNA

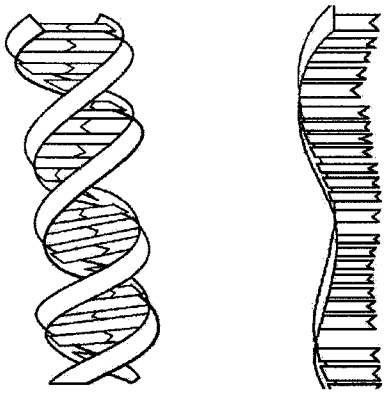

Figure 1.1: The segments of DNA and RNA.

thymidine is replaced by uracil (U). A RNA segment is shown in Fig. 1.1 right.

There are three major types of RNA:

1) mRNA, messenger-RNA, which transfers the genetic information about the aminoacid sequence from the DNA to the protein.

2) rRNA, ribosomal-RNA, which builds up the ribosome together with proteins.

3) tRNA, transfer-RNA, which transfer aminoacids to the ribosome for protein synthesis.

Complementary DNA, also named cDNA, is a single-stranded DNA synthesized from a mature mRNA template which only has exons[1] of genes. The property of lacking introns (A non-coding sequence of DNA) and only having exons makes cDNA widely used in gene function analysis.

1.1.2 Gene and Gene Expression

The genetic information on DNA is divided into different segments named genes. Gene is the basic unit of genetic function. Genes are continuous segments of genomic DNA constructed from four nucleotide blocks, namely A, G, T, and C. Every gene composes of three kinds of segment: regulatory segment, which contains information

[1]The region of a gene that contains the code for producing the gene's protein

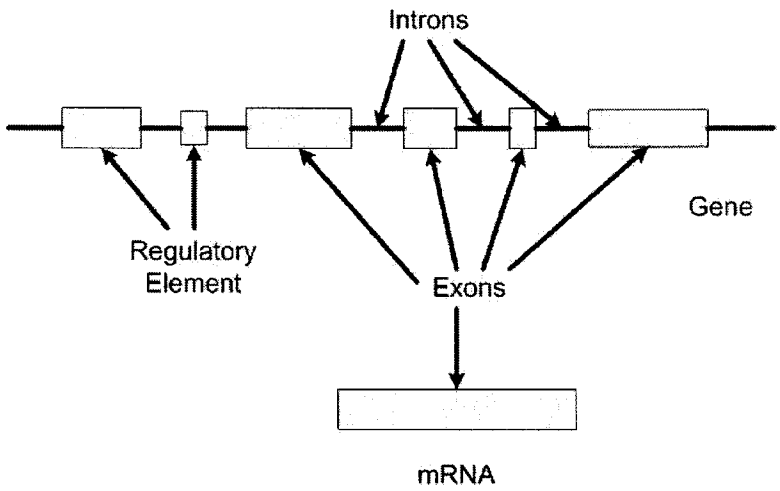

Figure 1.2: Gene with these three segments.

of initiation and regulating instructions, exon, which is the genetic coding region for mRNA, and intron, which is the non-coding part. Figure.1.2 shows a gene with these segments.

A gene is a recipe for synthesizing particular protein (a large 3-dimensional molecule playing structural and functional role as the basic building block for organisms) or in some case RNA.

The process by which a gene is converted into functional proteins is known as gene expression. It has been argued that a gene is expressed, if the gene is used in the process. In most cases, the same gene is involved in the synthesis of many different proteins.

In gene expression, a cell functions by using its genes to produce proteins. At any rate, the mechanism by which genes are expressed is the same for all cells and involves the transcription of a gene into mRNA before being translated into a protein. The production of mRNA is a reflection of the activity of a gene under certain conditions, and a lot of genetic information can be extracted by studying it.

For many years, study of gene expression in this manner had to be done by

individually looking at whether a specific gene is expressed under certain conditions or not. During the last half of the 20^{th} century, the analysis of the regulation and function of genes has largely driven step-by-step studies of individual genes and proteins.

1.2 What is a Microarray?

Since cells express their genes only when they are required for a cellular process under specific physiological conditions, how many genes are expressed under a given condition is an important clue to understand gene functions.

A microarray is such a device, a small analytical device that measures how genes are expressed in experiments, with such a speed and precision unprecedented in the history of biology.

A DNA microarray consists of an orderly arrangement of DNA fragments representing the genes under study. Each DNA fragment representing a gene is duplicated enough and assigned to a specific location on the microarray, usually a glass or nylon slide, and then "spotted"($\leq 1mm$) to that location. Through the use of highly accurate robotic spotters, over 30,000 spots can be placed on one slide, allowing molecular biologists the possibility to analyze virtually every gene present in a genome [5].

The main use of microarrays is that the spots are single stranded DNA fragments that are strongly attached to the slide, allowing cellular DNA, which is fluorescently labeled, to lay on top of the microarray. Cellular DNA in the sample will stick through a chemical process (called hybridization), to complementary DNA on the microarray at a pre-specified position. That is, gene-A from cellular DNA will stick to a spot on microarray composed of a gene-A fragment. By exposing the microarray to a laser with a specific wavelength, we can inspect the intensity of fluorescence. The more fluorescently labeled cellular DNA are hybridized with the DNA fragments,

the stronger the intensity is, which means the gene is more expressed under such specific conditions. Spots that have nothing hybridized to them will not be visible.

1.3 Two Types of Microarrays

There are two main types of commercial arrays widely used in gene expression experiments, oligonucleotide arrays (Affymetrix) and cDNA arrays. The basic idea of these arrays is similar. The difference is the way in which DNA fragments representing the genes are attached to the array.

1.3.1 Oligonucleotide Arrays

An oligonucleotide array (also called GeneChip) uses small DNA fragments with 25 base-pairs representing the genes to be spotted onto the array. These DNA fragments, called probes, are selected to have little cross-reactivity with other genes so that non-specific hybridization will be minimized [5].

Nevertheless, some non-specific hybridization will occur; to combat this, a second probe that is identical to the first, except for a mismatched base at its centre, is placed next to the first. This is called the Perfect Match/Mismatch (PM/MM) probe strategy. Any background hybridization with the MM probe is subtracted from the PM probe signal which results in perfect hybridization.

Fig.1.3 depicts an experiment using an oligonucleotide array involving the preparation of a cDNA sample for hybridization to the array. This sample is prepared with mRNA being extracted from cells. Because cDNA is more stable than mRNA in lab-environment, mRNA has to be reverse transcribed into cDNA. During the reverse-transcription step, a fluorescent dye is incorporated into the newly formed cDNA. Then the sample is cut into small fragments and bound to the probes on the array. By exposing the array to a laser with a specific wavelength, two sets of

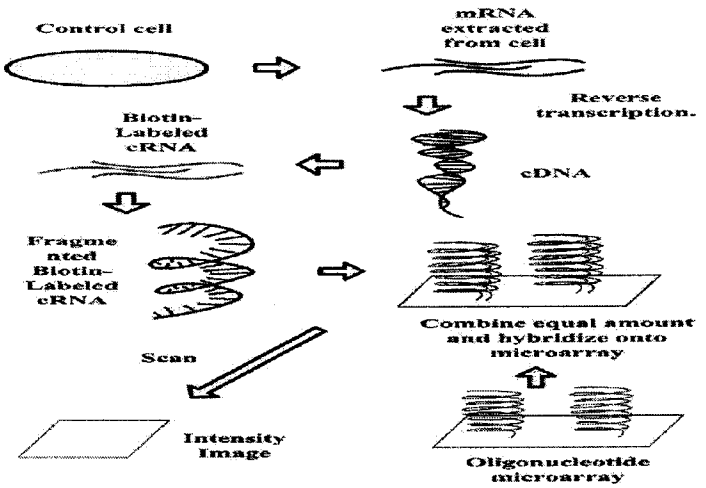

Figure 1.3: Procedure of an oligonucleotide Microarray.

intensities can be infered, one is from PM and other one is from MM. The difference between the two intensities is used to represent the level of gene expression. These data are what will be analyzed by some microarray analysis techniques.

Compared with cDNA array, there are two benefits [19]: first, oligonucleotide arrays are synthesized based on sequence information alone, i.e. without the need for intermediates such as clones, PCR products a cDNA; the other benifit is high confidence on the intensity signals, because of the use of probe redundancy and the use of mismatch control probes. However, oligonucleotide arrays are more complicated than cDNA arrays, and it is more difficult to analyze the data from the experiments using oligonucleotide arrays [8].

1.3.2 cDNA Microarrays

cDNA microarrays are similar to the oligonucleotide arrays. Instead of hybridizing short DNA fragments on the chips, each spot contains a cDNA clone from a known gene, usually with hundreds of bases. Nowadays, the cDNA clones for making the array can be generated from a commercially, available cDNA library ensuring a close

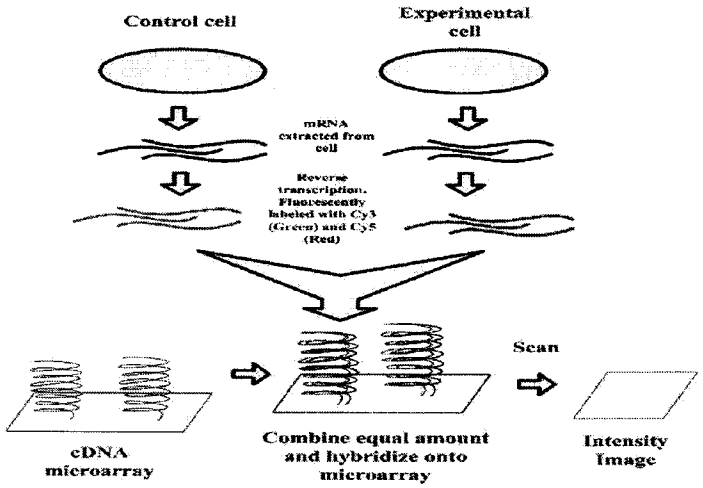

Figure 1.4: Procedure of a cDNA Microarray.

representation of the entire genome of an organism on the array [5].

Alternatively, polymerase chain reaction (PCA, which is a method enabling the isolation, cloning and amplification of genes) using specific primers can be used to duplicate specific genes from genomic DNA in library to generate the cDNA clones. A separate PCR must be performed for each gene, although these reactions can be done in parallel. These cDNA clones prepared can be mechanically spotted onto a glass slide.

Fig. 1.4 is an experiment using a cDNA microarray involving the preparation of two samples for hybridization to the array: a control sample and an experimental sample. These samples are prepared as before with mRNA being extracted from cells and reverse-transcribed into cDNA, with one exception: during the reverse transcription step a fluorescent dye is incorporated into the newly formed cDNA. Here, an ingenious trick is used and a different dye employed to label the different samples. For example, the control sample can be labeled with a green fluorescent dye called Cy3 and the experimental sample labeled with a red fluorescent dye called Cy5. Since the samples are labeled differently they can be combined and hybridized

to the microarray together (see Fig. 1.4). Then, two samples will competitively bind to the probes on the array and the sample containing more gene expression for a particular probe will win out. That is, if there is more of an mRNA transcript in the control sample than in the experimental sample (i.e., the gene is down-regulated in the experiment) then more Cy3 will bind to the probe on the array and the spot will fluoresce green. As opposed to this, if there is more experimental transcript, the reverse will happen and the spot will fluoresce red. When the two samples have the same amount: of transcript, the dyes will cancel each other out and the spot will fluoresce yellow.

The experiments using cDNA arrays are comparative hybridization experiments. We compare the hybridization level in each spot to the level of hybridization under control conditions, so our results from cDNA array experiments contain more pertinent information than that from oligonucleotide arrays.

Since the appearance of microarray technique, some applications acquired a brilliant achievement in gene research, human disease, drug discovery, and genetic screening and diagnostics.

1.4 Microarray Analysis

Microarray is now able to produce large amounts of data about many genes in a highly parallelized manner and allows scientists to study many, if not all, genes of an organism's at once. This high throughput achievement allows for the global study of changes in gene expression, giving us a complete cellular snapshot. Analysis on microarray is unique in the history of biology. No other technology has ever involved so much technology and combined expertise from so many different disciplines, including biology, chemistry, physics, engineering, mathematics and statistics, and computer science. Thus providing a quantitative and systematic view of a biological system.

Microarray differs from traditional research in a number of striking ways [32], one of which is the relationship between the amount of experimental time required and the amount of data obtained. Traditional experimental approaches based on gels and filter blots require a relatively large amount of experimental time to obtain a small volume of data, whereas microarray analysis offers vast quantities of data with relatively little experimental time. Microarrays purchased commercially provide a clear example, allowing a single researcher to generate large amounts of data in a few weeks.

How can we understand the role of the genes as a whole in biological functions based on so large amount of data? In other words, how can we define the role of each gene (or sequence of genes) in some biological function and subsequently understand how the genes function as a whole?

The action of discovering patterns of gene expression is closely related to correlating genes to specific biological functions and thereby understanding the role of genes in biological functions on a genomic scale. In order to properly comprehend and interpret expression data produced by microarray technology to this relationship, computational and data mining techniques are typically used to interpret the meaning of microarray data. We discuss below a few of these techniques.

1.4.1 Scatter Plots

Scatter Plots [29] are one of the most useful representations of gene expression. A Scatter plot is a graphical 2D representation of microarray data in which the signal intensities of two samples under different conditions are plotted along the x- and y-axes respectively, and the ratio values are plotted on the view.

One of the most useful applications of scatter plots is the visualization of gene expressions obtained from cells under control condition and experimental condition, such as the conditions with/without oxygen. Usually, the y-axis represents the

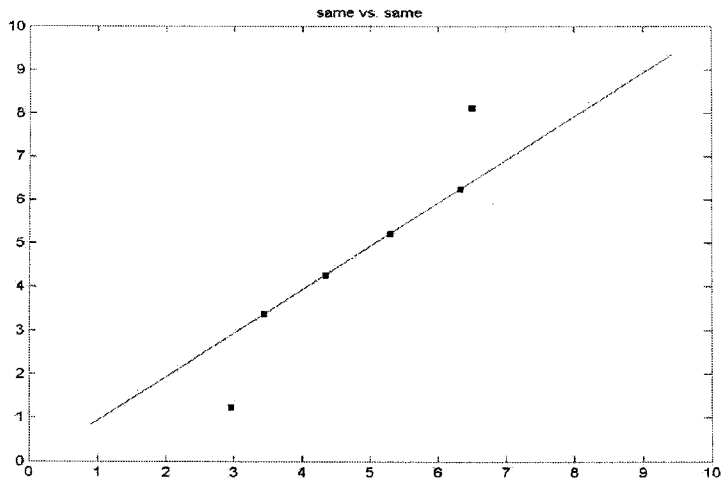

Figure 1.5: Visualization of gene expressions using scatter plots.

signal intensities of control samples and the x-axis represents the signal intensities of experimental samples. The line such that the ratio is equal to 1, $x = y$, is called the identity line. The genes with no changes of expression fall along the identity line. Genes expressed at a high level or genes that are activated fall above the line. Genes expressed at a low level or that are repressed fall below the line. Figure. 1.5 shows that two genes are expressed/repressed respectively as well as four genes are not. The scatter plot data reveals a significant scattering above and below the identity line and indicates the differences in gene expression levels in two samples.

Another application of scatter plots is precision assessment in ratio calculations. The precision of ratio measurement can be assessed by using two identical samples. The ratio of signal intensities should yield 1.0 and all genes should fall along the identity line. Figure. 1.6 assess the precision by same vs. same comparison. The concordance of the data along the identity line indicates the ratios are close to 1.

Usually time is involved as the third dimension. This 3D scatter plot represents the variance of genes expression in a period. The analysis of gene expression is more precise, when using a 3D scatter plot.

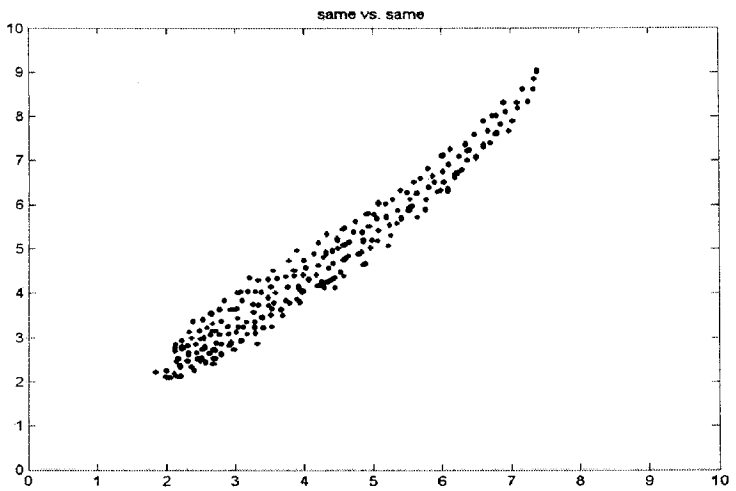

Figure 1.6: Same vs. same comparison for precision assessment using scatter plot.

