AN EMPIRICAL CLUSTERING TECHNIQUE

A dissertation submitted towards the partial fulfillment of the
requirement for the Award of the Degree of

Master of Engineering
in
Computer Technology & Application

Submitted by
Nitasha Rathore
College Roll No: 14/CTA/04
University Roll no: 8508

Under the guidance of
Mrs. Rajni Jindal

DEPARTMENT OF COMPUTER ENGINEERING
DELHI COLLEGE OF ENGINEERING
UNIVERSITY OF DELHI
2004 – 2006

CERTIFICATE

This is to certify that the work that is being presented in this dissertation entitled ***"AN EMPIRICAL CLUSTERING TECHNIQUE"*** submitted by **Nitasha Rathore** in the partial fulfillment of the requirement for the award of the degree of **Master of Engineering** in Computer Technology and Application, Delhi College of Engineering is an account of her work carried out under my guidance and supervision.

Dr. Goldie Gabrani	**Mrs. Rajni Jindal**
Head of Department	Lecturer
Department of Computer Engineering	Department of Computer Engineering
Delhi College of Engineering	Delhi College of Engineering
Delhi	Delhi

ACKNOWLEDGEMENT

It is a great pleasure to have the opportunity to extent my heartiest felt gratitude to everybody who helped me throughout the course of this project.

It is distinct pleasure to express my deep sense of gratitude and indebtedness to my learned supervisor Mrs Rajni Jindal, for her invaluable guidance, encouragement and patient review. Her continuous inspiration only has made me complete this major project.

I would also like to take this opportunity to present my sincere regards to my teacher's viz. Prof. Goldie Gabrani, Dr S. K. Saxena, Mr. Rajeev Kumar and Prof. D. Roy Choudhury for their support and encouragement.

I would like to express my heartiest felt regards to Faculty of Computer Engineering Department for providing facility and for their co-operation and patience.

I am thankful to my friends and classmates for their unconditional support and motivation during this project.

Nitasha Rathore
M.E. (Computer Technology & Applications)
College Roll No. 14/CTA/04
University Roll No. 8508
Delhi College of Engg, Delhi

Table of Contents

List of figures

Abstract

Clustering is the unsupervised classification of patterns into groups (clusters). Clustering technique aims at identifying groups of similar objects and, therefore helps to discover distribution of patterns and interesting relations in large data sets. It has been subject of wide research since it arises in many application domains in engineering, business and social sciences. Especially, in the last few years the availability of huge transactional and experimental data sets and the arising requirements for data mining created needs for clustering algorithms that scale and can be applied in diverse domains. This thesis introduces the fundamental concepts of clustering while it surveys the widely known clustering algorithms.

In this thesis we have implemented different approaches to take advantage of clustering technique and performed an empirical study evaluating clustering results using gene-expression datasets. We have tested several different clustering algorithms and similarity measure combinations on the same datasets & evaluated the clustered datasets.

Keywords: clustering algorithms, unsupervised learning, genes, microarrays

Chapter 1
Introduction

1.1 Introduction

The biggest problem of today's Information society is over abundance of information on any one topic; this presents a challenge for the information professional as well as the user of information. Moreover, for many information users of different levels, the problem is a lack of appropriate information, namely, one which they can easily comprehend, assimilate and make use of with a certain amount of confidence and reliability within the framework of their own environment. Hence, the crux of the problem lies in the improper packaging of information not the overabundance; the information needs to be packaged in a form that should be useful to different groups of users. This only can ensure fruitful use of existing information. Thesis report is one such kind of information product.

Clustering as a subject has been making waves since the time of its evolution as a complementary technique to statistics and computer analysis and then its later development as a full-fledged application science. This thesis is an attempt to survey the evolution of clustering as a subject, the core concepts and techniques arising from clustering methodology and the important works done therein. It aims to provide a snapshot view of the rich history of clustering techniques and its development as a mature field that is making important contributions to application areas in different subjects.

1.2 Thesis Work

The thesis has been undertaken with the purpose of preparing a clustering technique. As a part of the thesis requirement, I have tried to compile this report as an extensive information source on clustering techniques and have concentrated on the most recent developments in the subject area specially document clustering.

1.2.1 Objectives of the Thesis:

- To visualize the structure and development of the clustering as a subject especially in the relation to other subjects.
- To provide a comprehensive document for background reading about clustering techniques.

- To alert the expert regarding the work that has been already done as well as the forthcoming ones in various areas of clustering. It should help in avoiding the duplication of the work and thereby conserve the intellectual's potential.

1.2.2 Purpose of the Thesis:

- To provide a ready-made reference to the developments of a discipline over a certain period of time, to the researchers and their professionals.
- To serve as a general source of information for the professionals already working in the area of clustering and to bring out the present trends, latest developments and subsequent applications, thereby aiding in the further research.

1.2.3 Methodology Adopted

Preparation of thesis and Report requires skills and knowledge to make the product an up-to-date source on the given topic, viz., the familiarity with the different aspects of a subject, the user requirements and also the knowledge of the helpful methods of presentation of ideas.

So, a review of relevant literature was conducted to assess clustering concepts, clustering technologies, techniques and researches in this field. Various books, journals, articles and papers available on the Internet pertaining to the subject matter and its related fields were reviewed to prepare report.

1.3 Steps of the Thesis and Report Preparation

The scope of the subject was determined in terms of a framework determined by studying the subject hierarchy.

1.3.1 Data collection:

The next step was data collection, for this the core journals containing articles on and related to clustering were identified. Secondary journals were also consulted along with primary journals. Numerous websites, portals, discussion forums, magazines, newsletters were also referred for data collection.