1.4.2 Principal Component Analysis

Scatter plots visualize a comparison of two variables in 2D space, and a 3D scatter plot could be used for the comparison of three variables. In most microarray experiments, multivariate data are collected. The comparison of hundreds of variables means a scatter plot with hundreds of dimensions is needed. That is an impossible task to display multi-dimensional scatter plots. Alternately, by using a technique called dimensionality reduction, these hundreds of variables can be displayed into a 2D/3D plot.

One of the most useful methods of dimensionality reduction is principal component analysis (PCA) [35]. Principal component analysis transfers the relationships among these variables that exist in high-dimension space into a low-dimension space.

Principal component analysis is also used in microarray analysis to identify groups of genes in lower dimension of space. Principal component analysis is also used in reducing the noise in data.

Principal components analysis (PCA) is often used to find a basis set which is determined by the dataset itself. The principal components are orthogonal and

projections of the data onto them are linearly uncorrelated. Independent components analysis (ICA) [15] aims at a loftier goal: it seeks a linear transformation (an unmixing matrix) to coordinates in which the data are maximally statistically independent, not merely uncorrelated. Viewed from another perspective, ICA is a method of separating independent variables, which have been linearly mixed to produce the data.

1.4.3 Cluster Analysis

Another multivariate analysis method, developed in 1930's, is known as cluster analysis, which is to group the samples based on a similarity to one another. Clustering places similar samples into the same cluster, so that similar samples are close to each other and dissimilar samples are far away. In 1990's, clustering was first introduced to microarray analysis and have been used widely till today.

There are two categories of classification: *supervised classification* and *unsupervised classification* [29]. Supervised classification uses a trainer resulted from a subset of sample data to classify the rest of the data. Unsupervised clustering tries to discover the natural groups inside a data set without any input from a trainer. In this thesis, we focus on the methods of unsupervised clustering.

Many clustering methods have been used widely in microarray analysis. In this section, Hierarchical clustering algorithms, Self-Organizing Map,s and clustering affinity search techniques are introduced briefly. In the following chapters, k-means/fuzzy k-means clustering algorithm and simulated annealing are introduced in detail.

Hierarchical Clustering

We consider the clustering problem as a sequence of partitions with n samples into k clusters based on the similarity matrix. The basic idea of hierarchical clustering

algorithms [9], [34], is to build a tree as a sequence of partitions, by which the n samples are grouped into one cluster.

Depending on the way the tree is constructed, there are two kinds of hierarchical clustering algorithm: *agglomerative* and *divisive hierarchical clustering.*

In agglomerative hierarchical clustering, first, we suppose that there are n clusters initially using each sample as a cluster, and we calculate the similarity matrix, which is an n-by-n matrix usually containing the distance of each pair of clusters. Then, we merge a pair of cluster with most similarity into one cluster. A new similarity matrix of these clusters is calculated and the two clusters with most similarity are merged into one cluster again, and so on, until all clusters are grouped into one cluster. This algorithm is called agglomerative hierarchical clustering algorithm (or usually just called hierarchical clustering algorithm), because it builds a tree by combining similar clusters. This is a bottom-up approach.

Now, the problem is how to measure the similarity between two clusters. The term *distance* is usually used as the measure of similarity. The longer the distance is, the more dissimilar the two clusters are.

There are some variants of hierarchical clustering algorithms with different distances, including single-linkage clustering, complete-linkage clustering, average-linkage clustering, and centroid-linkage clustering [27].

Single-linkage clustering algorithms. The distance between two clusters is represented by the *minimum* distance among the members of two clusters. This is also called *nearest-neighbor* clustering. The distance is defined as follows:

$$d_{\min}(D_i, D_j) = \min_{x \in D_i, y \in D_j} \|x - y\|, \tag{1.1}$$

where D_i and D_j are the i^{th} and i^{th} clusters, x and y are samples in clusters D_i and D_j, $\|\cdot\|$ defines the distance function.

The problem is that a pair of clusters separated widely may be assigned as the

pair with best similarity, if there is a minimal distance existing among these two clusters.

Complete-linkage clustering algorithms. This algorithm uses *maximum* distance among the member of two clusters, and is also called *furthest-neighbor* algorithm. The distance is computed as follows:

$$d_{\max}(D_i, D_j) = \max_{x \in D_i, y \in D_j} \|x - y\|, \tag{1.2}$$

where D_i and D_j are the i^{th} and j^{th} clusters, x and y are samples in clusters D_i and D_j, $\|\cdot\|$ is distance computation.

This algorithm still has the drawback that we may miss the pair of clusters with the most similarity, if only a pair of samples that is very far away from each other exists in the clusters.

Average-linkage clustering algorithms. The algorithm applies the average distance among the members of two clusters. This algorithm is more reasonable than the two algorithms based on the weakness of the two algorithms we mentioned above. The formula of the distance is given as:

$$d_{avg}(D_i, D_j) = \frac{1}{n_i n_j} \sum_{x \in D_i} \sum_{y \in D_j} \|x - y\|, \tag{1.3}$$

where D_i and D_j are the i^{th} and j^{th} clusters, n_i and n_j are the numbers of samples that belong to the i^{th} and j^{th} clusters, and x and y are samples in clusters D_i and D_j.

Centroid-linkage clustering algorithms. In this algorithm, the distance is represented by using the distance between the means of two clusters. The distance between two clusters is defined as follows:

$$d_c(D_i, D_j) = d(\overline{x}, \overline{y}), \tag{1.4}$$

where $\overline{x}$ and $\overline{y}$ are the mean of clusters D_i and D_j.

In divisive hierarchical clustering algorithm, first, we suppose that there is a single cluster initially containing all n samples. We then split the cluster into 2 clusters by any of a large number of non-hierarchical clustering algorithm. Subsequently, the process is recursively repeated on these sub-cluster until each one has only one sample [8].

For its simplicity, hierarchical clustering becomes one of the most widely used algorithms on Microarray analysis. In [9], a hierarchical clustering algorithm was applied for the analysis on yeast microarray dataset. In [7], [13] and [26], the same algorithm was applied to analyze the microarray datasets on the molecular classification of cancers and biological modeling. D. Eppstein [11] developed data structures to obtain a faster hierarchical clustering algorithm.

A problem that typically arises is how to use a tree that results from Hierarchical clustering algorithm, and how to determine if a sub-tree is a cluster instead of a part of a bigger cluster. We need additional algorithms to interpret the tree into a form that can be understood by biologists.

In [16], an optimal leaf ordering algorithm was introduced. This algorithm makes the optimal leaf ordering maximize the sum of the similarity of adjacent elements in the ordering. This algorithm could also help users identify and interpret the data.

Self-Organizing Maps

A self-organizing map (SOM) [17] is a clustering algorithm very similar to the k-means clustering algorithm. However, the SOM is a two-level clustering algorithm. First, SOM projects n high-dimensional data points onto a low-dimension map, instead of dividing the original data points into the k clusters directly, and then to group the units on this map into k clusters. Usually, the map is a two-dimensional grid with M units, and each unit represents a vector, $m_i = (m_{i1}, m_{i2} \ldots m_{id})$ where

d is the dimension of original data.

With this 2-level approach, there are two benefits [31]: decreasing the cost of computation and noise reduction.

The training of the SOM is iterative. At each step, a sample is randomly chosen. The distance between vectors represented by the unit is calculated and the best-matching unit m_{best} is the nearest one to the sample x. The best-matching unit is computed as follows:

$$\|x - m_{best}\| = \min_i \{\|x - m_i\|\}, \tag{1.5}$$

where m_i is i^{th} unit and x is a sample.

Then the neighbor units of the best-matching unit are moved towards the sample x based on the formula $m_{i+1}(t+1) = m_i(t) + \alpha(t)h_{bi}(x - m_i)$, in which $\alpha(t)$ is the learning rate decreasing with time, $h_{bi}(t)$ is a neighborhood function, denoting how much the neighbor i^{th} units is affected by the best-matching unit. The algorithm continues until reaching the desired numbers of iteration or the learning rate $\alpha(t)$ is 0.

In [30], Tamayo et al. employed a self-organizing maps clustering algorithm to interpret the patterns of the yeast microarray data set.

The SOM not only is a good method to cluster data sets, also is a good tool to show the cluster structure of microarray data. Lee [18] presented a method to reveal the gene expression profiles for more than thousands of genes.

Clustering Affinity Search Technique

In this section, we discuss an algorithm, which does not require the prior knowledge of k, named *clustering affinity search technique* (CAST). In 1999, this algorithm was presented by Ben-Dor et al. first in Journal of Computational Biology [3]. This algorithm groups data points into clusters based on the average similarity (also called affinity), between the current cluster (*open cluster*) and un-clustered data points.

CAST algorithm alternates between adding and removing a data point into/from a cluster.

S1. *Initialization Step*

Initialize an affinity threshold t.

S2. *Generation Step*

A blank cluster C_{new} is opened.

Repeat until no changes occur:

Add a unsigned data point x into C_{new}, if their similarity greater than t.

Remove a data point y from C_{new}, if their similarity less than t.

Close C_{new}.

S3. *Judgement step*

If no unsigned data point, stop.

Otherwise, go to Step 2.

Many possible similarity measures can be used in CAST, such as typically used Euclidean distance.

So, at the end of the algorithm, there is always a 'cleaning' step to make the affinity between data points and the clusters as maximum as possible by adjusting the data points among the clusters.

In CAST, we do not need the prior knowledge of k, but there is a serious drawback. That is, we have to initialize a control parameter, the threshold t. This parameter affects the shape of the clustering structure. Bellaachia et al. [2] proposed an enhanced CAST algorithm, in which, there is a dynamic threshold t, instead of a fixed t, which is computed at the beginning of the generation of the new cluster.

1.5 Conclusion

In this chapter, we introduced some biological knowledge about microarray and microarray analysis methods, which are typically used to analyze the results from microarray experiments. In Chapter 2, two clustering algorithms, k-means and fuzzy k-means, will be introduced in detail. In Chapter 3, an optimization algorithm, simulated annealing, is introduced. The related works will be reviewed in Chapter 4. Our approach to optimize the parameters of fuzzy k-means will be proposed in Chapter 5. To prove the efficiency of our approach, some experiments are showed in Chapter 6.

Chapter 2

Clustering Algorithms for Microarray Data

Clustering algorithms are a generic tool for pattern recognition, grouping the data-points into groups, also called clusters. The data-points in same groups are more similar to each other than those in different groups.

2.1 k-Means/Fuzzy k-Means Clustering Algorithm

Clustering algorithms are categorized as hierarchical and non-hierarchical. In the last section, the details of hierarchical clustering algorithms were already discussed. In non-hierarchical clustering algorithms, also called partitional clustering, each gene is assigned to a cluster based on the similarity between the gene expression and the expression of genes in the cluster.

In this thesis, we focus on a partitional clustering algorithms, namely fuzzy k-means clustering algorithm which is gaining wide use in microarray analysis. For simplicity of understanding of this algorithm, first, we introduce another non-hierarchical clustering algorithm, k-means, which is very similar to the fuzzy k-means clustering algorithm.

2.1.1 k-Means

k-Mean [23] is a clustering algorithm based on a mixture model [25].

In a mixture model, the dataset is composed of a number of clusters, and there is a distribution associated with each cluster. k-Means clustering algorithm splits each data into these clusters. In Bayesian decision theory, the probability $P(\omega_i \mid x_j)$, also named *membership*, of assigning the cluster ω_i after observing the j^{th} sample, x_j, is defined as:

$$P(\omega_i \mid x_j) = \frac{p(x_j \mid \omega_i)P(\omega_i)}{p(x_j)}, \tag{2.1}$$

where $p(x_j \mid \omega_i)$ can be seen as the distribution of the sample x_j, which belongs to cluster ω_i, $P(\omega_i)$ is the prior probability of ω_i, $p(x_j) = \sum_{i=1}^{k} p(x_j \mid \omega_i)P(\omega_i)$ (which can be viewed as a scale factor that guarantees $P(\omega_i \mid x_j)$ sum to one) is the probability of sample x_j, and k is the number of clusters.

Here, we regard the distribution $p(x_j \mid \omega_i)$ as the normal distribution, which is also called Gaussian distribution. In this case, the probability $P(\omega_i \mid x_j)$ can be expressed as:

$$P(\omega_i \mid x_j) = \frac{\mid \Sigma_i \mid^{-1/2} \exp[-\frac{1}{2}(x_j - \mu_i)\Sigma_i^{-1}(x_j - \mu_i)]P(\omega_i)}{\sum_{l=1}^{k} \mid \Sigma_l \mid^{-1/2} \exp[-\frac{1}{2}(x_j - \mu_l)\Sigma_l^{-1}(x_j - \mu_l)]P(\omega_l)}, \tag{2.2}$$

where μ_i is the centroid of the i^{th} cluster, Σ_i is the covariance matrix of the i^{th} cluster, and x_j is the j^{th} sample, also called the j^{th} *data point*.

From Equation 2.2, it is clear that the probability $P(\omega_i \mid x_j)$ is large when the squared *Mahalanobis distance* [21] $(x_j - \mu_i)\Sigma_i^{-1}(x_j - \mu_i)$ is small. That is, the sample x_j belongs to the cluster Σ_i whose centroid μ_i is closest to x_j and approximate $P(\omega_i \mid x_j)$ as

$$P(\omega_i \mid x_j) = \begin{cases} 1, & \text{if } x_j = \arg\min_i (x_j - \mu_i)\Sigma_i^{-1}(x_j - \mu_i); \\ 0, & \text{otherwise,} \end{cases} \tag{2.3}$$

where

$$\mu_i = \frac{\sum_{j=1}^{n} P(\omega_i \mid x_j) x_j}{\sum_{j=1}^{n} P(\omega_i \mid x_j)}, \tag{2.4}$$

n is the number of samples in data set, x_j is the j^{th} data point, and $\Sigma = I$, which, for simplicity, is Euclidean distance.

The k-means clustering algorithm works as follows:

1. the algorithm initializes k (the number of clusters), and the centroid μ_i of each cluster;

2. the data points are placed into the nearest cluster to get its new membership based on Equation (2.3).

3. by using the membership, the centroid of each cluster is updated using Equation 2.4.

4. If no changes in μ_i, stop. Otherwise, go to Step 2.

2.1.2 Fuzzy k-Means

It is well known that a given gene is involved in the synthesis of many proteins, either directly or indirectly (via a network of gene expressions). Thus, the clusters of genes are not necessarily distinct and sharp.

A given cluster may represent a molecular function shared by its member, and a given gene can belong to this cluster with a certain degree of membership. In general, a gene may exhibit many functions. To understand the functions of genes better, we need another method that can assign each data point into several clusters. From the discussion above, k-means assigns each data point to be in *exactly one cluster*, as implied by Equation (2.3).

We can relax this condition and assume each data point has a fuzzy membership to a cluster. This means that each data point can be assigned into many clusters.

Fuzzy k-means is based on the first-order differentiation aiming to find a clus-

tering structure that minimizes the sum of weighted within-cluster distance:

$$\mathbf{J_F} = \sum_{j=1}^{n}\sum_{i=1}^{k} P(\omega_i \mid x_j)^b \|x_j - \mu_i\|^2 \tag{2.5}$$

where k is the number of clusters, n is the number of data points, $P(\omega_i \mid x_j)$ is the probability of the j^{th} data point to i^{th} cluster, μ_i is the centroid of the i^{th} cluster, x_j is j^{th} point in data set, and b is a fuzzy control parameter whose value is usually set to 1.25.

We have the following necessary conditions:

$$\partial \mathbf{J_F}/\partial \mu_i = 0 \tag{2.6}$$

$$\partial \mathbf{J}/\partial P = 0, \tag{2.7}$$

which lead to the solutions

$$\mu_i = \frac{\sum_{j=1}^{n}[P(\omega_i \mid x_j)]^b x_j}{\sum_{j=1}^{n}[P(\omega_i \mid x_j)]^b} \tag{2.8}$$

and

$$P(\omega_i \mid x_j) = \frac{(1/d_{ij})^{1/(b-1)}}{\sum_{r=1}^{k}(1/d_{rj})^{1/(b-1)}}, \tag{2.9}$$

in which $d_{ij} = \|x_j - \mu_i\|^2$. Since we used μ_i to compute $P(\omega_i \mid x_j)$, and vice versa, (2.6) and (2.7) have no direct solution to solve this system of algorithms. Fuzzy k-means is applied, which works as follows:

1. the algorithm initializes k (the number of clusters), and the centroid μ_i of each cluster;

2. the data points are moved into the nearest cluster to get his new membership based on Equation (2.9);

3. by using the membership, the centroid μ_i of each cluster is updated using

(2.8);

4. If no changes in μ_i, stop. Otherwise, go to step 2.

2.1.3 Limitations of k-Means

Although k-means and fuzzy k-means have been widely used for their simplicity, their critical weakness is their need for the correct value of k (the number of clusters). That is, which value of k yields the best result, where k is not know a priori.

One of the ways to get the best k known in entire search space is to run k-means clustering algorithm on all of possible values of k, evaluate the result (for each k) and then select the value k which gives the best result. The evaluation of results is done through validity indices that will be discussed in later section.

Another way is to determine the number of clusters by the knowledge of biology. Unfortunately, this is not always possible since we only understand few of knowledge about genes.

For fuzzy k-means clustering algorithm, the choice of the fuzziness parameter b is also a problem.

2.2 Similarity Measures

As discussed earlier, the main idea of k-means and fuzzy k-means is to distribute the data points into clusters by measuring their similarity to each other. Thus in fuzzy k-means, we need measurements for the similarity of the data points. Many distance functions are used to measure similarity between two data points, we briefly discuss the most well-known distance functions below.

2.2.1 Mahalanobis Distance

Mahalanobis [21] distance was introduced first in 1936. The general multivariate normal probability density formula is written as

$$p(x) = \frac{1}{(2\pi)^{d/2} \mid \Sigma \mid^{1/2}} \exp[-\frac{1}{2}(x-\mu)^t \Sigma^{-1}(x-\mu)]. \tag{2.10}$$

Samples drawn from a normal population tend to fall in a single cluster, whose center is determined by μ and the shape of the cluster is determined by Σ, where x is the data point, μ is the centroid of cluster and Σ is the covariance matrix. The formula is written as

$$\Sigma = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^t. \tag{2.11}$$

The quantity $r^2 = (x-\mu)^t \Sigma^{-1}(x-\mu)$ is usually called the squared Mahalanobis distance from x to μ. For the definition above, the Mahalanobis distance works well for correlated variables with different scales, and describes the shape of clusters more precisely, but increasing the computational complexity.

2.2.2 Euclidean Distance

Euclidean distance can be seen as a special case of Mahalanobis distance, when the covariance matrix $\Sigma = I$ (i.e. the Identity matrix).

The Euclidean distance is supposed works best for un-correlated random variables and with same variance. Compared with the Mahalanobis distance, although the Euclidean distance gives a less precise description of the similarity of the data points, it is still used widely for its simplicity.

2.2.3 Minkowski Distance and City Block Distance

Another general class of distances can also be used in clustering algorithms, the Minkowski distance. The Minkowski distance between the d-dimensional data points is written as:

$$dist_{ij} = (\sum_{k=1}^{d} \mid x_{ik} - y_{ik} \mid^{p})^{1/p}, p > 0. \tag{2.12}$$

The City Block distance is a special case of Minkowski distance when $p = 1$. The City Block distance measure the shortest path between two points, each segment of which is parallel to a coordinate axis. It is also called Manhattan distance.

In fact, the Euclidean distance is also a special case of the Minkowski distance when $p = 2$.

2.2.4 Pearson's Correlation Coefficient

Correlation is a strong technique for measuring the relationship between two random variables, for example, the mileage of a used car and its resell price. Pearson's correlation coefficient is a quantitative measure of the strength of a linear relationship between two variables.

Suppose we have two random variables X and Y with means $\overline{x}$ and $\overline{y}$. The Pearson's correlation coefficient is computed as

$$r = 1 - \frac{\sum_{i=1}^{d}(x_i - \overline{x})(y_i - \overline{y})}{[\sum_{i=1}^{d}(x_i - \overline{x})^2]^{1/2}[\sum_{i=1}^{d}(y_i - \overline{y})^2]^{1/2}}, \tag{2.13}$$

where d is the number of observations, x_i and y_i is i^{th} observation in variable X and Y.

The Pearson's correlation coefficient ranges from -1 to $+1$. A positive value is an evidence of tendency that this two variables are correlated; a negative value of Pearson's correlation coefficient represents the tendency that these two variables are negatively correlated. The closer the correlation coefficient is to $+/-1$, the

stronger the relationship is between these two variables. A value of zero indicates no correlation at all.

Fig.2.1 shows the linear relationship of a pair of variables. In Fig.2.1.a, the Pearson's correlation coefficient is -0.0584 and in Fig.2.1.b, the Pearson's correlation coefficient is $+0.9831$.

Pearson's correlation coefficient can also be applied to the similarity measure of time series microarray data. Here, we regard each gene as a variable, and the correlation coefficient between two genes are computed by Equation (2.13). If the correlation coefficient is close to $+1$, the patterns of gene expressions are similar; if the value is close to 0, the similarity of the patterns of gene expression is weak; and the patterns of gene expressions are reversed if the correlation coefficient is close to -1.

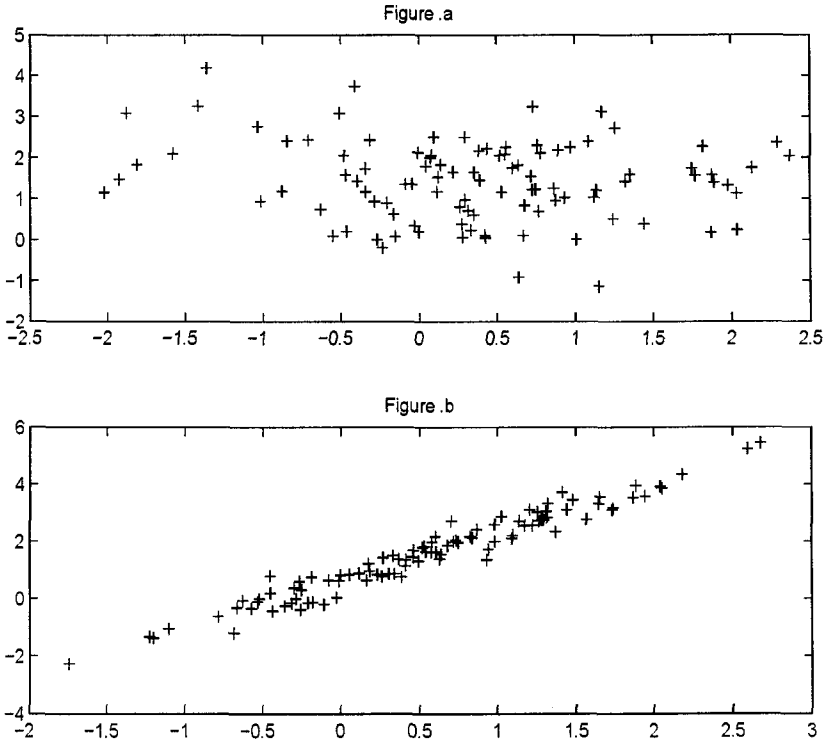

Figure 2.1: Two examples with different Pearson's correlation coefficient.

2.3 Clustering Validity Indices

Finding the optimal number of clusters is another important issue in clustering. For this purpose, validity indices are typically used to measure how good the result of clustering is. Broadly speaking, a clustering structure with minimal within-cluster distance and maximal between-cluster distance is the best one we seek. We discuss below the most well-known indices found in the literature [22].

The **XB Index** is a ratio of the compactness of the partition of the data to its separation. The expression for this index is written as follows:

$$XB = \frac{\sum_{i=1}^{n}\sum_{j=1}^{k} u_{ij}^2 \left\|x_i - \mu_j\right\|^2}{n \min_{m,n} \left\|\mu_m - \mu_n\right\|^2}, \tag{2.14}$$

where μ_j is the mean of the j^{th} cluster, u_{ij} is the degree of membership of the i^{th} element to the j^{th} cluster, and x_i is the i^{th} point in the data set. The smaller the value of XB, the better the clustering structure, and thus, we seek to *minimize* XB.

The **I Index** is defined as follows:

$$I(k) = \left(\frac{E_1 \times D_k}{k \times E_k}\right)^p, \tag{2.15}$$

where $D_k = \max_{i,j=1}^{k} \left\|\mu_i - \mu_j\right\|$, $E_k = \sum_{i=1}^{n}\sum_{j=1}^{k} u_{ij} \left\|x_i - \mu_j\right\|$, k is the number of clusters, and n is the number of data points. E_1 is a constant for a given data set, and p is used to control the contrast between the different cluster configurations.

The term $\frac{1}{k \times E_k}$ decreases with k, and D_k increases with k. Thus, these two factors compete and balance each other in order to obtain the optimal value of k. The value of k for which $I(k)$ is *maximized* is considered to be the correct number of clusters.

The **Calinski-Harabasz (CH) Index** for n data points and k clusters is given

by:

$$CH = \frac{[traceB/(k-1)]}{[traceW/(n-k)]}, \tag{2.16}$$

where $traceB = \sum_{j=1}^{k} n_j \|\mu_j - \mu\|^2$, n_j is the number of points in the j^{th} cluster, μ is the mean of the entire data set, μ_j is the mean of the j^{th} cluster, and $traceW = \sum_{j=1}^{k} \sum_{i=1}^{n_j} \|x_i - \mu_j\|^2$. In this index, the number of clusters is considered as an important factor. The larger the value of CH is, the better the partition is, and hence we seek for *maximizing CH.*

The **Davies-Bouldin (DB) Index** is the ratio of the sum of within-cluster distance to between-cluster separation, and is computed as follows:

$$DB = \frac{1}{k} \sum_{i=1}^{k} R_i, \tag{2.17}$$

where $R_i = \max_{j,j \neq i} \left\{ \frac{S_i + S_j}{d_{ij}} \right\}$, in which $S_i = \frac{1}{|C_i|} \sum_{x \in C_i} \|x - \mu_i\|$ and $d_{ij} = \|\mu_i - \mu_j\|$. Smaller values of DB represent better clustering, and the value that *minimizes DB* is the optimal number of clusters.

Although the above mentioned indices give good results, obtaining the optimal number of clusters is still an open problem. Clustering methods use a fixed parameter, k, as the number of clusters. Such parameter is determined by a trial-and-error procedure in order to obtain a value that yields the best clustering results. In particular, for large data sets, there is no evidence that the value of k obtained is optimal (unless one knows the correct number of clusters based on the nature of the data set).

2.4 Conclusion

In this chapter, we introduced k-means/fuzzy k-means clustering algorithms in detail and we summarized the weakness in k-means/fuzzy k-means. Some similarity

measures and validity indices, which are widely used in clustering algorithms are also presented.

Chapter 3

Simulated Annealing

In previous chapter, we mentioned a major limitation of k-means clustering. That is, different initializations yield different results which are not necessarily optimal.

In this chapter, we will discuss an algorithm for searching (near) optimal clustering results given a well-defined objective function. We use a well-known optimization method, simulated annealing, which has been widely applied to many optimization problems.

3.1 The Basic Idea

As the name indicates, the main idea of the algorithm is derived from the analogy of thermodynamics with metal cooling and annealing. The amazing fact is that nature is able to make metals stick at minimum energy state by slow cooling. Along the way towards minimum energy state, atoms can not only move to the position with lower energy, but also move to higher energy with some probability.

Boltzmann factor $P(T) \propto \exp(\frac{-\Delta E}{T})$ describes exactly the probability of moving to a new state in a thermal system contact at temperature T based on the difference of energy, ΔE . The probability of mobility of the atoms is decreased as the temperature T cools down. In 1953, Metripolis et al. [24] first introduced this principle

into numerical calculations.

To make a use of simulated annealing algorithm, there are three terms that have to be defined: *configuration*, *objective function* and *energy*. The aim of simulated annealing algorithm is to seek one configuration among all of possible configuration that makes an objective function best. Here, each configuration is a possible solution, which contains one or many variables. For example, finding the minimal value of a function $f(x, y, z)$ with three variables. A configuration is one of possible solutions with the values of x,y,z. In microarray analysis with simulated annealing, the configuration may be the membership matrix, the mean of the clusters, the number of clusters, k, or the fuzziness parameter, b.

The objective function is a function that measures the quality of the configurations. As an example, Validity indices for clustering algorithms can be used as objective function.

Another term, energy, was introduced to simulated annealing algorithm. Energy is the value of quantizing an objective function for measuring the quality of the configuration.

The simulated annealing algorithm changes the current configuration to a new one with the probability $P = \exp[\frac{-(E_{new}-E_{current})}{T}]$, usually called the Metropolis acceptance rule. E_{new} is the energy of the new configuration and $E_{current}$ is the energy of the current configuration, while T is one value of a sequence of temperatures, called annealing schedule. If $E_{new} < E_{current}$, the system moves to the new configuration. Otherwise, the system moves to the new configuration only with probability P. In other words, the system always moves downhill towards minimal energy while sometimes moving uphill with probability P. Based on the acceptance rule, T is very high enough at the beginning to make sure that most of the moves will be accepted. However, as T approaches 0, most of the uphill moves are rejected. The temperature T is updated to be lower, another new configuration will be generated

randomly, and its energy will be computed. The comparison will be done again until the temperature T is smaller than a *threshold*.

The simulated annealing algorithm works as follows:

S1. *Initialization Step*

Temperature T_0 is initialized.

A configuration is randomly chosen.

$E_{current}$ is calculated.

S2. *Generation Step*

A new configuration is randomly generated.

The energy E_{new} is calculated

S3. *Selection Step*

Accepted new configuration with the probability $P = \exp[\frac{-(E_{new}-E_{current})}{T}]$, in which E_{new} and $E_{current}$ are the energies of the new configuration and current configuration.

If the configuration is accepted, then $E_{current}$ is assigned to E_{new}.

S4. *Updating Step*

The temperature is updated.

If the temperature is above the threshold, then go to S2.

Otherwise, stop.

The computation time and quality of the result depend on the threshold. In the following section, we will discuss the annealing schedule T and how it relates to quality of solution and execution time.

3.2 Annealing Schedule

Based on the previous description about simulated annealing algorithm, determining a good annealing schedule is one of the very important issues for the efficiency of the simulated annealing algorithm. It is a sequence of the temperature with a

high value at the beginning that decreases slowly. High temperature values yield higher probability of acceptance, and the probability decreases as the temperature decreases. Here, the temperature is a parameter to control the probability of accepting a move. There are two kinds of annealing schedules, based on *fixed* decrement rules and *adaptive* decrement rules.

The fixed decrement rule is independent of the algorithm itself. The temperature is decreased proportionately to a constant over the course of the algorithm. There are several choices, such as the geometric cooling schedule ($T(t) = \alpha^t T_0$) and the logarithmic cooling schedule ($T(t) = \frac{T_0}{\log t}$), where T_0 is the initial temperature, t is the number of iterations, and α is a parameter defined by users.

As an example, we have plotted the geometric cooling schedule and the logarithmic cooling schedule, which are widely used in applications. Figure 3.1 shows the plot of the geometric cooling schedule, in which $T_0 = 10000$, $\alpha = 0.8$ and t is from 2 to 70 stepped up by 1. Figure 3.2 shows the plot of the logarithmic cooling schedule, in which $T_0 = 1$ and t is from 2 to 70 stepped up by 1.

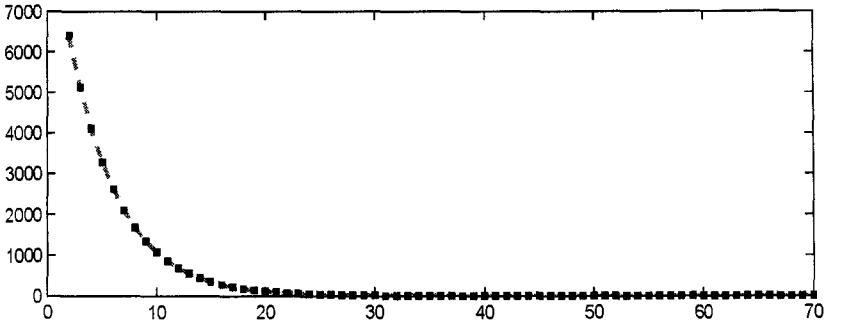

Figure 3.1: Geometric cooling schedule

The second kind of schedules is based on adaptive decrement rules varying dynamically the proportional scale of the current configuration and the temperature decrements over the course of the algorithm. Elmohamed et al. [10] introduced several adaptive annealing schedules.

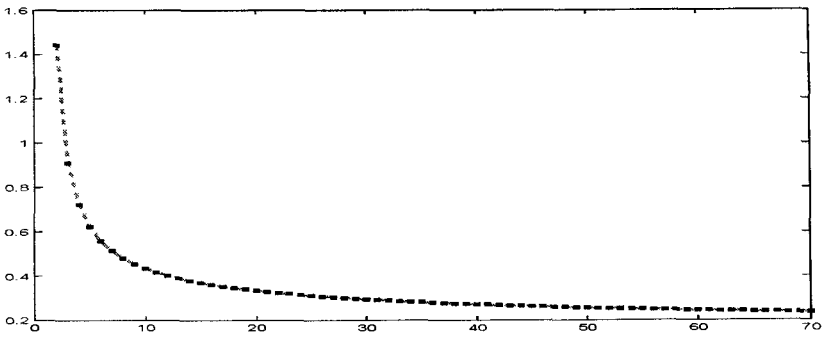

Figure 3.2: Logarithmic cooling schedule

For optimality criterion, there are, usually, several optimization criteria, such as the probability to be in the ground state, the final energy regarding the last configuration as the final result, and the best-so-far energy representing the lowest energy found in the solution path, introduced by K.H. Hoffmann [14].

3.3 Conclusion

In the first section, simulated annealing was briefly introduced, and the pseudo code was listed at the end of this section. In the second and third sections, the important issues, annealing schedule and optimality criterion, were also introduced.