1.3.2 Analysis:

The collected data needs to be analyzed for making a comprehensive study of the evolutionary trends in the subject. For the purpose, each micro-document was analyzed to mark it with appropriate subject proposition. There were documents that dealt with more than one topic; in these cases multiple entries of document were made and appropriate subject proposition was assigned at the top of the entries.

1.3.3 Arrangement:

After analysis, each entry was arranged under different headings according to subject proposition. The advantage of this arrangement is that the entries on the same topic come together. Then each group of entries was checked to ascertain their appropriateness and their subject grouping within each part.

1.3.4 Consolidation:

This step entails taking the entries of one aspect under the part and then consolidation of the abstract into a readable text; here, the references are given using serial number of the entries.

1.3.5 Organization of the Thesis:

In this thesis, chapter 1 covers objective, purpose and various Steps of the thesis and Report preparation. Chapter 2 covers introduction of clustering, which includes Definition and Requirements and steps of clustering. Chapter 3 covers Clustering Techniques which includes Division and subdivision of clustering, Partition based clustering, Hierarchy based clustering, Self Organizing Maps, and Principal Component Analysis. Chapter 4 covers Implementation Issues which includes Distance function, Calculating the distance between clusters, Finding the cluster centroid and choosing the distance measure. Chapter 5 covers Applications of clustering in Biology and Marketing research. Chapter 6 covers experimental results. Chapter 7 covers Conclusions.

Chapter 2
Introduction to Clustering

2.1 Introduction to Clustering

With the increasing demand for rich, deep, digitized content, the volume of data required by organizations has become overwhelming, and has almost surpassed current capabilities of most database technologies. Now with the increased prevalence of Internet and widespread development of additional methods of content-delivery, the volume of information is increasing in explosive proportions and presenting challenges to the information professionals for proper packaging and delivery of significant information.

Clustering has been used since long for grouping together entities with similar traits and classifying objects but in present context it has acquired new dimensions as a solution to the chaos that we have on Internet, voluminous databases, and information repositories. The reason for its increased significance is that it relies on finding natural groups in the existing data rather than classifying them on the basis of some externally imposed artificial criteria.

Clustering techniques are applied only when there is no class to be predicted, rather when we need to divide the instances into natural groups. These clusters presumably reflect some mechanism at work in the domain from which instances are drawn; the mechanism causes some instances to bear a stronger resemblance to one another than they do to the remaining instances. However, clustering requires different techniques to the classification and association learning methods.

There are two straightforward ways how gene expression matrices can be studied [1]

- Comparing expression profiles of genes by comparing rows in the expression matrix and
- Comparing expression profiles of experiments by comparing columns in the matrix.

Additionally, if the data normalization allows it, combinations of both are possible. We can look either on similarities or differences. If two genes (rows) are similarly expressed, we can hypothesize that the respective genes are co-regulated and possibly functionally related

(Guild by Association) [2]. The comparison of experiments can provide us with the information, which genes are differentially expressed in two conditions and this enables us to study, for example, effects various compounds have on an investigated condition.

Clustering can be defined as the process of separating a set of objects into several subsets on the basis of their similarity [8]. The aim is generally to define clusters that minimize intracluster variability while maximizing intercluster distances, i.e. finding clusters, which members are similar to each other, but distant to members of other clusters in terms of gene expression based on the used similarity measurement. Two clustering strategies are possible: supervised (based on existing knowledge) or unsupervised.

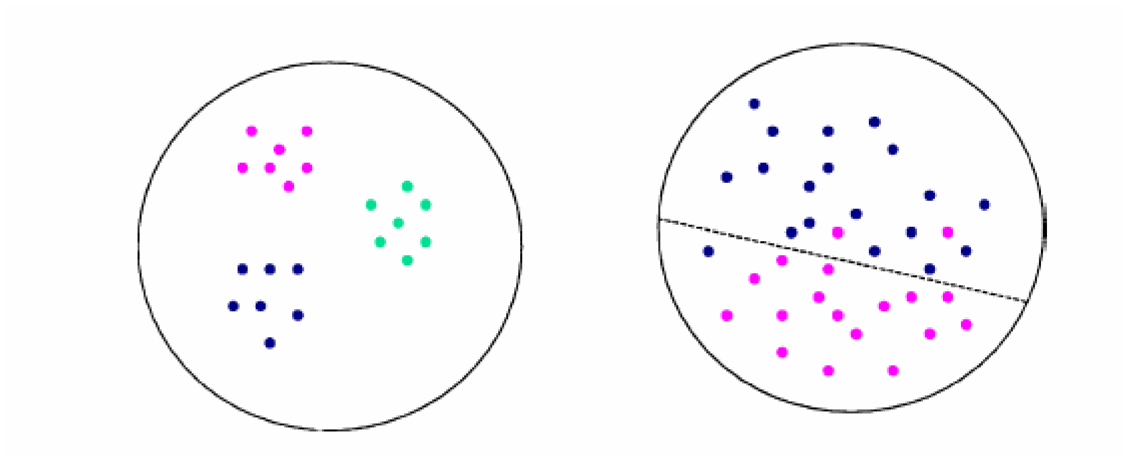


Figure1: Supervised and unsupervised data analysis.

In the unsupervised case (left) we are given data points in n-dimensional space (n=2 in the example) and we are trying to find ways how to group together points with similar features. For instance, there are three natural clusters in the example, each consisting of data points close to each other in a sense of Euclidean distance. A clustering algorithm should identify all these clusters. In the supervised case (right), the objects are labeled (e.g. we have magenta and blue points in the example), and the task is to find a set of classification rules allowing us to discriminate between these points as precisely as possible. For instance, dotted line in the drawing [1].