Chapter 4

Related Work

Clustering approaches have been quite important tools for analyzing microarray data. Gasch et al. [12] used fuzzy k-means clustering to identify overlapping clusters of yeast genes by allowing genes to belong to more than one cluster, because of the co-regulation of genes. Using fuzzy k-means clustering, they identified some previously unrecognized gene clusters and uncovered correlations between the environmental conditions.

In recent years, fuzzy k-means has been widely used in many fields. As a result, the need to solve the inherent weaknesses, the prior knowledge on the number of clusters k and the fuzzy control parameter b, is arising.

Ray et al. [28] proposed a method to automatically determine the number of clusters using a simple objective function. The method obtains the best clustering by searching the entire searching space of k based on the objective value.

Due to the large size of microarray data, the need to cluster such data set into the 'exact number' of natural clusters becomes very important. Many approaches based on simulated annealing have been adopted in literature.

Lukashin et al. [20] proposed an algorithm to determine the optimal number of clusters by applying SA to cluster microarray data. In their approach, the membership matrix varies until the one minimizing the objective function, in this case, the

sum of within-cluster distance, is obtained. Then, the optimal number of clusters is the number of clusters obtained by SA.

Wang et al. [33] introduced a method to determine the number of clusters by simulated annealing. In each iteration step of SA, the new center of clusters is obtained by applying certain operations on the clusters, and using the XB index.

Another major problem in applying fuzzy k-means clustering is the choice of the fuzziness parameter b. D. Dembele and P. Kastner [6], shows that a fixed value of b is not appropriate for some data sets and optimal value of b vary widely from one data set to another. They also presented an empirical method to determine an adequate value for the fuzziness parameter b. In that paper, the optimal number of clusters is obtained by the algorithm CLICK (Cluster Identification via Connectivity Kernels). To assess the quality of the clusters, the *silhouette* measure is used.

We propose a search method, which aims to determine the optimal pair (k, b), where k is the number of clusters, and b is the fuzziness parameter, by combining simulated annealing and fuzzy k-means. Table 4.1 presents a comparison between the proposed approach and the existing methods. The column "μ" shows the method used to determine the means of the clusters. The column "P" shows the methods used to determine the probabilities that genes belong to the clusters. The column "k" shows the method used to determine the number of clusters in the corresponding approach, and the column "b" shows the method used to determine the fuzzy control parameter b. The column "*Validity Index*" shows the validity indices used to measure the quality of the clusters.

For example, the fifth row shows the method discussed in the approach proposed by Dembele et al., the means of the clusters and the probabilities that genes belong to a cluster are determined by fuzzy k-means, the number of clusters k is determined by CLICK, and the fuzzy parameter is determined by a *bruce-force* method on the entire search space. Silhouette is used to measure the quality of the clusters.

Approach	μ	P	k	b	Validity Index
Gasch et al. [12]	FCM	FCM	-	-	-
Ray et al. [28]	FCM	FCM	Bruce-force	-	intra/inter
Lukashin et al. [20]	SA	SA	SA	-	J
Wang et al. [33]	SA	SA	SA	-	XB
Dembele et al. [6]	FCM	FCM	CLICK	Bruce-force	Silhouette
Proposed	FCM	FCM	SA	SA	J, XB, CH, I, DB

Table 4.1: Comparison of the proposed method with the existing methods.

In this chapter, some different methods, which are used to determine the parameters in fuzzy k-means clustering algorithm, were listed. A comparison between our approach and these methods was given at the end of this chapter.

Chapter 5

Approach for Optimizing Parameters

As we mentioned earlier, in fuzzy k-means, there are limitations because of the need of the prior knowledge of parameters k and b. In this chapter, we discuss our approach on optimizing the parameter k and the pair of (k,b) respectively in fuzzy k-means.

5.1 Finding a Optimal Number of Clusters

Our approach to find the optimal number of clusters, k, was designed to achieve a nearly-optimal value of k by using simulated annealing. The simulated annealing algorithm selects a value k according to the *Metropolis acceptance rule*. This is repeated many times and the best value of k, so far, is returned as the solution we seek. We use geometric cooling as our annealing schedule. The energy value associated with a given value k (that is, the objective value that tells us how good the clustering result is, when k is used as the current number of clusters), denoted

by $E(k)$, is simply[1] $XB(k)$, $CH(k)$, $I(k)$ or $DB(k)$; where XB, CH, I and DB are validity indices mentioned earlier in this thesis.

To determine the energy for k, $E(k)$, we apply the fuzzy k-means clustering algorithm on the given data set (where k is the number of clusters) and then apply one given validity index, *e.g.* if we use the index CH to cluster, the value $CH(k)$ is then returned as $E(k)$.

The following pseudo code formalizes the procedure for finding a nearly optimal number of clusters by using the simulated annealing approach. The algorithm receives the dataset D, and a threshold, δ, as parameters.

S1. *Initialization Step*

Initialize the temperature $T \leftarrow T_0$.

Randomly choose a value of k_0

Run the fuzzy k_0-means clustering algorithm

Obtain the initial energy, $E(k_0)$.

$k_{current} \leftarrow k_0$

S2. *Generation Step*

Randomly choose k_{new}, and run fuzzy k_{new}-means.

Calculate the energy $E(k_{new})$.

S3. *Selection Step*

If $E(k_{new}) < E(k_{current})$

$k_{current} \leftarrow k_{new}$.

else

Accept k_{new} as current $k_{current}$ with

probability $P = \exp\left[\frac{-(E(k_{new})-E(k_{current}))}{T}\right]$.

[1]Note that the variables x for the formula $E(x)$ are a full membership matrix, M and the cluster centers, μ, and so $x = (\mu, M)$. In order to make the notation simpler, we commit some "abuse of notation "and use $x = k$, where k is the number of clusters. Note that this is a valid assumption if, for each k, a *unique* pair (μ, M) is returned by fuzzy k-means, e.g. using an arbitrary initialization and a fixed value for b.

$k_{current} \leftarrow k_{new}$.

S4. *Updating Step*

Update temperature T based on the annealing schedule.

If T is above the threshold δ, go to S2.

Otherwise, terminate

We have applied our approach to various data sets, and obtained quite good results. In the next chapter, we show the results on two of the datasets.

5.2 Finding a Optimal Pair (k,b)

In the last section, there is only one variable for simulated annealing, the number of clusters k. In this section, we apply the approach into a two-variable search, the number of cluster k and fuzziness parameter b.

In order to apply the approach on this two-variable searching, three extensions to the approach originally presented in previous section were introduced.

In *Initialization Step*, k_0 and b_0 are randomly selected. The initial energy, $E(k_0, b_0)$ is calculated.

The second extension is in *Generation Step*, k_{new} and b_{new} are randomly chosen. The new energy, $E(k_{new}, b_{new})$ is calculated.

The third extension is the calculation of the energy. In this subsection, the objective function is changed to

$$E(k,b) = \begin{cases} \mathbf{J_F} + I, & \text{if } I_{opt} = I_{min} \\ \mathbf{J_F} - I, & \text{if } I_{opt} = I_{max} \end{cases} \tag{5.1}$$

in which $\mathbf{J_F}$ is computed as in Equation (2.5), and I is a validity index, which can be *XB*, *I Index*, *CH*, or *DB*. Thus, the algorithm seeks the minimum value of the function $E(k,b)$.

In Section 2.1.2, based on Equations (2.6) and (2.7), fuzzy k-means finds a clustering structure whose means μ and the partition membership M minimize $\mathbf{J_F}$ (Equation (2.5)). A minimal value of $\mathbf{J_F}$ maps a clustering structure with its means $\mu_j, j = 1, \ldots, k$, and partition membership M. In Section 2.3, we mentioned that validity indices are typically used to determine the number of clusters, k. Thus, in our approach, to determine the optimal pair (k,b), the energy $E(k, b)$ is calculated by using Equation (5.1).

SA aims to find a state with minimum energy. To adopt an objective function, we use $\mathbf{J_F} - I$ for validity indices XB and DB which are *maximized*, and for validity indices CH and I *Index*, $\mathbf{J_F}+I$ is used.

For this extension, normalization of $\mathbf{J_F}$ and I is needed. In this case, the values of $\mathbf{J_F}$ and I are normalized respectively by $((V_{original} - \mu_s)/\sigma_s)$, in which $V_{original}$ is the original values of $\mathbf{J_F}$ or I, μ_s is the mean of 1,000 seeds that are selected randomly from $V_{original}$, and σ_s is the standard deviation of 1,000 seeds from $V_{original}$.

The following pseudo code formalizes the procedure for finding a nearly optimal pair of k and b by using the simulated annealing approach. The algorithm receives a data set D, and a threshold δ, as parameters.

S1. *Initialization Step*

Initialize the temperature $T \leftarrow T_0$.

Randomly choose values of k_0 and b_0

Run fuzzy k_0-means

Obtain the initial energy, $E(k_0, b_0)$.

$k_{current} \leftarrow k_0$ and $b_{current} \leftarrow b_0$

S2. *Generation Step*

Randomly choose k_{new} and b_{new}

Run fuzzy k_{new}-means.

Calculate the energy $E(k_{new}, b_{new})$.

S3. *Selection Step*

If $E(k_{new}) < E(k_{current})$

$k_{current} \leftarrow k_{new}$,

$b_{current} \leftarrow b_{new}$

else

Accept k_{new} and b_{new} as current $k_{current}$ and

$b_{current}$ with probability

$P = \exp\left[\frac{-(E(k_{new},b_{new})-E(k_{current},b_{current}))}{T}\right]$.

S4. *Updating Step*

Update temperature T based on the annealing schedule.

If T is above the threshold δ, go to S2.

Otherwise, terminate.

5.3 Initialization of fuzzy k-means clustering

In k-means/fuzzy k-means clustering, there are two initialization methods based on which one, the means μ or partition membership M, is randomly set in *Initialization Step.*

In the experiments of our approach, initializing the means μ is adopted in the experiments to find the optimal pair (k,b) on yeast data set from Cho et al [4], and the rest of experiments randomly set the partition membership M in *Initialization Setp.* The results of these experiments are discussed in the next chapter.

5.4 Conclusion

In this chapter, we proposed our optimization approach to determine the parameters of fuzzy k-means clustering algorithm. Two applications using the approach,

determining the optimal number of k and determining the optimal pair (k,b), were introduced. Two initialization methods in fuzzy k-means were presented, which will be used in the next chapter.

Chapter 6

Experiments and Discussions

To analyze the performance of our algorithms, we applied our approach on synthetic (i.e. artificial) data sets as well as real-world data sets. The value of b, the fuzziness parameter, is fixed to 1.25 for all experiments involving finding the best number of clusters, k, only.

6.1 Experiments for Finding the Best Number of Clusters

To test our clustering method, we have implemented experiments on two different data sets. One such data set is derived from time-series experiments of the yeast cell cycle[1], which has 17 time points of expression data that correspond to synchronized yeast cells over a period of two cell cycles with over 6,000 genes.

The other data set is a two-dimensional overlapping synthetic data set, which is generated using normal distribution with 1,000 data points grouped in 10 clusters. Fig.6.1 shows the data set in a two dimensional plot. For each cluster, the identity matrix is used as the covariance matrix.

[1]The data set can be obtained from http://arep.med.harvard.edu/ExpressDB/.

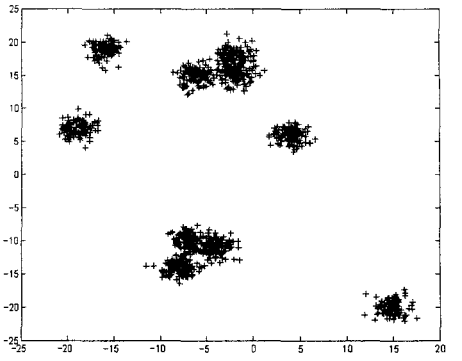

Figure 6.1: The synthetic data set with 1,000 samples grouped into 10 clusters.

To see if SA is able to find (near) optimal solutions, we first ran a brute-force procedure to generate the validity values corresponding to all values of $k = 2, \ldots, 100$. For each k, the clustering structure is obtained by fuzzy k-means, and the validity value is calculated using the indices.

Figs. 6.2 to 6.5 show the curve of validity values on the real-life microarray data set. Figs. 6.6 to 6.9 show the curve of validity values on the synthetic data. The x-axis represents k, the number of clusters, while the y-axis is the validity value to assess the quality of the clustering structure obtained by fuzzy k-means with the corresponding k, where $b = 1.25$.

Next, we applied SA on the same data sets, as above, to assess its ability to find the best value of k. If SA performs well, then it should find the optimal (or near optimal) values of k as illustrated on the validity curves. Then, by using simulated annealing on the curve of validity values, the algorithm is tested. In these experiments, $k_{min} = 2$, $k_{max} = 100$ and $T_0 = 10,000$. The SA algorithm runs until the temperature T is below a threshold, $\delta = 0.001$.

By adjusting the parameters in the annealing schedule $T(t) = \alpha^t T_0$, we tested the algorithm 11 times with different search iterations individually, 81, 68, 58, 51, 45, 40, 36, 32, 29, 26 and 24. For example, 81 search iterations means that the algorithm searches 81 out of 100 optional configurations, in this case, to find the

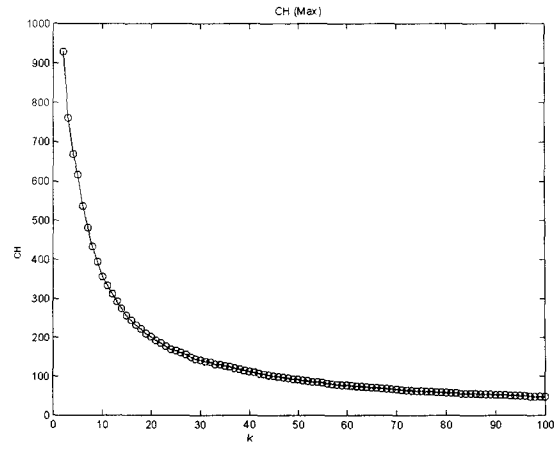

Figure 6.2: The validity values of the CH index for fuzzy k-means on the yeast microarray data set.

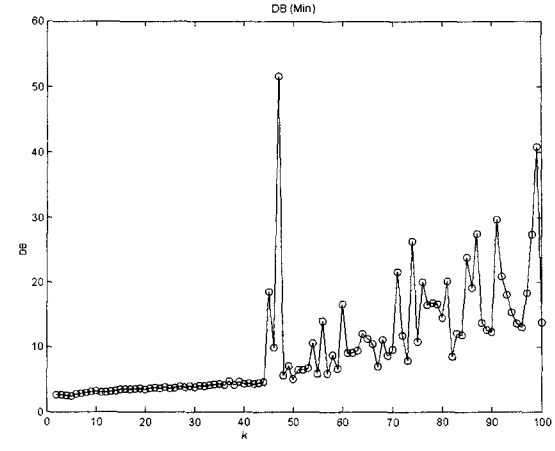

Figure 6.3: The validity values of the DB index for fuzzy k-means on the yeast microarray data set.

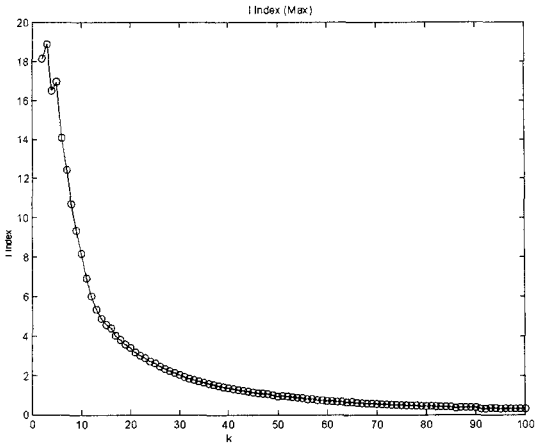

Figure 6.4: The validity values of the I Index for fuzzy k-means, plotted for values of $k = 2, \ldots, 100$, on the yeast microarray data set.

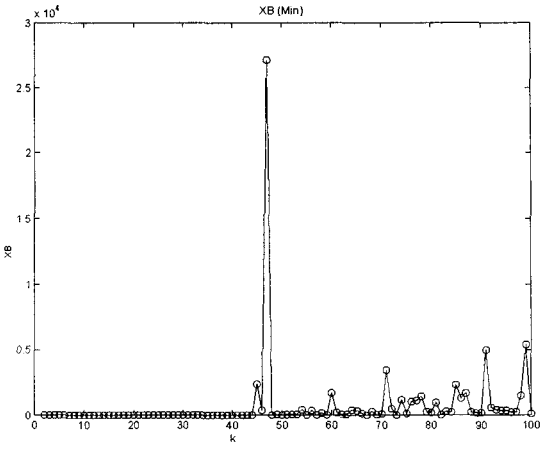

Figure 6.5: The validity values of the XB index obtained by using fuzzy k-means, corresponding to $k = 2, \ldots, 100$, on yeast microarray data set.

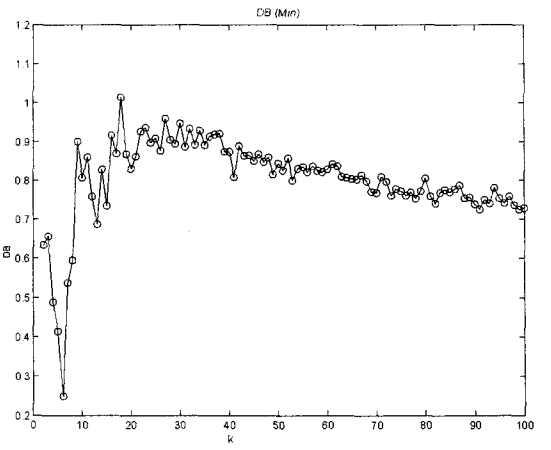

Figure 6.6: The validity values of the DB index on the synthetic data set, where $k = 2, \ldots, 100$.

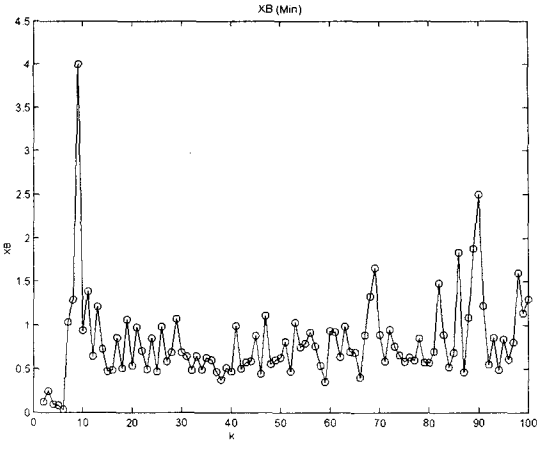

Figure 6.7: The validity values of the XB index on the synthetic data set, where $k = 2, \ldots, 100$.

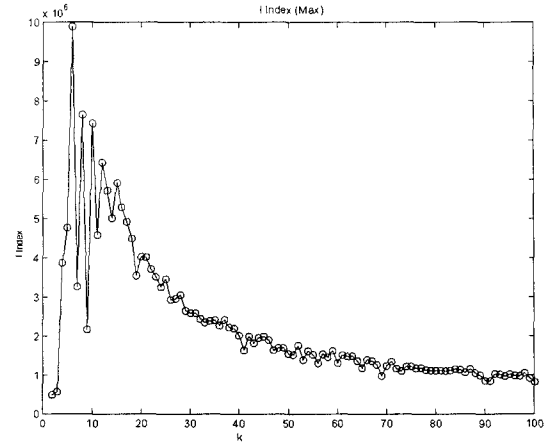

Figure 6.8: The validity values of the I Index on the synthetic data set, where $k = 2, \ldots, 100$.

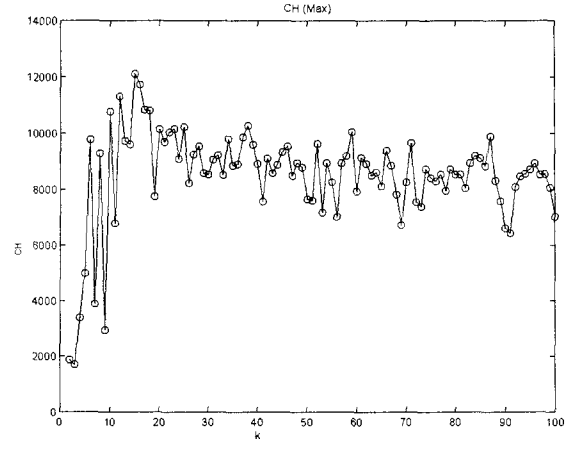

Figure 6.9: The validity values of the CH index on the synthetic data set, where $k = 2, \ldots, 100$.

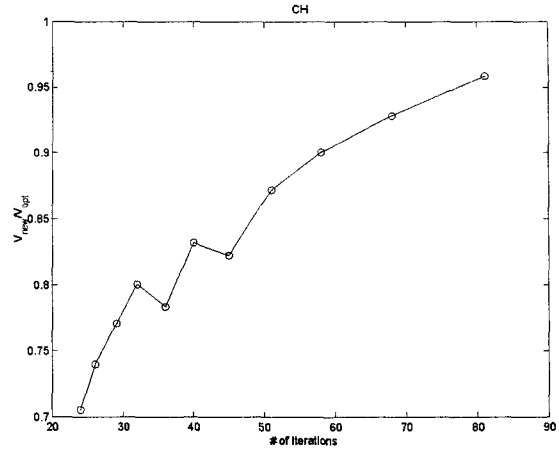

Figure 6.10: The quality of the CH validity values obtained by SA compared with the best value on the yeast microarray data set

optimal solution.

Figs. 6.10 to 6.13 show the quality of the validity value obtained compared with the best validity value on the real-life microarray data set for the four indices; and Figs. 6.14 to 6.17 show the results on the synthetic data set. In these figures, the x-axis represents the different numbers of iterations and the y-axis represents V_{new}/V_{opt}, where V_{opt} is optimal evaluation value in the corresponding validity curve and V_{new} is the average of the best validity values obtained by running this algorithm 100 times. When the value of V_{new}/V_{opt} is 1, it means that we obtain the optimal value.

We observe that the SA does a good job in obtaining values which are near to the optimal. In case of the I Index for yeast, Figure 6.12, it reaches a value of 0.987 in 81 iterations, which means that the solution found by SA is almost optimal, as can also be observed in Table 6.1. SA with the other indices also obtain high quality solution.

To demonstrate the power of our approach from another perspective, we show the results on real-life microarray data in terms of numerical data. Table 6.1 shows

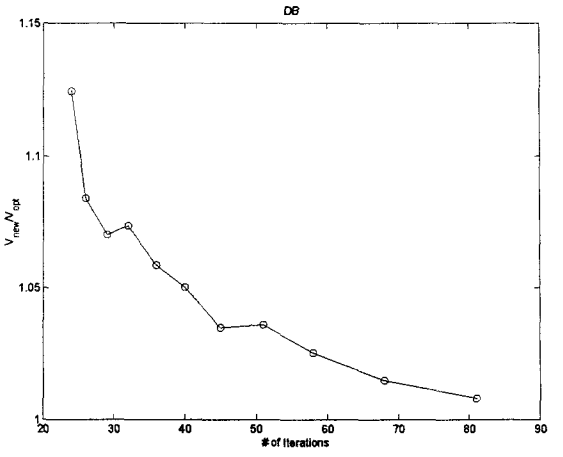

Figure 6.11: The quality of the DB validity values obtained compared with the best value on the yeast microarray data set

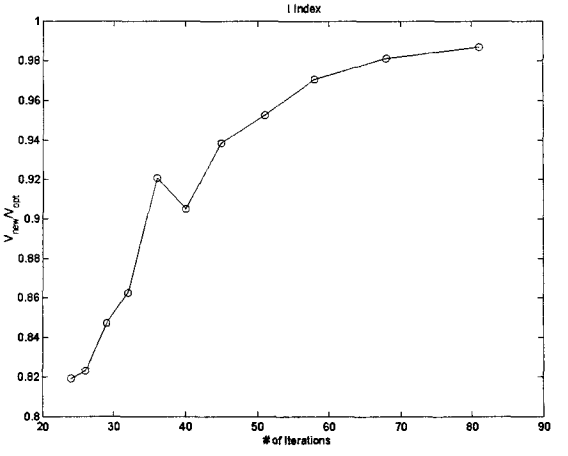

Figure 6.12: The quality of the result obtained on I Index validity values of the yeast microarray data set

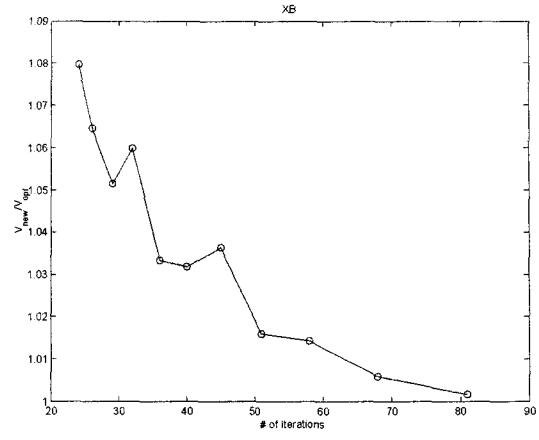

Figure 6.13: The quality of the XB validity values obtained by SA compared with the best value on the yeast microarray data set

# of itera	DB index		XB index		IndexI index		CH index	
	HitR	Solution	HitR	Solution	HitR	Solution	HitR	Solution
24	13	2.766	25	0.890	16	15.468	16	655.584
26	25	2.667	28	0.877	25	15.541	29	687.385
29	28	2.633	32	0.866	26	15.991	29	715.989
32	30	2.641	28	0.873	26	16.284	37	743.782
36	33	2.604	42	0.851	37	17.384	28	727.701
40	42	2.583	36	0.850	37	17.088	38	772.960
45	48	2.545	43	0.854	45	17.718	37	763.909
51	45	2.548	52	0.837	54	17.986	52	809.983
58	59	2.522	59	0.836	61	18.526	59	836.512
68	78	2.496	67	0.829	71	18.525	66	862.689
81	82	2.480	82	0.825	82	18.634	79	890.626
Optimal	2.460		0.824		18.879		929.199	

Table 6.1: Search quality with four indices: DB, XB, I Index and CH, which were obtained by SA on yeast microarray data.

# of itera	DB index		XB index		IndexI index		CH index	
	HitR	Solution	HitR	Solution	HitR	Solution	HitR	Solution
24	25	0.4806	29	0.1524	28	7304936	33	11395.31
26	27	0.4521	29	0.1776	23	7172090	32	11410.52
29	35	0.4291	30	0.1362	34	7606400	26	11381.30
32	28	0.4442	26	0.1479	33	7750920	29	11392.85
36	41	0.3875	39	0.1034	35	7953000	36	11534.06
40	41	0.3897	49	0.0954	48	8404510	28	11590.92
45	42	0.3790	50	0.0799	48	8436748	41	11591.98
51	46	0.3711	50	0.0753	52	8675105	51	11769.39
58	52	0.3492	52	0.0698	56	8807824	59	11849.63
68	64	0.3221	70	0.0597	71	9201801	78	11977.92
81	86	0.2733	86	0.0436	84	9527548	85	12035.17
Optimal	0.2488		0.0365		9889376		12113.34	

Table 6.2: Quality of the results obtained by SA on the synthetic data set.

the quality of the solution corresponding to 11 tests with different search iterations for each of the four indices on the yeast microarray data set. Table 6.2 shows the quality of the solution corresponding to the tests with different search iterations for each of the four indices on the synthetic data set. The column "*# of itera*" shows how many iterations the algorithm runs to search the optimal value. "*HitR*" is the number of times SA reaches its optimal value in 100 runs. "*Solution*" is the value obtained by the algorithm. The last row contains the optimal value for each of the four indices. We observe that the value obtained by SA becomes close to the optimal value gradually, while the number of iterations increases, and certainly the time to search the optimal value grows.

6.2 Experiments for Finding the Optimal Pair of k and b

In these experiments, we seek the best pair of k and b using our SA approach. To test our approach, we have run experiments with two data sets: one is the yeast

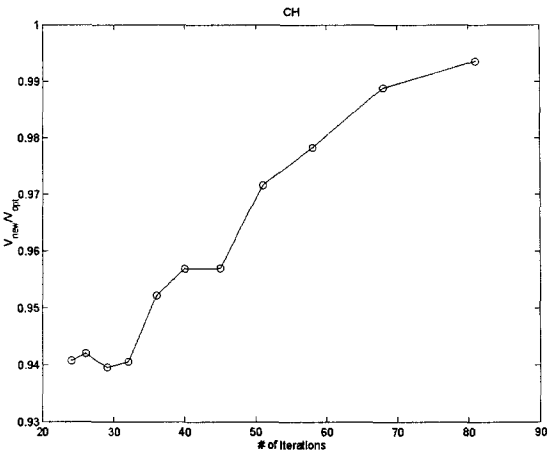

Figure 6.14: The quality of the CH validity values obtained by SA on the synthetic data set.

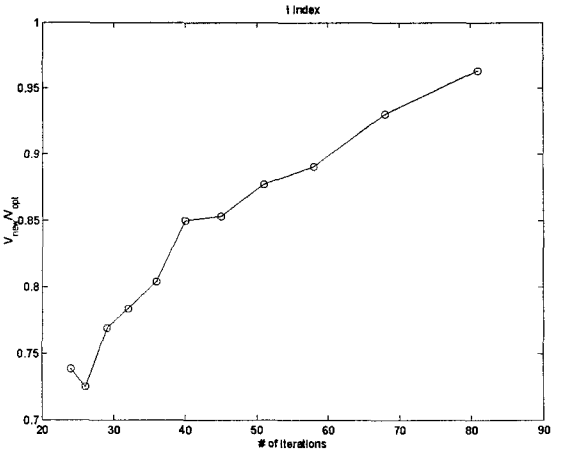

Figure 6.15: The quality of the I Index validity values obtained by SA on the synthetic data set.

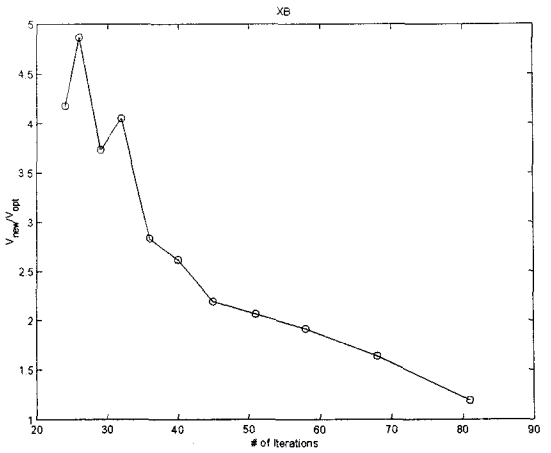

Figure 6.16: The quality of the XB validity values obtained by SA on the synthetic data set.

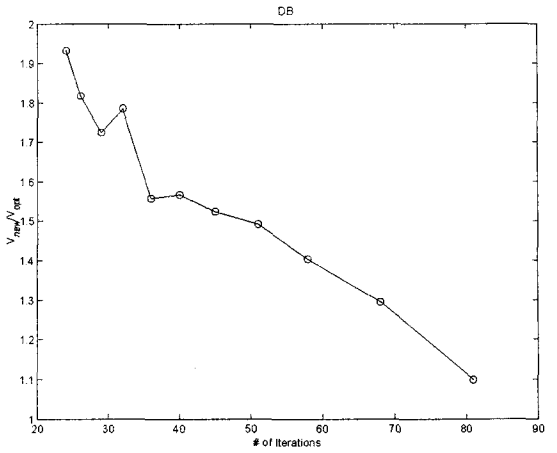

Figure 6.17: The quality of the DB validity values obtained by SA on the synthetic data set.

microarray data set, and the other is the serum data set[2], which contains 517 genes whose expression vary in response to serum concentration in human fibroblasts.

Again, we have run a brute-force procedure that generates the validity values for all pairs of values ($k = 2, \ldots, 100$, and $b = 1.05, \ldots, 3.00$, for the yeast microarray data set and $k = 2, \ldots, 60$ and $b = 1.05, \ldots, 3.00$, for the serum microarray data set). For each pair of k and b, the corresponding clustering structure is obtained. In every iteration, the search method simply accesses these values directly, instead of clustering the data points.

Figs. 6.18 to 6.21 show the objective values on yeast with $k = 2, \ldots, 100$ and $b = 1.05, \ldots, 3.00$. Figs. 6.22 to 6.25 show the values for each index on serum with $k = 2, \ldots, 60$ and $b = 1.05, \ldots, 3.00$. The x-axis represents k, the number of clusters, the y-axis represents the values of b, and the z-axis is the value of the objective function which is used to measure the quality of the clustering structure obtained by fuzzy k-means with the corresponding values k and b.

The parameters for our SA algorithm were set to $k_{min} = 2$, $k_{max} = 100$ (or $k_{max} = 60$ for serum) and $T_0 = 10000$. The algorithm runs until the temperature T reaches the threshold σ, set to 0.0005. By adjusting the parameter in the annealing schedule $T(t) = \alpha^t T_0$, we tested the algorithm 11 times with different search iterations for yeast and 9 times for serum. We note that for indices XB and DB the plots show highly peaked regions. this behavior cause some difficulties in finding the optimal parameters, as it will be discussed later. The plots for CH and I Index are smoother, and hence much easier to explore using heuristics such as SA (see later discussions)

Fig. 6.26 shows the quality of the results obtained compared with the best value on yeast, while Fig. 6.27 shows the results on serum. In these figures, the x-axis represents the number of iterations and the y-axis is V_{new}/V_{opt}, where V_{opt} is the optimal validity value of the corresponding index and V_{new} is the average of the

[2]The data set can be downloaded from http://www.sciencemag.org/feature/data/984559.shl.