2.2 Definition and Scope

2.2.1 Definition:

In general, the word clustering is almost synonymous with classification. But literally, clustering is the grouping of similar objects. Such classification occurs constantly in thought and speech. Objects that differ in insignificant details are given the same name, can be treated the same, and can be expected to act the same. For example, a wife notices that the man coming in the door differs only in insignificant details fro her husband that left in the morning, and so she expects him to answer to the same name.

Clustering is a nonlinear activity that generates ideas, images and feelings around a stimulus word. As students cluster, their thoughts tumble out, enlarging their word bank for writing and often enabling them to see patterns in their ideas. Clustering may be a class or an individual activity.

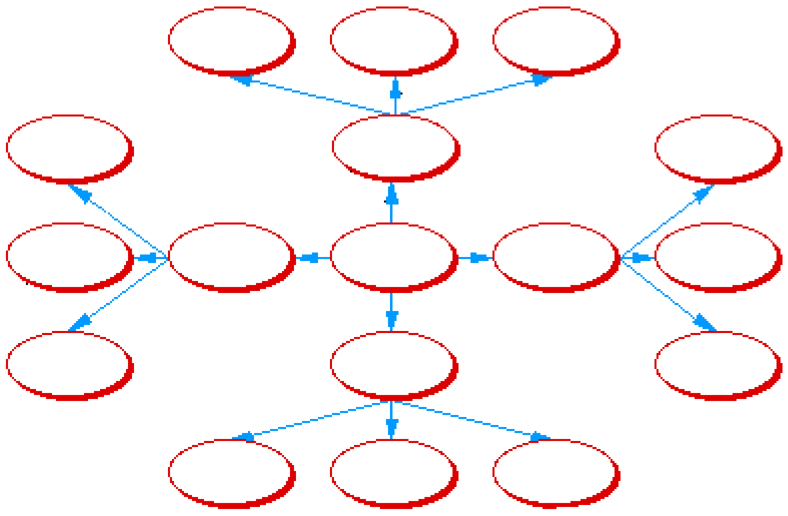


Figure2: A diagrammatic representation of clustering of similar ideas/objects into different nodes.

Other way to define is that clustering algorithms find groups of items that are similar. To explain the above diagram, clustering could be used by an insurance company to group customers into different nodes according to income, age, types of policies purchased and prior claims experience. The above diagram divides a data set so that records with similar content are in the same group, and groups are as different as possible from each other. Since the categories are unspecified, this is sometimes referred to as unsupervised learning [3].

Further, it may also be defined as the technique of grouping records together based on their locality and connectivity within the n-dimensional space. This is an unsupervised learning technique [4].

So, clustering is the process of grouping a set of physical or abstract objects into classes of similar objects. A cluster is a collection of data objects that are similar to one another with the same cluster and are dissimilar to the objects in other clusters. A cluster of data objects can be treated collectively as one group in many applications.

Clustering is a form of learning by observation rather than learning by examples. Cluster analysis is an important human activity in which we indulge since childhood when we learn to distinguish between animals and plants etc by continuously improving subconscious clustering schemes. This has been widely used in numerous applications, including pattern recognition, data analysis, image processing, and market research etc.

2.2.2 Scope

Clustering is a very important application area but widely interdisciplinary in nature, that makes it very difficult to define its scope. It is used in several research communities to describe methods for grouping of unlabeled data, now, these communities have different terminologies and assumptions for the components of the clustering process and the contexts in which clustering is used. The area of clustering can be comprehended in the following way:

Cluster analysis has been studied extensively for years, focusing mainly on distance-based cluster analysis. Many clustering tools were made based on k-means, k-medoids, and some of the methods were incorporated in many statistical analysis software a [packages or systems like S-plus, SPSS, and SAS. In machine learning, clustering is termed as an example of unsupervised learning. These do not rely on predefined classes and class-labeled training examples unlike classification.

The present active themes of research in this area focus on the scalability of clustering methods, the effectiveness of methods for clustering complex shapes and types of data, high-dimensional clustering techniques, and methods for clustering mixed numerical and categorical data in large databases.

However, clustering, as a subject is still vulnerable on two fronts, i.e., the classifications delivered are not sufficiently compelling to convince the experts always, they believe that detailed knowledge is more important than fancy manipulation; and the second is the techniques themselves are not based on sound probability model and the results many times are poorly evaluated or turn unstable when evaluated.

2.3 Requirements and Steps of Clustering

2.3.1 Basic Requirements for Clustering are following [5]:

Scalability:

Many clustering algorithms work well on small data sets that contain less than 200 data objects; however a large database may contain millions of objects. In that case clustering on a sample of a given large data set may lead to biased results; and highly scalable clustering algorithms are needed for the purpose.

Ability to deal with different types of attributes:

Many algorithms are designed to cluster interval-based (numerical) data. However, applications may require clustering other types of data, viz. binary, categorical (nominal), and ordinal data, or mixtures of these data types.

Discovery of clusters with arbitrary shape:

Many clustering algorithms determine clusters based on Euclidean or Manhattan distance measures. Algorithms based on such distance measures tend to find spherical clusters with similar size and density. However, a cluster could be of any shape, it is important to develop algorithms that can detect clusters of arbitrary shape.

Minimal requirements for domain knowledge to determine input parameters:

Many clustering algorithms require users to input certain parameters in cluster analysis (such as the number of desired cluster). The clustering results can be quite sensitive to input parameters. Parameters are often hard to determine, especially for data sets containing high-dimensional objects. This not only burdens users but also makes the quality of clustering difficult to control.

Ability to deal with noisy data:

Most real-world databases contain outliers or missing, unknown or erroneous data. Some clustering algorithms are sensitive to such data and may lead to clusters of poor quality.