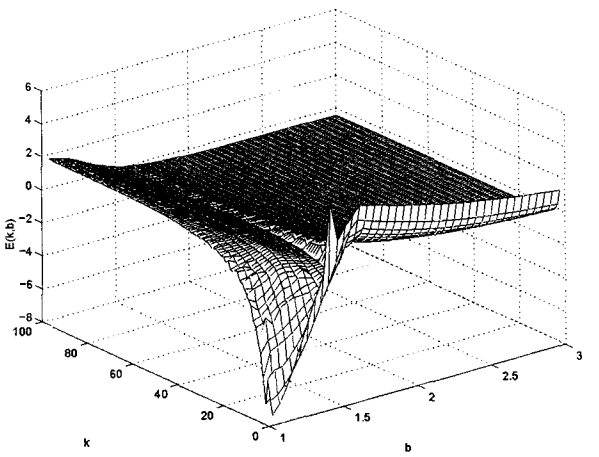

Figure 6.18: The objective values of the I Index for seeking the optimal pair of k and b, on the yeast microarray data set.

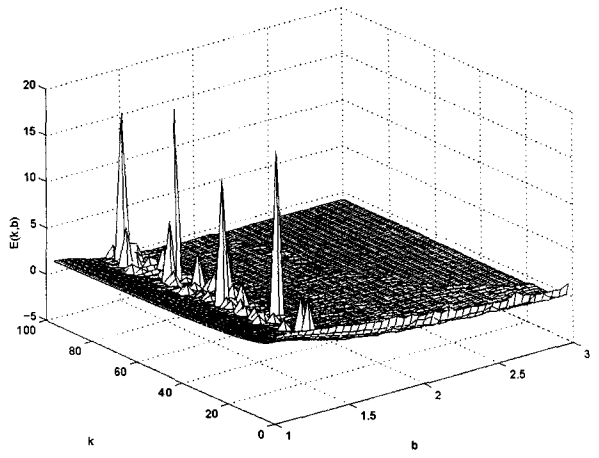

Figure 6.19: The objective values of the DB index for seeking the optimal pair of k and b, on the yeast microarray data set.

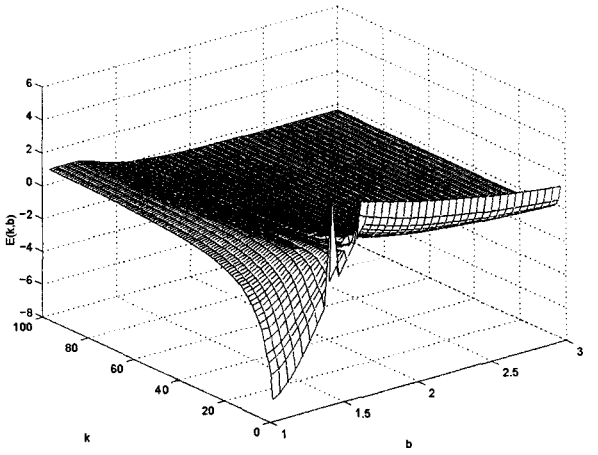

Figure 6.20: The objective values of the CH index for seeking the optimal pair of k and b, on the yeast microarray data set.

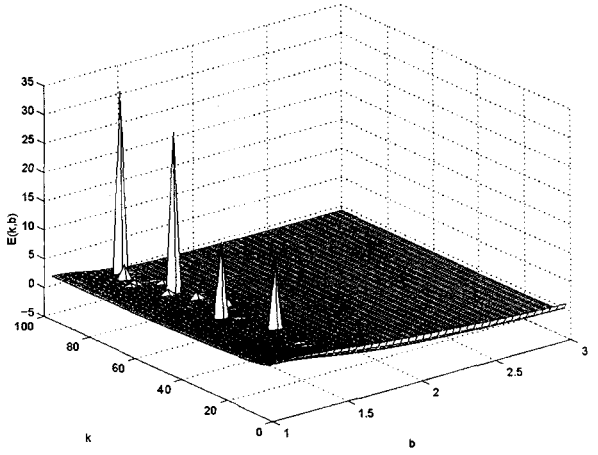

Figure 6.21: The objective values of the XB index for seeking the optimal pair of k and b, on the yeast microarray data set.

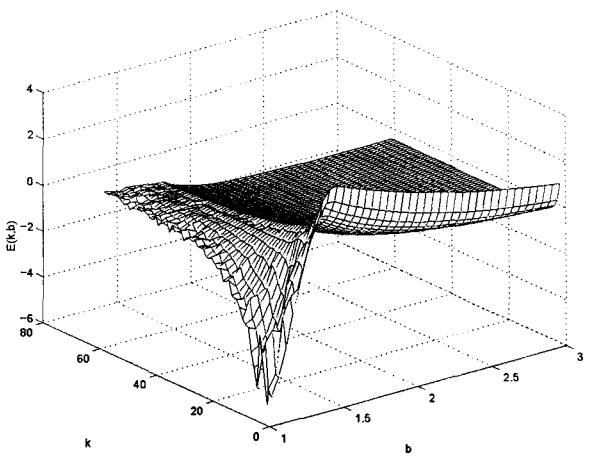

Figure 6.22: The objective values of the I Index for seeking the optimal pair of k and b, on the serum data set.

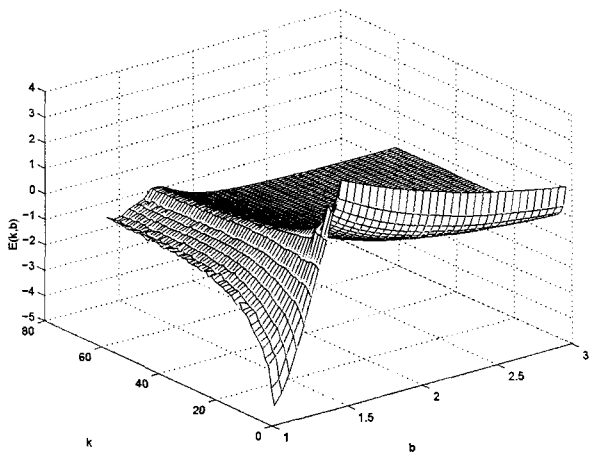

Figure 6.23: The objective values of the CH index for seeking the optimal pair of k and b, on the serum data set.

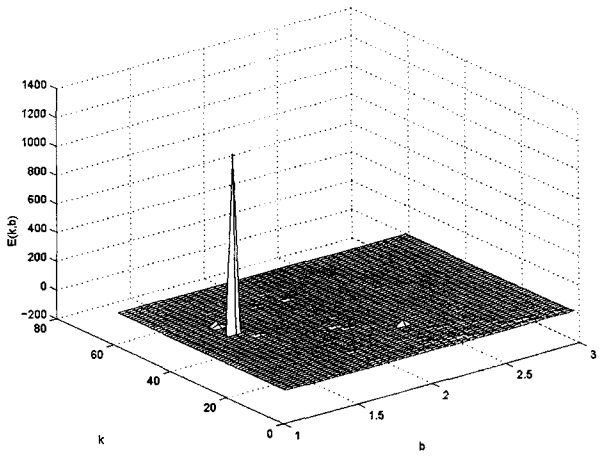

Figure 6.24: The objective values of the XB index for seeking the optimal pair of k and b, on the serum data set.

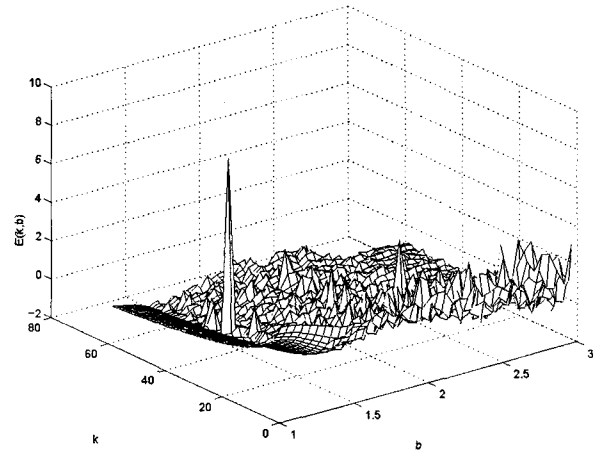

Figure 6.25: The objective values of the DB index for seeking the optimal pair of k and b, on the serum data set.

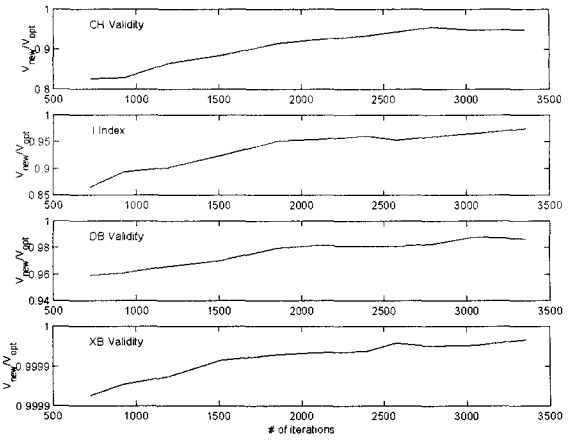

Figure 6.26: Quality of the objective values obtained by SA on the yeast data set, compared with the best values.

values obtained by running the algorithm 100 times. When the value of V_{new}/V_{opt} is 1, it means that we obtained the optimal value.

To demonstrate the results from another perspective, we show the results in terms of numerical data. Tables 6.3 and 6.4 show the number of times the optimal value was obtained by SA corresponding to 11 iterations for each of $E(k,b)$ on the two data sets. The column "*# of it*" shows how many iterations the algorithm runs to find the optimal value. "*HitR*" is the number of times SA reaches its optimal value in 100 runs. "*Solution*" is the value obtained by the algorithm. The last row contains the optimal value for each of $E(k,b)$. The value obtained by SA becomes close to the optimal value gradually, as the number of iterations increases, this reflects the efficiency of our SA approach in finding the optimal parameters.

6.3 Comparison with Pre-Clustered Yeast Data

Cho et al. [4] showed that there are 416 yeast genes that demonstrate consistent periodic changes in transcript levels, and 232 functionally characterized genes whose transcripts display periodic fluctuation. The genes are listed based on their biological

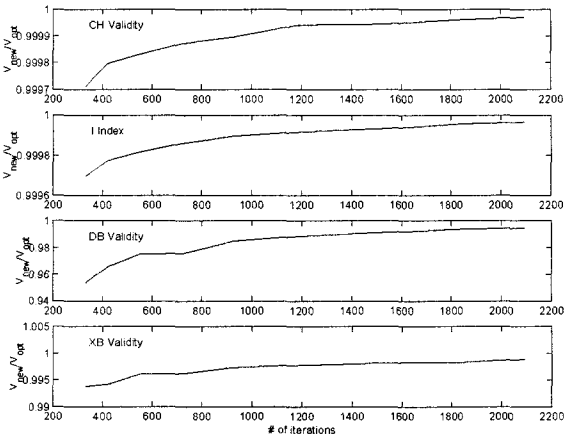

Figure 6.27: Quality of the objective values obtained by SA on the serum data set, compared with the best values.

# of it	J+DB index		J+XB index		J+IndexI index		J+CH index	
	HitR	Solution	HitR	Solution	HitR	Solution	HitR	Solution
722	20	-1.05946	20	-0.758778	15	-6.56859	24	-5.22239
925	16	-1.06156	18	-0.758789	22	-6.78657	18	-5.24081
1192	23	-1.06703	23	-0.758796	24	-6.84594	26	-5.46262
1519	26	-1.07202	32	-0.758812	33	-7.02020	35	-5.59476
1859	44	-1.08229	40	-0.758817	43	-7.22585	38	-5.78348
2092	49	-1.08435	37	-0.758819	46	-7.24943	36	-5.84386
2393	45	-1.08372	35	-0.758820	49	-7.29640	43	-5.90026
2577	46	-1.08354	54	-0.758828	44	-7.23825	52	-5.96485
2793	47	-1.08553	49	-0.758825	47	-7.28754	62	-6.03839
3048	64	-1.09149	55	-0.758826	56	-7.33337	57	-5.99178
3359	59	-1.08986	62	-0.758331	66	-7.39885	55	-6.00166
Optimal	-1.10469		-0.758844		-7.59599		-6.32311	

Table 6.3: The quality of results for the optimal pair of (k, b) on yeast data set

# of it	J+DB index		J+XB index		J+IndexI index		J+CH index	
	HitR	Solution	HitR	Solution	HitR	Solution	HitR	Solution
336	12	-1.40325	8	-1.050922	9	-1.134337	11	-1.176123
424	14	-1.42105	10	-1.051390	11	-1.134425	19	-1.176223
553	19	-1.43414	27	-1.053334	13	-1.134472	18	-1.176263
724	17	-1.43501	21	-1.053439	23	-1.134523	23	-1.176312
927	36	-1.44848	30	-1.054667	28	-1.134562	25	-1.176339
1194	39	-1.45416	42	-1.055103	30	-1.134587	43	-1.176393
1521	50	-1.45813	47	-1.055562	37	-1.134605	43	-1.176397
1861	52	-1.46200	51	-1.055721	52	-1.134633	49	-1.176418
2094	53	-1.46313	66	-1.056294	58	-1.134642	59	-1.176426
Optimal	-1.47094		-1.057468		-1.134682		-1.176462	

Table 6.4: The quality of results for the optimal pair of (k, b) on serum data set.

function under each phase in the yeast cell cycle.

In these experiments, we applied our SA approach to seek the optimal pair of k and b on a subset of the yeast data set consisting of 210 genes that Cho et al. grouped into 5 clusters based on their biological functionality. The clusters are denoted by 'Early G1', 'Late G1', 'S', 'G2'and 'M'.

The experiments are similar to the experiments for finding the optimal pair of k and b, as detailed in Section 6.2, except $k_{min} = 2$ and $k_{max} = 20$, and the initialization of fuzzy k-means was done in two different ways: one is that k samples are randomly selected as the means of the clusters, and the other is that $2k$ samples are selected, and the means the clusters were computed by taking k pairs of samples whose distances are the shortest.

The optimal pair (k,b) obtained by SA with the combined objective function $\mathbf{J_F}$ and CH/I Index is (4, 1.05). Table 6.5 shows the clustering accuracy of the memberships. Each cell in the table contains the number of genes in the intersection of the 5 clusters categorized biologically and clusters obtained by our SA. We observe that, though unsupervised, the best assignment of biological clusters into the resulted clusters is that Early G1 into the 1^{st} cluster, Late G1 into the 2^{nd} cluster,

	1^{st}Cluster	2^{nd}Cluster	3^{rd}Cluster	4^{th}Cluster
Early G1	**26**	4	0	0
Late G1	8	**70**	3	0
S	2	12	**25**	4
G2	1	2	10	**13**
M	4	1	1	**24**

Table 6.5: The clustering accuracy on Cho yeast data set with J combined with the indices, Index and CH.

S into the 3^{rd} cluster and G2 and M into the 4^{th} cluster, whose accuracy is 75.23% [3]. We observed, on the other hand, that the clustering structures obtained by using the combined function $\mathbf{J_F}$ and XB/DB are not stable, and the optimal pair of k and b is (20, 3.0), which is not reliable, since various experiments in the literature show that the best value of b is between 1.05 and 1.25. Such un-stability on DB and XB can be demonstrated by observing figures 6.18 through 6.25. All figures associated with XB and DB show search landscape that contain highly peaked regions (with narrow peaks), whereas those associated with I Index and CH have no peaks and show a very smooth landscape. SA yields poor results on DB and XB because it is harder to find the top of the tallest peaks among many peaks with almost flat surrounding areas.

6.4 Experiments with Tabu List

In this section, a comparison between with and without Tabu list [1] is presented. First, a description about Tabu list is presented, and then the design of the experiments is introduced. In the last part, the results are given.

[3] the accuracy is computed as follews: $\frac{26+70+25+13+24}{n} = 75.23\%$, where n is the total number of genes

6.4.1 What is Tabu List?

Tabu list is derived from another optimization method, Tabu search [1]. Tabu search that is an optimization technique that uses a form of short-term memory to keep a search from becoming trapped in a local minimal value. This short-term memory is named Tabu list that keeps track of recent solutions. At each iteration in the optimization process, new solutions are checked against the Tabu list. The solutions that are in Tabu list will not be chosen for the next iteration.

6.4.2 Design of Experiments with Tabu List

In these experiments, Tabu list is introduced into our search approach to find the optimal value of k. In each iteration of searching process, a new configuration is checked first in Tabu list. If this configuration is in Tabu list, called *hitting* the time-consuming fuzzy k-means clustering, will not be executed and the energy of this configuration is accessed directly from corresponding memory. Otherwise, fuzzy k-means clustering will not be avoided and then the energy computed using validity indices is stored into a memory for the following iteration.

For the algorithm without Tabu list, the computation of clustering and energy for each configuration can not be avoided. Thus, the efficiency of the algorithm with Tabu list is measured by counting the average hitting rate of the Tabu list. The more hitting rate the Tabu list yields, the more computation is avoided and more time is saved for the algorithm with Tabu list.

To obtain the average hitting rate of Tabu list, the algorithm is run ten times and the average hitting rate is the ratio of the sum of hitting in this ten times running to the sum of the number of configurations totally.

In this section, to show the efficiency of the algorithm with Tabu list, 11 experiments with different search iterations, 81, 68, 58, 51, 45, 40, 36, 32, 29, 26 and 24, were carried out. Figs. 6.28 to 6.31 show the average hitting rates for these 11

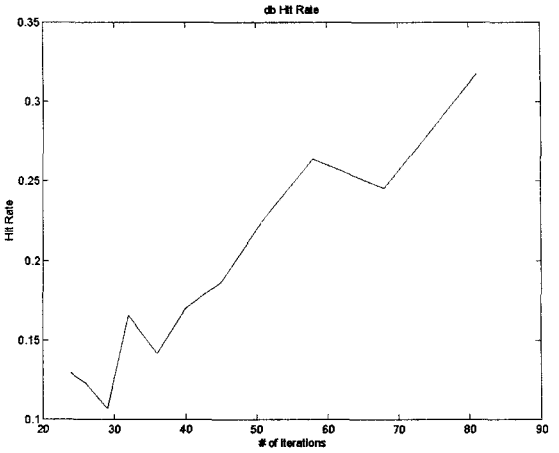

Figure 6.28: The curve of hitting rate of the DB index on the yeast microarray data set with different iterations.

experiments, in which, the energy was calculated with the four indices: CH, I Index, DB and XB. In these figures, the x-axis represents the different search iterations that the experiments has, while the y-axis represents the average hitting rates.

To demonstrate the results from another perspective, we show the results on real-life microarray data in terms of numerical data. Table 6.6 shows the hitting rate corresponding to 11 tests with different search iterations for each of the four indices on the yeast microarray data set. The column "*# of iterations*" shows how many iterations the algorithm runs to search the optimal value. "*CH index*", "*IN Index*", "*XB index*" and "*DB index*" are the average hitting rates that configurations were found in Tabu List on the procedure to the optimal solution in 10 runs.

Table. 6.6 and Figs. 6.28 to 6.31 show that with Tabu list, our SA algorithm can obtain over 30% hitting rate under 81 search iterations. That is, over 30% time-consuming fuzzy clustering will be saved. And the algorithm with Tabu list is faster than the algorithm without Tabu list.

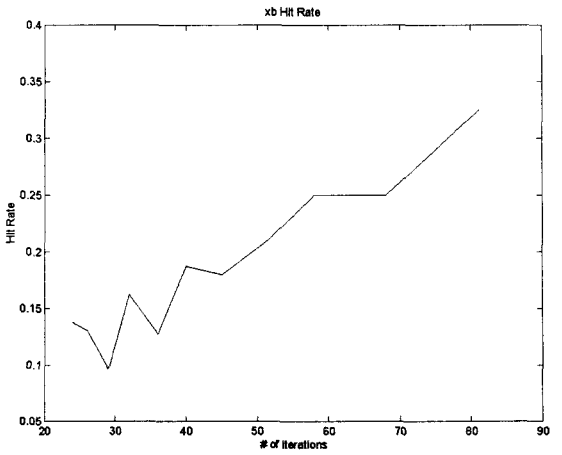

Figure 6.29: The curve of hitting rate of the XB index on the yeast microarray data set with different iterations.

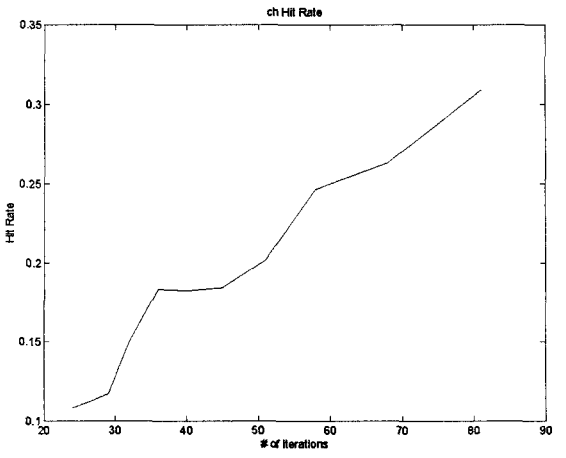

Figure 6.30: The curve of hitting rate of the CH index on the yeast microarray data set with different iteration numbers.

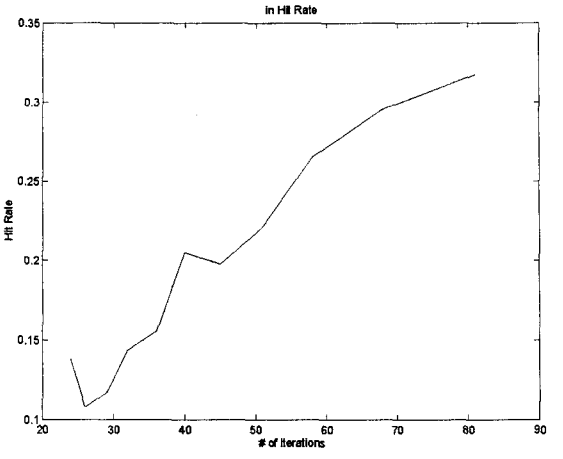

Figure 6.31: The curve of hitting rate of the I Index on the yeast microarray data set with different iteration numbers.

# of iterations	CH index	I Index	XB Index	DB index
24	10.83	13.75	13.75	12.91
26	11.15	10.77	13.07	12.30
29	11.72	11.72	9.65	10.68
32	15.00	14.37	16.25	16.56
36	18.33	15.55	12.77	14.16
40	18.25	20.50	18.75	17.00
45	18.44	19.77	18.00	18.66
51	20.19	22.15	20.78	22.54
58	24.65	26.55	25.00	26.37
68	26.32	29.55	25.00	24.55
81	30.86	31.72	32.46	31.72

Table 6.6: The Hitting rate with Tabu List by using four indices.

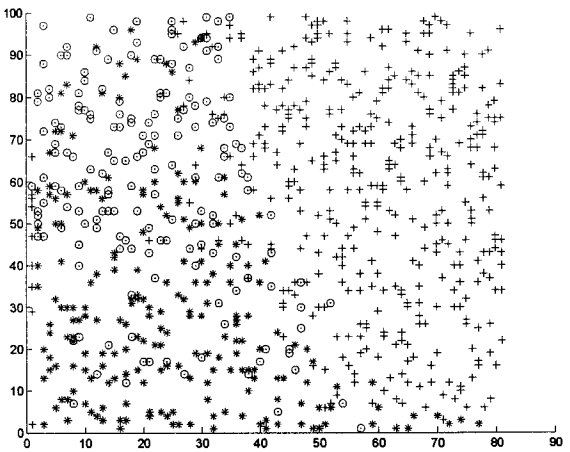

Figure 6.32: The procedure of SA using the DB index without shrinking the scope of randomization.

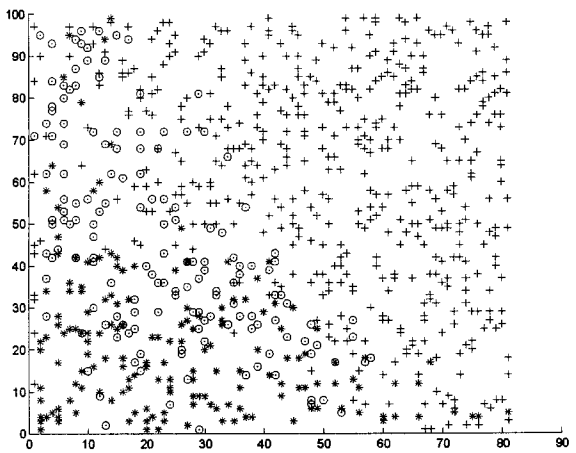

Figure 6.33: The procedure of SA using the XB index without shrinking the scope of randomization.

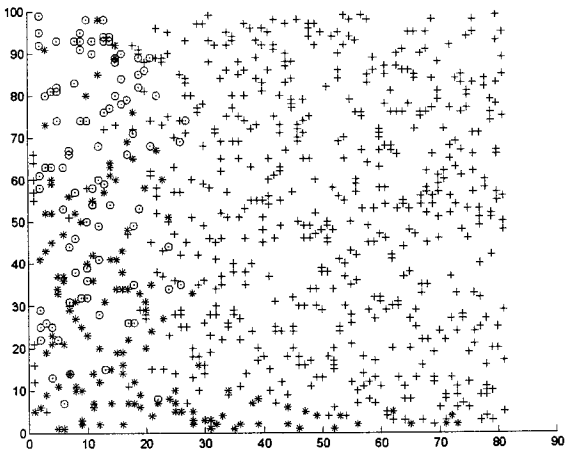

Figure 6.34: The procedure of SA using the CH index without shrinking the scope of randomization.

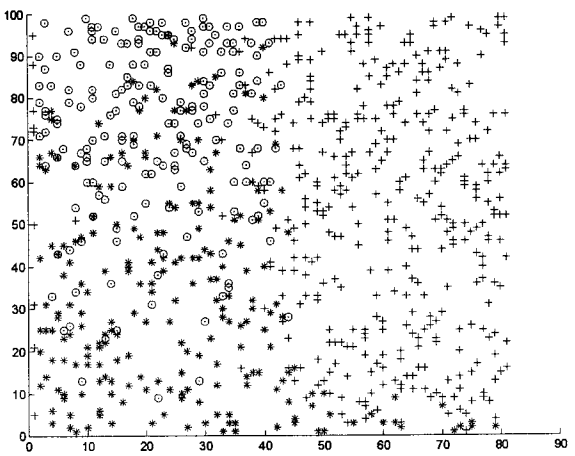

Figure 6.35: The procedure of SA using the I Index without shrinking the scope of randomization.

6.5 Experiments with '*Shrinking*' the Scope of Randomization

As We mentioned earlier, annealing schedule controls the running of SA. SA will keep running until the temperature reaches a certain threshold. The threshold determines the number of search iterations. One question that typically arises is how to determine the threshold. Smaller thresholds yield larger running time and larger thresholds yield weaker solutions.

Figs. 6.32 to 6.35 show the different kinds of configurations that are randomly selected during the procedure of SA running 10 times. 'Star' points present the better configurations accepted directly as current solutions, 'cycle' points present the worse configurations rejected with the probability, and 'plus' points present the configuration accepted with the probability. Based on these figures, we can find that the 'star' points converge close to the final solution early, while the 'plus' points increase rapidly.

There are two reasons that cause the unnecessary extra running of SA: the threshold is either too small, or the fixed range of randomly selecting a configuration.

In this experiment, We added a self-adjusted ending condition into SA algorithm to speed up the convergence toward optimal solution. The range of randomly selecting the configuration will shrink based on the following equation:

$$k_{new} = random(R_{down}, R_{up}), \tag{6.1}$$

where the upper bound of the range $R_{up} = k_{current} + (50 - NB \times v)$ and the lower bound of the range $R_{down} = k_{current} - (50 - NB \times v)$, NB is the number of the configurations that are rejected with probability, showing as 'plus' points in these figures and v is the parameter to control the scale of shrinking. For instance, $v = 2$ means the range will be shrunk by 2 for each 'plus' point encountered.

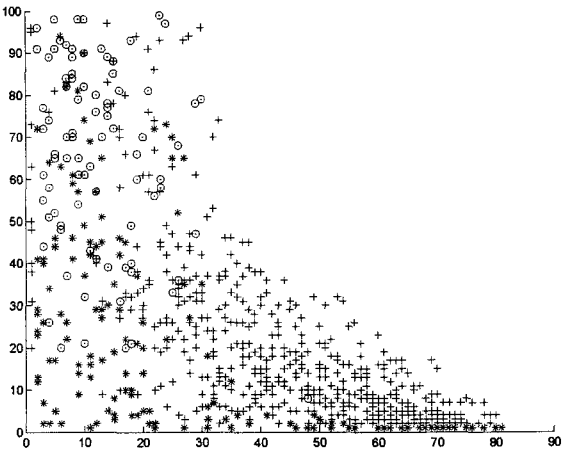

Figure 6.36: The procedure of SA using the CH index with shrinking the scope of randomization by 1.

Another contribution of this improvement is that the second ending condition is introduced into the SA: when $R_{up} - R_{down} \leq 1$, that means that if there is no space for selecting, then the algorithm terminates.

Figs. 6.36 to 6.39 show the procedure of shrinking the range of randomization with the different scales by using CH index. In Fig. 6.36, we can see that although the computation is as much as that in SA without shrinking, the 'plus' points are converged to the optimal solution and the intensity of 'star' points is increased around the optimal values. Figs. 6.37 and 6.38 show that the computation is less time consuming than that without shrinking the range even up to about 50%. Fig. 6.39 shows less improvement than Fig. 6.38, because of the fewer 'plus' points that can be used to shrink the range.

6.6 Conclusion

In this chapter, to verify the power of our approach, some experiments are presented, and to enhance the efficiency of the approach, two methods are applied, searching with Tabu List and shrinking the scope of randomization.

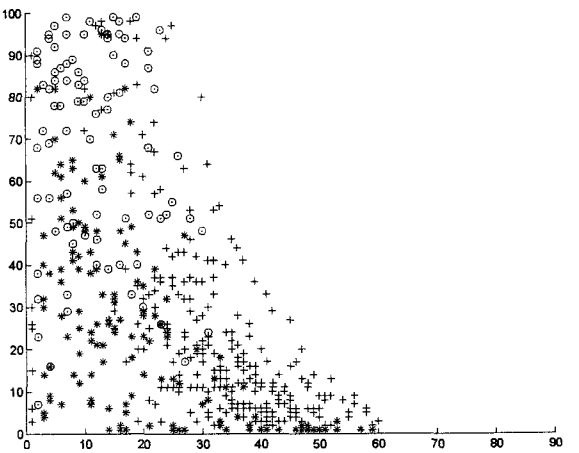

Figure 6.37: The procedure of SA using the CH index with shrinking the scope of randomization by 2.

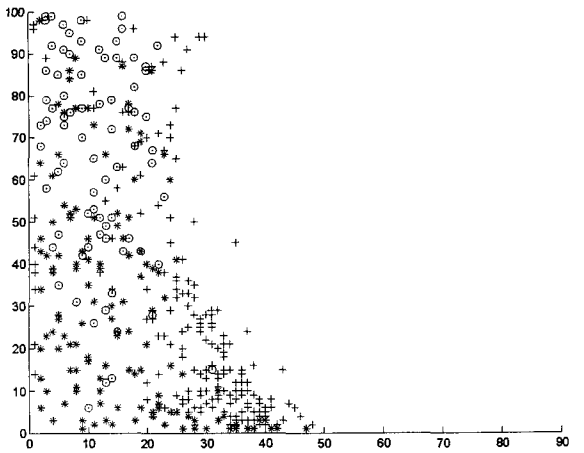

Figure 6.38: The procedure of SA using the CH index with shrinking the scope of randomization by 3.

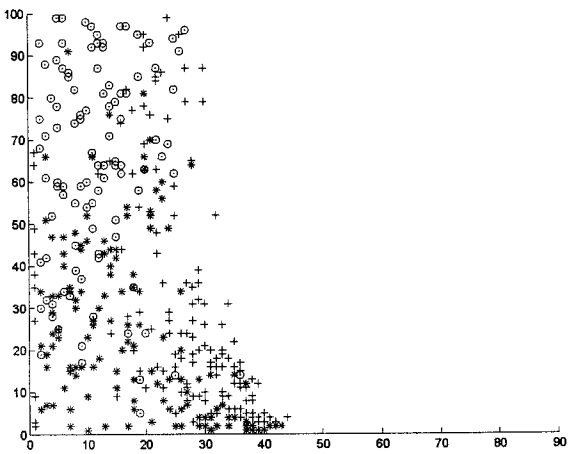

Figure 6.39: The procedure of SA using the CH index with shrinking the scope of randomization by 4.

Chapter 7

Conclusions and Further Work

We have presented a new method to optimize the number of clusters, k, and the fuzziness parameter, b, in fuzzy clustering microarray time-series data. A nearly optimal value for the pair of k and b can be reached using simulated annealing on gene expression data. Also two methods are applied to improve our approach.

Based on the analysis of the figures and tables, our method has been found to run very quickly and accurately by 88.7% computation time for up to 66% hitting rate on optimal value. The average of the values obtained is 99.88% closed to the optimal value, which is showed for the combined function $\mathbf{J_F}$ and XB in Table 6.4. We have also validated our method using well-known biological data analyzed by Cho et al.

These results have been attained on both real-life and synthetic data. Our future work focuses on analyzing our approach for other clustering algorithms, such as k-means and expectation-maximization, as well as utilizing other distance measures.