Insensitivity to the order of input records:

Some clustering algorithms are sensitive to the order of input data; fro example, the same set of data, when presented with different orderings to such an algorithm, may generate dramatically different clusters. It is important to develop algorithms that are insensitive to eh order of input.

High dimensionality:

A database or data warehouse can contain several dimensions or attributes. Many clustering algorithms are good at handling low-dimensional data, involving only two to three dimensions. Human eyes are good at judging the equality of clustering for up to three dimensions. It is challenging to cluster data objects in high-dimensional space, especially considering that such data can be very sparse and highly skewed.

Constraint –based clustering:

Real world applications may need to perform clustering under various kinds of constraints. If one has to choose the locations for a given number of new automatic cash-dispensing machines in a city hen to decide upon this we may cluster households while considering constraints such as the city's rivers and highway networks, and customer requirements per region. A challenging task is to find groups of data with good clustering behavior that satisfy specified constraints.

Interoperability and usability:

Users expect clustering results to be interpretable comprehensible, and usable. That is, clustering may need to be tied up with specific semantic interpretations and applications. It is important to study how an application goal may influence the selection of clustering methods.

2.3.2 Clustering steps [6]:

Preprocessing and feature selection

Most clustering models assume that all data items are represented by n-dimensional feature vectors. This first step therefore involves choosing appropriate features, and doing appropriate preprocessing and feature extraction on data items to measure the values of the chosen feature set. It will often be desirable to choose a subset of all the features available, to reduce the dimensionality of the problem space. This step often requires a good deal of domain knowledge and data analysis.

Similarity measure

This is a function, which takes two sets of data items as input, and returns as output a similarity measure between them. Item-set versions use any item-item version as subroutines and include max / min / average distance; another approach looks at the distance from the item to the distance to the cluster representative of the set, where point representatives are chosen as the mean vector / mean center / median center of the set, and hyperplane or hyperspherical representatives of the set can also be used. Set-set versions include max / min average distance, as well as item-item versions applied to the two set representatives.

Clustering algorithm

Clustering algorithms are general schemes which use particular similarity measures as subroutines. The particular choice of clustering algorithms depends on the desired properties of the final clustering, e.g. what are the relative importances of compactness, parsimony, and inclusiveness? Other considerations include the usual time and space complexity.

Result interpretation and application

Typical applications of clustering include data compression (via representing data samples by their cluster representative), hypothesis generation (looking for patterns in the clustering of data), hypothesis testing (e.g. verifying feature correlation or other data properties through a high degree of cluster formation), and prediction (once clusters have been formed from data and characterized, new data items can be classified by the characteristics of the cluster to which they would belong).

Motivation

Data analysis underlies many computing applications, either in a design phase or as part of their on-line operations. Data analysis procedures can be dichotomized as either exploratory or confirmatory, based on the availability of appropriate models for the data source, but a key element in both types of procedures (whether for hypothesis formation or decision-making) is the grouping, or classification of measurements based on either (i) goodness-of-fit to a postulated model, or (ii) natural groupings (clustering) revealed through analysis. Cluster analysis is the organization of a collection of patterns (usually represented as a vector of measurements, or a point in a multidimensional space) into clusters based on similarity.

The variety of techniques for representing data, measuring proximity (similarity) between data elements, and grouping data elements has produced a rich and often confusing assortment of clustering methods.

It is important to understand the difference between clustering (unsupervised classification) and discriminate analysis (supervised classification). In supervised classification, we are provided with a collection of labeled (reclassified) patterns; the problem is to label a newly encountered, yet unlabeled, pattern. Typically, the given labeled (training) patterns are used to learn the descriptions of classes which in turn are used to label a new pattern. In the case of clustering, the problem is to group a given collection of unlabeled patterns into meaningful clusters. In a sense, labels are associated with clusters also, but these category labels are data driven; that is, they are obtained solely from the data.

Clustering is useful in several exploratory pattern-analysis, grouping, decision-making, and machine-learning situations, including data mining, document retrieval, image segmentation, and pattern classification. However, in many such problems, there is little prior information (e.g., statistical models) available about the data, and the decision-maker must make as few assumptions about the data as possible. It is under these restrictions that clustering methodology is particularly appropriate for the exploration of interrelationships among the data points to make an assessment (perhaps preliminary) of their structure. The term "clustering" is used in several research communities to describe

Methods for grouping of unlabeled data:

These communities have different terminologies and assumptions for the components of the clustering process and the contexts in which clustering are used. Thus, we face a dilemma regarding the scope of this survey. The production of a truly comprehensive survey would be a monumental task given the sheer mass of literature in this area. The accessibility of the survey might also be questionable given the need to reconcile very different vocabularies and assumptions regarding clustering in the various communities.

The goal is to survey the core concepts and techniques in the large subset of cluster analysis with its roots in statistics and decision theory. Where appropriate, references will be made to key concepts and techniques arising from clustering methodology in the machine-learning and other communities.

The audience for this paper includes practitioners in the pattern recognition and image analysis communities (who should view it as a summarization of current practice), practitioners in the machine-learning communities (who should view it as a snapshot of a closely related field with a rich history of well understood techniques), and the broader audience of scientific professionals (who should view it as an accessible introduction to a mature field that is making important contributions to computing application areas).

2.3.3 Microarrays

Microarray technology [7] is one very promising approach for high throughput analysis and gives the opportunity to study gene expression patterns on a genomic scale. It all began about a quarter century ago, with Ed Southern's key insight that labeled nucleic acid molecules could be used to interrogate nucleic acid molecules attached to a solid support. Today, thousands or even tens of thousands of genes can be spotted on a microscope slide. Applied to functional genetics and mutation screening, microarrays give us the opportunity to determine thousands of expression values in hundreds of different conditions, allowing the contemplation of genetic processes on a whole genomic scale to determine genetic contributions to complex polygenic disorders and to screen for important changes in potential disease genes.

Because of the vast amount of data produced by a microarray experiment, sophisticated software tools are used to normalize and analyze the data. First the scanned images are analyzed using image analysis software, which evaluates the expression of a gene.
The next step is to extract the fundamental patterns of gene expression inherent in the data in a mathematical process called **clustering**, which organizes the genes into biological relevant clusters with similar expression patterns genes.

The interest is in how gene expression is changed by various diseases or compound treatments, respectively. For example one can investigate the differences in gene expression between a normal and a cancer cell. Several clustering techniques were recently developed and applied to analyze microarray data [8]. However, to the best of my knowledge, there is no single tool, which integrates the common clustering methods. Such a tool would be valuable for comparison and evaluation of clustering algorithms and their result and would help biologists to gain biological meaningful information out of microarray datasets in a less costly way.

Chapter 3
Clustering Techniques

3.1 Divisions and Subdivisions of Clustering

Clustering is an applied science and very methodical in approach, so, its divisions and subdivisions can't be defined in as concrete manner as is the case with other conventional subjects.

However, the different methods that are applied for clustering and further analysis are used as a basis to define its subdivisions.

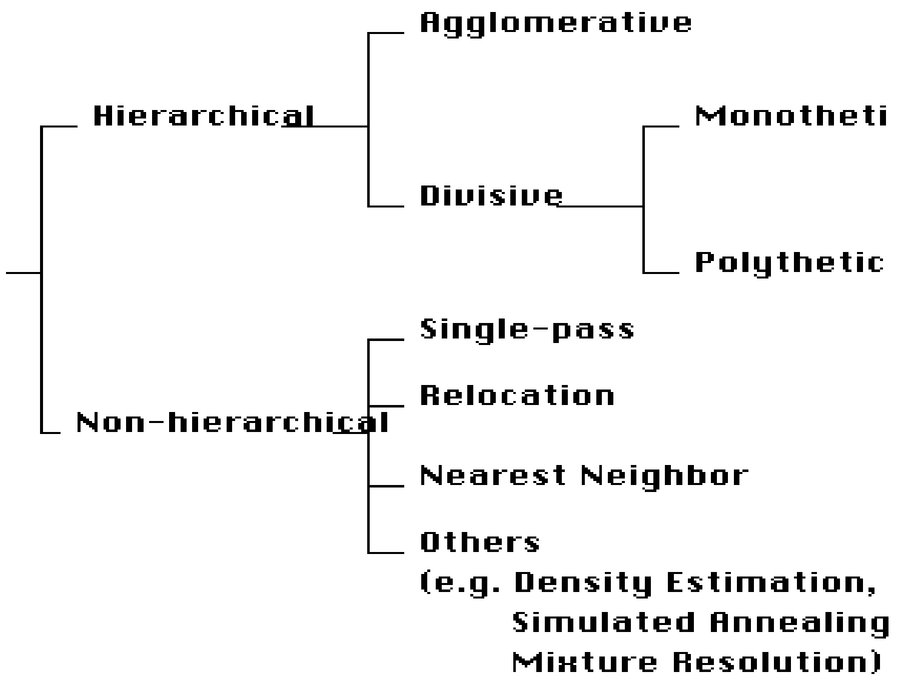


Figure3: Division and subdivision of clustering

To explain the above diagram, there were broadly two main subdivisions i.e., hierarchical and non-hierarchical but in due course of time new subdivisions evolved from the above divisions like density based methods, grid-based methods, model-based methods etc. These subdivisions were ramifications of above main divisions only but they developed to become independent divisions of the subject gradually [9].

3.2 Partition based clustering:

3.2.1 Introduction

K-means [20-23] is a commonly used clustering method because it is based on a very simple principle and provides good results. It is very similar to SOM, unsupervised, and can be seen as a Bayesian (maximum likelihood) approach to clustering.

The basic idea is to maintain two estimates:

1. An estimate of the center location for each cluster and
2. A separate estimate of the partition of the data points according to which one goes into which cluster.

One estimate can be used to refer the other. If we have an estimate of the center locations, then (with reasonable prior assumptions) the maximum likelihood solution is that each data point should belong to the cluster with the nearest center. Hence, we can compute a new partition: from a set of center locations.

It constructs k partitions of the data given a database on n objects or data tuples; here each partition represents a cluster and k<= n. so, it classifies the data into k groups, which together satisfy the following requirements: Each group must contain at least one object, and Each object must belong to exactly one group The general criterion of a good partition is that objects in the same cluster are "close" or related to each other, whereas objects of different clusters are "far apart" or very different. There are various other criteria for judging the quality of partitions. Partition-based clustering requires the exhaustive enumeration of all of the possible partitions to achieve global optimality but the clustering applications generally adopt the popular heuristic methods. These are further divided to following types:

3.2.2 k-Means Method: Centroid-Based Technique

It takes the input parameter, k, and partitions a set of n objects into k clusters so that the resulting intracluster similarity is high but the intercluster similarity is low. Cluster similarity is measured in regard to the mean value of the objects in a cluster, which can be viewed as the cluster's center of gravity.

But, this works well only when the mean of a cluster is defined which may not be the case in some applications, such as when data with categorical attributes are involved. So, the necessity for users to specify k, the number of clusters, in advance is actually a disadvantage.

The k-means method was not suitable for discovering clusters with non-convex shapes or clusters of very different size. Moreover, it is sensitive to noise and outlier data points since a small number of such data can substantially influence the mean value.