Appendix-Source Code

Fuzzy k-means and indices function

Fuzzy k-means

```
function fuzzykmeans(StartB,EndB,StartK,EndK,Cutoff,BStep,data)
% By using the MatLab build-in function, fcm, this function
% clusters the data and compute the evaluation value based on
% the indices CH, XB, I Index and DB.
% INPUT
%  data is the dataset to cluster
%  StartB and EndB are the boundary of b
%  StartK and EndK are the boundary of k
%  Cutoff is the threshold for fuzzy k-means clustering
%  BStep is the increasing step for each iteration.
Itera = 10000;


for i = StartK:EndK
    for j = StartB:BStep:EndB


        [center,U,v] = fcm(data,i,[j 10000 CUTOFF 0]);
```

```
        chf = CH(U',data,center,i);
        xbf = XB(U',data,center,i);
        indexf = INDEX(U',data,center,i);
        dbf = DB(U',data,center,i);
    end
end
```

four indices function, XB,CH,INDEX and DB

```
function E=XB(mem,dataset,mu,K)
% INPUT:
%  mem is the membership matrix
%  dataset is the dataset used to cluster
%  mu is the means of clusters
%  K is the number of k
% OUTPUT
%  e is the evaluation value

[M,N]=size(dataset);
w=0;

for i=1:K
    for j=1:M
        w=w+mem(j,i).*mem(j,i).*(dataset(j,:)-mu(i,:))*(dataset(j,:)-mu(i,:))';
    end
end
min_mu=(mu(1,:)-mu(2,:))*(mu(1,:)-mu(2,:))';
for i=2:K
```

```
    for j=1:i-1
        dis=(mu(i,:)-mu(j,:))*(mu(i,:)-mu(j,:))';
        if dis <= min_mu
            min_mu=dis;
        end
    end
end

E=w/(M.*min_mu);



function E=INDEX(mem,dataset,mu,K)
% INPUT
%  mem is membership matrix
%  dataset is the dataset used to cluster
%  mu is the means of clusters
%  K is the number of clusters
% OUTPUT
%  E is evaluation value
[M,N]=size(dataset);
p=2;Ek=0;
for i=1:K
    for j=1:M
        Ek=Ek+mem(j,i).*sqrt((dataset(j,:)-mu(i,:))(dataset(j,:)-mu(i,:))');
    end
end
```

```
max_mu=(mu(1,:)-mu(2,:))(mu(1,:)-mu(2,:))';
for i=2:K
    for j=1:i-1
        dis=(mu(i,:)-mu(j,:))(mu(i,:)-mu(j,:))';
        if dis >= max_mu
            max_mu=dis;
        end
    end
end
Dk=max_mu;

zmu=zeros(1,N);
for i=1:M
    zmu=zmu+dataset(i,:);
end
zmu=zmu./M;
E1=0;
for i=1:M
    E1=E1+sqrt((dataset(i,:)-zmu)(dataset(i,:)-zmu)');
end

E=(1./K.*E1./Ek.*Dk)^p;



function E=CH(mem,dataset,mu,K)
% INPUT
%  mem is the membership matrix
```

```
%  dataset is the dataset used to cluster
%  mu is the means of clusters
%  K is the number of clusters
% OUTPUT
%  E is the evaluation value
[M,N]=size(dataset);
zmu=zeros(1,N);
zmu = mean(dataset);
num=zeros(1,K);
num=sum(mem);
traceb=0;
for i=1:K
    traceb=traceb+num(1,i).*((mu(i,:)-zmu)(mu(i,:)-zmu)');
end
tracew=0;
for i=1:K
    for j=1:M
        tracew=tracew+mem(j,i).*((dataset(j,:)-mu(i,:))(dataset(j,:)-mu(i,:))');
    end
end
E=traceb.*(M-K)./(tracew.*(K-1));



function E=DB(mem,dataset,mu,K)
% INPUT
%  mem is the membership matrix
%  dataset is the dataset used to cluster
```

```
%  mu the means of clusters
%  K is the number of clusters
% OUTPUT
%  E is the evaluation of value
[M,N]=size(dataset);

WD=zeros(1,K);
%Computing the within-distance of cluster
for i = 1:K
    Ci=sum(mem(:,i));
    for j = 1:M
        WD(i) = WD(i)+pdist([dataset(j,:);mu(i,:)],'Euclid').*mem(j,i);
    end
    WD(i) = WD(i)./Ci;
end

%Computing the Between-distance of cluster
BD=zeros(K,K);
for i = 1:K
    for j = 1:K
        if i ~= j
            BD(i,j) = pdist([mu(i,:);mu(j,:)],'Euclid');
        end
    end
end

E=0;
```

```
for i = 1:K
    Max=0;
    for j = 1:K
        if i ~= j
            te=(WD(i)+WD(j))/BD(i,j);
            if te>Max;
                Max=te;
            end
        end
    end
    E=E+Max;
end
E=E/K;
```

SA to Determine the Number of k

```
function [BSF,BSFK]=OptimalK(Re,RangeLow,RangeUp,T0,Tg,alpha)
% The algorithm to determine the number of k
% randint is a MatLab build-in function for random integer
% INPUT
%  RangeLow and RangeUp is the scope of the k
%  T0, Tg and alpha are the parameters of SA
%  Re is the evaluation values, also can be seen as the
%  function to cluster the dataset and return the energy in
%  real computation.
% OUTPUT
%  BSF is the best-so-far energy obtained by SA
```

```
%  BSFK is the best-so-far k obtained by SA

% the random current k
current = randint(1,1,[RangeLow RangeUp]);

% the energy of the current k
ccost = Re(1,current);

% the best-so-far energy and the k
BSF = ccost; BSFK = current;

t = 0;
while (Tg > 0.001)
    t = t + 1;
    new = randint(1,1,[RangeLow RangeUp]); % a random new k

    ncost = Re(1,new);

% store the best-so-far energy and the corresponding k
    if (ncost > BSF)
        BSFK = new;
        BSF = ncost;
    end
% accept directly toward to the next step
    if ((ncost - ccost) >= 0)
        current = new;
        ccost = ncost;
```

```
    else
% accept with the probability
        if(exp((ncost-ccost)./Tg)>=(randint(1,1,[0,10000])/10000))
            current = new;
            ccost = ncost;
        end
    end
% update the temperature
    Tg = T0*alpha.^t;
end
```

SA to determine the pair of k and b

```
function [BSF,BSFK,BSFB]=OptimalKB(Re,RLowK,RUpK,RLowB,RUpk,T0,Tg,alpha)
% The algorithm to determine the pair of k and b
% randint is a MatLab build-in function for random integer
% INPUT
%  RLowK and RUpK are the scope of the k
%  RLowB and RUpB are the scope of the b
%  T0, Tg and alpha are the parameters of SA
%  Re is an array storing the evaluation values, also can be
%  seen as the function to cluster the dataset and return the
%  energy in real computation.
% OUTPUT
%  BSF is the best-so-far energy obtained by SA
%  BSFK is the best-so-far k obtained by SA
%  BSFB is the best-so-far b obtained by SA
```

```
% the current k and b randomly selected
kc = randint(1,1,[RLowK RUpK]);
bc = randint(1,1,[RLowB RUpB]);

% the energy of the pair of current k and b
ccost = Re(kc,bc);

% the variables for best-so-far energy and the pair of k and b
BSF = ccost;
BSFK = kc;
BSFB = bc;

t = 0;
while (Tg > 0.0005)
    t = t + 1;

% the new pair of k and b
    newk = randint(1,1,[RLowK RUpK]);
    newb = randint(1,1,[RLowB RUpB]);

    ncost = Re(newk,newb);

% best-so-far energy and the pair of k and b
% here is for XB and DB
    if (ncost < BSF)
        BSFK = newk;
```

```
        BSFB = newb;
        BSF = ncost;
    end
    % accept directly
    if ((ncost - ccost) <= 0)
         kc = newk;
         bc = newb;
         ccost = ncost;
     else
 % accept with the probability
         if(exp(-(ncost-ccost)*p0./Tg)>=(randint(1,1,[0,10000])/10000))
             kc = newk;
             bc = newb;
             ccost = ncost;
         end
 % update the temperature of SA
         Tg = T0*alpha.^t;
     end
 end
```

SA to Determine the Number of k with Tabu List

```
function [BSF,BSFK]=OptimalKTabu(Re,RLowK,RUpK,T0,Tg,alpha,TabuL)
% The algorithm to determine the number of k with Tabu List
% randint is a MatLab build-in function for random integer
% INPUT
%  TabuL is the length of the Tabu List
```

```
%  RangeLow and RangeUp is the scope of the k
%  T0, Tg and alpha are the parameters of SA
%  Re is an array storing the evaluation values, also can be
%  seen as the function to cluster the dataset and return the
%  energy in real computation.
% OUTPUT
%  BSF is the best-so-far energy obtained by SA
%  BSFK is the best-so-far k obtained by SA

% random current k and its energy
current = randint(1,1,[RLowK RUpK]);
ccost = Re(1,current);

% variables for the best-so-far energy and k
BSF = ccost;
BSFK = current;

% initialization of Tabu List
TabuList = zeros(1,TabuL);
TabuListEngergy = zeros(1,TabuL);
TabuList(1,1) = current;
TabuListEnergy(1,1) = ccost;

t = 0;
while (Tg > 0.001)
% a new k randomly selected
    new = randint(1,1,[RLowK RUpK]);
```

```
    t = t + 1;

% check Tabu List,
% if not in the list, added
% otherwise, restore the value to the new k
    if (find(TabuList(1,:) == new) == [])
        offset = FindEmpty(TabuList);
        TabuList(1,offset) = new;
        ncost = Re(1,new);
        TabuListEnergy(1,offset) = ncost;
    else
        ncost = TabuListEnergy(1,find(TabuList(1,:) == new));
    end

% the best-so-far energy and k
    if (ncost > BSF)
        BSFK = new;
        BSF = ncost;
    end
    % accept directly
    if ((ncost - ccost) >= 0)
        current = new;
        ccost = ncost;
    else
% accept with the probability
        if(exp((ncost-ccost)*p0./Tg)>=(randint(1,1,[0,10000])/10000))
            current = new;
```

```
            ccost = ncost;
        end
end
% update the temperature of SA
    Tg = T0*alpha.^t;
end
```

SA to Determine the Number of k with 'shrinking'

```
function [BSF,BSFK]=OptimalKShrink(Re,RLow,RUp,T0,Tg,alpha,rcf)
% The algorithm to determine the number of k with 'shrinking'
% randint is a MatLab build-in function for random integer
% INPUT
%  rcf is the shrink step
%  RLow and RUp is the scope of the k
%  T0, Tg and alpha are the parameters of SA
%  Re is an array storing the evaluation values, also can be
%  seen as the function to cluster the dataset and return the
%  energy in real computation.
% OUTPUT
%  BSF is the best-so-far energy obtained by SA
%  BSFK is the best-so-far k obtained by SA

current = randint(1,1,[RLow RUp]);
ccost = Re(1,current);
BSF = ccost;
BSFK = current;
```

```
% the number of the points rejected with the probability
NRejected = 0;

t = 0;
while (Tg > 0.001)
    t = t + 1;
    new = randint(1,1,[RLow RUp]);
    ncost = Re(1,new);

    if (ncost > BSF) %for max
        BSFK = new;
        BSF = ncost;
    end

    if ((ncost - ccost) >= 0) % for max
        current = new;
        ccost = ncost;
    else
        if(exp((ncost-ccost)*p0./Tg)>=(randint(1,1,[0,10000])/10000))
            current = new;
            ccost = ncost;
        else
% shrink the scope of k
            NRejected = NRejected + 1;
            if (NRejected > 3)
                RLow = current - (50 - NRejected*rcf);
```

```
                RUp = current + (50 - NRejected*rcf);
                if (RLow < 1)
                    RLow = 1;
                end
                if (RUp > 99)
                    RUp = 99;
                end
                if (rcf == 0)
                    RUp = 99;
                    RLow = 1;
                end
% the second end condition for SA
% end the loops when upper bound and lower bound overlap
                if (RLow >= RUp)
                    Tg = 0.001;
                end
            end
        end
    end
% update the temperature of SA
    Tg = T0*alpha.^t;
end
```

Bibliography

[1] K.S. Al-Sultan. A tabu search approach to the clustering problem. *Pattern Recognition*, 28(2):1443–1451, 1995.

[2] A. Bellaachia and D. Portnoy. E-cast: a data mining algorithm for gene expression data. *U.S.A and National Institute of Health*, pages 49–54, 2002.

[3] A. Ben-Dor, R. Samir, and Z. Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 1999.

[4] R.J. Cho, M.J. Campbell, E. Winzeler, L. Steinmetz, A. Conway, L. Wodicka, T.G. Wolfsberg, A.E. Gabrielian, D. Landsman, D.J. Lockhart, and R.W. Davis. A genome-wide transcriptional analysis of the mitotic cell cycle. *Molecular Cell*, 12:65–73, 1998.

[5] B. Coe and C. Antler. Spot your genes-an overview of the micorarray. *BioTech*. Available at http://www.bioteach.ubc.ca/MolecularBiology/index.html. Last time accessed: Auguest 2nd,2005.

[6] D. Dembele and P. Kaster. Fyzzy c-means method for clustering microarray data. *Bioinformatics*, 19(8):973–980, 2003.

[7] S.M. Dhanasekaran, T.R. Barrete, and D. Ghosh. Delineation of prognostic biomarkers in prostate cancer. *Nature*, 412:822–826, Aug 2001.

[8] S. Draghici. *Data Analysis Tools for DNA Microarray.* Chapman & hall CRC, 2003.

[9] M.B. Eisen, P.T. Spellman, P.O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expresion patterns. *Proceeding of the National Academy of Sciences*, 95:14863–14868, 1998.

[10] M.A.S. Elmohamed and G. Fox. A comparion of annealing techniques for academic course scheduling. In *Lecture Notes in Computer Science Selected Papers from the Second Interational conference on Practice and Theory of Automated Timetabling II*, pages 92–114, 1997.

[11] D. Eppstein. Fast hierarchical clustering and other applicatioins of dynamic closest pairs. *Bioinformatics*, 17:22–29, 2001.

[12] A.P Gasch and M.B. Eisen. Exploring the conditional coregulation of yeast gene expression through fuzzy k-means clustering. *Genome Biology*, 2002.

[13] C.A. Harrington, C. Rosenow, and J Retief. Monitoring gene expression using dna microarray. *Current Opinion in Microbiology*, 3:285–291, 2001.

[14] K.H. Hoffmann and P. Salamon. The optimal simulated annealing schedule for a simple model. *J. Phys. A: Math. Gen*, 23:3511–3523, 1989.

[15] A. Hyvarinen and E. Oja. Independent component analysis: Algorithms and application. *Neural Networks*, page 411:430, 2000.

[16] D.K. Joseph, Z.B. Gifford and T.S. Jakola. Fast optimal leaf ordering for hierarchical clustering. *Bioinformatics*, 17, 2001.

[17] T. Kohonen. The self-organizing map. *IEEE*, 78:1464–1480, 1990.

[18] S.K. Lee. A self-organizing map for finding the optimal gene order in displaying microarray data. *Seoul National University.* Available at http://citeseer.ist.psu.edu/586923.html. Last time accessed: Augues 2nd,2005.

[19] R.J. Lipshutz, S.P.A. Fodor, and T.R. Gingeras. High density synthetic oligonucleotide arrays. *Nature Genetics Supplement*, 21, 1999.

[20] A.V. Lukashin and R. Fuchs. Analysis of temporal gene expression profiles: Clustering by simulated annealing and determining the optimal number of clusters. *Bioinformatics*, 17(5):405–414, 2000.

[21] P. C Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Science, India*, 1936.

[22] U. Maulik and S. Bandyopadhyay. Performance evaluation of some clustering algorithms and validity indices. *IEEE Trans, Pattern analysis and machine intelligence*, 24, 2002.

[23] J. MaxQueen. Some methods for classification and analysis of multivariate observtions. *proceedings of the Fifth Berkeley Symposium onMathematical Statistics and Probability*, 1:281–297, 1967.

[24] N. Metripolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and H. Teller. Equation of state calculation by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[25] P Nurmi. Mixture models. Technical report, Department of Computer Science, University of Helsinki, Finland, 2004.

[26] C.M. Perou, T. Sorlie, M.B. Eisen, and Rijn M.V. et al. Molecular portraits of human breast tumor. *Nature*, 2000.

[27] J Quachenbush. Computational analysis of microarray data. *Nature Genetics*, 2:418–427, 2001.

[28] S. Ray and R.H. Turi. Determination of number of clusters in k-means clustering and application in colour image segmentation. In *4th International Conference on Advances in Pattern Recognition and Digital Techniques (ICAPRDT'99)*, 1999.

[29] M. Schena. *Microarray Analysis.* J. Wiley & Sons (New York), 2002.

[30] 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 Natl Acad Sci U S A*, 96, 1999.

[31] J. Vesanto and E. Alhoniemi. Clustering of the self-organizing map. *IEEE Trans, Neural Networks*, 1.11:586–600, 2000.

[32] M.E. Wall, P.A. Dyck, and T.S. Brettin. Svdman- singular value decomposition analysis of microarray data. *Bioinformatics*, 17:566–568, June 2001.

[33] X.Y. Wang, G. Whitwell, and J.M. Garibaldi. The application of a simulated annealing fuzzy clustering algorithm for cancer diagnosis. In *4th International Conference on Intelligent Systems Design and Application*, 2004.

[34] O. Wolkenhauer. Cluster analysys. Technical report, The University of Manchester, 2002.

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

VITA AUCTORIS

NAME: Wei Yang

PLACE OF BIRTH: Beijing, China

YEAR OF BIRTH: 1973

EDUCATION: South China University of Technology, GuangZhou, China
1992-1996

University of Windsor, Windsor, Ontario
2002-2005