Here is step by step k means clustering with flow chart:

Step 1 Begin with a decision on the value of k = number of clusters

Step 2 Put any initial partition that classifies the data into k clusters. You may assign the training samples randomly, or systematically as the following:

1. Take the first k training sample as single-element clusters
2. Assign each of the remaining training samples to the cluster with the nearest centroid. After each assignment, recomputed the centroid of the gaining cluster.

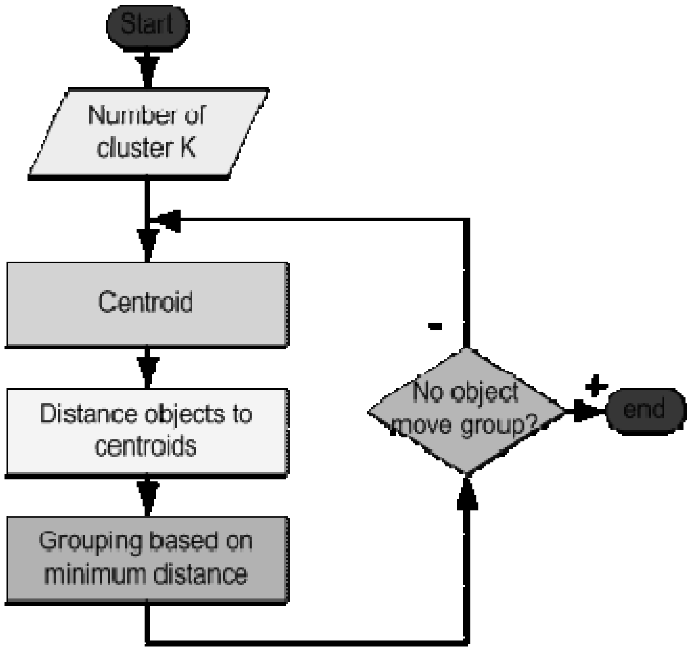

Start
Number of cluster K
Centroid
Distance objects to centroids
Grouping based on minimum distance
No object move group?
-
+
end

Figure 4: Flow chart of k-mean

Step 3 Take each sample in sequence and compute its distance from the centroid of each of the clusters. If a sample is not currently in the cluster with the closest centroid, switch this sample to that cluster and update the centroid of the cluster gaining the new sample and the cluster losing the sample.

Step 4 Repeat step 3 until convergence is achieved, that is until a pass through the training sample causes no new assignments.

3.2.3 K-means Clustering Algorithm

The essence of the k-means clustering algorithm is now to minimize the cost function of all clusters by executing the following steps:

1. Put each vector Xi of X in one of the k clusters.
2. Calculate the mean for each of the k clusters.
3. Calculate the distance between an object and the mean of a cluster.
4. Allocate an object to the cluster whose mean is the nearest to the object.
5. Re-calculate the mean of the clusters affected by the reallocation.
6. Repeatedly perform the operations (3) to (5) until no more reallocations occur.

If the number of data is less than the number of cluster then we assign each data as the centroid of the cluster. Each centroid will have a cluster number. If the number of data is bigger than the number of cluster, for each data, we calculate the distance to all centroid and get the minimum distance. This data is said belong to the cluster that has minimum distance from this data.

Since we are not sure about the location of the centroid, we need to adjust the centroid location based on the current updated data. Then we assign all the data to this new centroid. This process is repeated until no data is moving to another cluster anymore. Mathematically this loop can be proved to be convergent. The convergence will always occur if the following condition satisfied:

1. Each switch in step 2 the sum of distances from each training sample to that training sample's group centroid is decreased.
2. There are only finitely many partitions of the training examples into k clusters.

3.2.4 Properties

The k-means algorithm has the following important properties:

It is efficient in processing large data sets, due to the fact that the computational complexity of the algorithm is O(tkmn), where n is the number of vectors in X, m is the dimension of the vectors Xi, k is the number of clusters and t is the number of iterations over the whole data set. Usually, k, m, t« n It takes usually just a few seconds to calculate even datasets with 10000 elements and more, making it a valuable tool for the investigation of datasets that are too big for hierarchical clustering for instance.

Another big advantage is the moderate memory requirement for k-means clustering. Since there is no similarity matrix to calculate the memory requirements rise with O(n). The clusters have convex shapes. Therefore, it is difficult to use the k-means algorithm to discover clusters with non-convex shapes. It often terminates at a local optimum. To find the global optimum, techniques such as deterministic annealing and generic algorithms can be incorporated with the k-means algorithm.

The major drawback of the k-means algorithm is that the number of clusters has to be specified in advance. However, this is the only input needed for the clustering besides an abortion criterion to prevent infinite calculation, like an input for the maximum number of iterations to compute.

An additional advantage of k-means is the possibility to create fuzzy clusters, where one vector can belong to more than one cluster. This is a better model for the regulatory system that controls gene expression in a cell. One gene affects more than one other gene, i.e. it can be part of many different pathways and therefore has to belong to different clusters or no one.

3.3 Hierarchy Based Clustering

3.3.1 Introduction

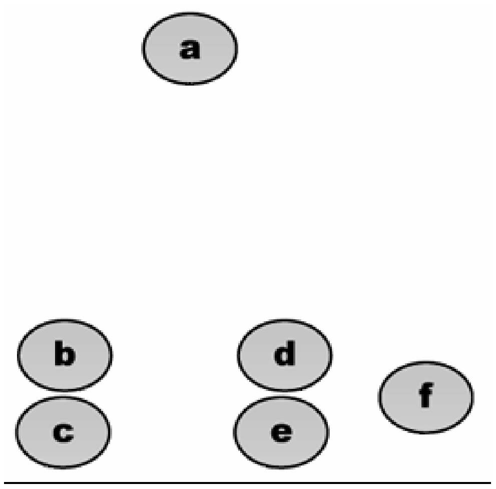
Hierarchical clustering [12-14] is an unsupervised procedure of transforming a distance matrix, which is a result of pair wise similarity measurement between elements of a group, into a hierarchy of nested partitions. The hierarchy can be represented with a tree-like dendrogram in which each cluster is nested into the next cluster. Hierarchical algorithms can be further categorized into two kinds:

3.3.2 Two Basic Types of Hierarchical Clustering

There are two types of hierarchical clustering - agglomerative and divisive. Agglomerative clustering takes each entity (i.e. gene) as a single cluster to start off with and then builds bigger and bigger clusters by grouping similar entities together until the entire dataset is encapsulated into one final cluster. Divisive hierarchical clustering works the opposite way around - the entire dataset is first considered to be one cluster and is then broken down into smaller and smaller subsets until each subset consists of only a single entity.

Only the agglomerative method will be explained in detail in this tutorial since it is the most commonly used in microarray analyses. The reasons for this is mainly computational - divisive clustering is more computationally expensive when it comes to making decisions in dividing a cluster in two given all possible choices. On the other hand, the divisive approach retains the 'super-structure' (i.e. the overall structure) of the data: what this means is that one can confidently say that the root or 'upper' levels of the dendrogram are highly representative of the original structure of the data. This does not mean to say that the agglomerative approach is not as robust, however.

(1) **Agglomerative procedures**: This procedure starts with *n* clusters (each object forms a cluster containing only itself) and iteratively reduces the number of clusters by merging the two most similar objects or clusters, respectively, until only one cluster is remaining.
($n \rightarrow 1$).
It starts by placing each object in its own cluster and then merges these atomic clusters into larger and larger clusters, until all to the objects are in a single cluster or until certain termination conditions are satisfied. Most hierarchical clustering methods belong to this category; they differ only in their definition of intercluster similarity.
For example, suppose this data is to be clustered. Where euclidean distance is the distance metric.



Raw data

The hierarchical clustering dendrogram would be as such:

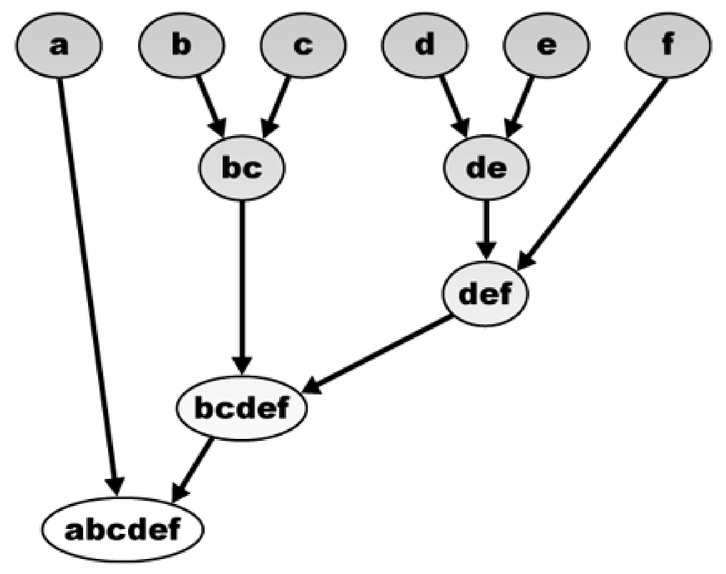


Figure 5: hierarchical clustering dendrogram

Traditional Representation

This method builds the hierarchy from the individual elements by progressively merging clusters. Again, we have six elements {a} {b} {c} {d} {e} and {f}. The first step is to determine which elements to merge in a cluster. Usually, we want to take the two closest elements, therefore we must define a distance d(element1,element2) between elements.

Suppose we have merged the two closest elements b and c, we now have the following clusters {a}, {b, c}, {d}, {e} and {f}, and want to merge them further. But to do that, we need to take the distance between {a} and {b c}, and therefore define the distance between two clusters. Usually the distance between two clusters $\mathcal{A}$ and $\mathcal{B}$ is one of the following:

The maximum distance between elements of each cluster (also called complete linkage clustering):

$$\max\{\, d(x,y) : x \in \mathcal{A},\, y \in \mathcal{B}\,\}$$

The minimum distance between elements of each cluster (also called single linkage clustering):

$$\min\{\, d(x,y) : x \in \mathcal{A},\, y \in \mathcal{B}\,\}$$

the mean distance between elements of each cluster (also called average linkage clustering):

$$\frac{1}{\mathrm{card}(\mathcal{A})\mathrm{card}(\mathcal{B})}\sum_{x\in\mathcal{A}}\sum_{y\in\mathcal{B}} d(x,y)$$

The sum of all intra-cluster variance

The increase in variance for the cluster being merged (Ward's criterion)

Each agglomeration occurs at a greater distance between clusters than the previous agglomeration, and one can decide to stop clustering either when the clusters are too far apart to be merged (distance criterion) or when there is a sufficiently small number of clusters (number criterion).

(2) **Divisive procedures**: This procedure starts with 1 cluster and iteratively splits a cluster, so that the heterogeneity is reduced as far as possible (1 $\rightarrow$ *n).* If it is possible to find a reasonable distance definition between clusters, agglomerative procedures are less computationally expensive than divisive procedures, since in one step two out of maximum *n* elements have to be chosen for merging, whereas in divisive procedures, fundamentally all subsets have to be analyzed so that divisive procedures have an algorithmic complexity in the magnitude of $O(2^n)$. Agglomerative procedures have the drawback that an incorrect merging of clusters in an early stage often yields results, which are far away from the real cluster structure. Divisive procedures immediately start with interesting cluster arrangements and are therefore much more robust. Usually agglomerative procedures are used because of their efficiency.

It does the reverse of agglomerative clustering; starts with all objects in one cluster. Then, it subdivides the cluster into smaller and smaller pieces, until each object forms a cluster on its won or until it satisfies certain termination conditions, such as a desired number of clusters obtained or the distance between the two closest clusters is above a certain threshold distance.

3.3.3 Hierarchical clustering Algorithm

The procedures of agglomerative hierarchical clustering execute the following basic steps:

1. Calculate the distance between all objects and construct the similarity distance matrix. Each object represents one cluster, containing only itself.
2. Find the two clusters rand s with the minimum distance to each other.
3. Merge the clusters r and s and replace r with the new cluster. Delete s and recalculate all distances, which have been affected by the merge.
4. Repeat step (2) and (3) until the total number of clusters become one.

3.3.4 Amalgamation or linkage rules

At the first step, when each object represents its own cluster, the distances between those objects are defined by the chosen distance measure. However, once several objects have been linked together, a linkage or amalgamation rule is needed to determine if two clusters are sufficiently similar to be linked together. There are numerous linkage rules that have been proposed. Here are some of the most commonly used:

3.3.4.1 Single linkage (Minimum or nearest neighbor Method)

In this method the distance between two clusters is determined by the distance of the two closest objects (nearest neighbors) in the different clusters. If there are several equal minimum distances between clusters during merging, single linkage is the only well defined procedure. Its greatest drawback is the tendency for chain building: Only one (random) small distance is enough to enforce the amalgamation of two otherwise very different clusters. Therefore, the resulting clusters tend to represent long "chains”. The dissimilarity between 2 clusters is the **minimum dissimilarity** between members of the two clusters. This method produces long chains which form loose, straggly clusters. This method has been widely used in numerical taxonomy.

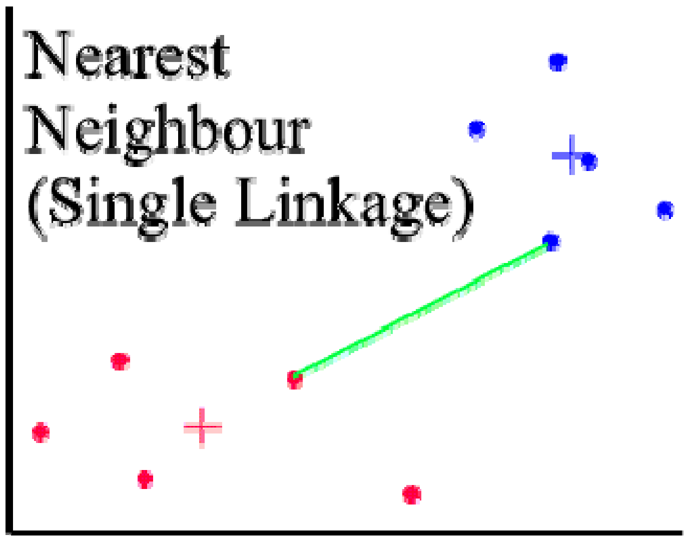


Figure 6: nearest neighbor linkage graph

3.3.4.2 Complete linkage (Maximum or furthest neighbor Method)

In this method, the distances between clusters are determined by the greatest distance between any two objects in the different clusters (i.e., by the "furthest neighbors"). Complete linkage usually performs quite well in cases when the objects actually form naturally distinct data clouds in the multidimensional space. If the clusters tend to be somehow elongated or of a "chain" type nature, then this method is inappropriate. Since only one (random) large distance is enough to pretend two clusters from merging, clusters tend to be small and merged together very late with a great error value. The dissimilarity between 2 groups is equal to the **greatest dissimilarity** between a member of cluster *i* and a member of cluster *j*. This method tends to produce very tight clusters of similar cases.

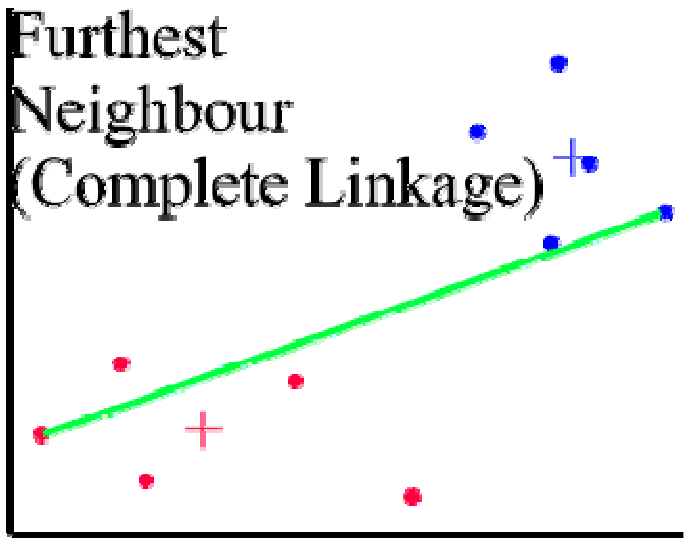


Figure 7: furthest neighbor linkage graph

3.3.4.3 Average linkage method

In this method, the distance between two clusters is calculated as the average distance between all pairs of objects in the two different clusters. This method is very efficient when the objects form natural distinct "clumps," however, it performs equally well with elongated, "chain" type clusters. Since the distance between two clusters lies between the minimum formation of single linkage and the maximum formation of complete linkage this procedure empirically shows no tendencies to either extreme described above, and is therefore more stable to unknown data point distributions. Admittedly, if there are several equal distances, the sequence of the amalgamation is critical. Note that the abbreviation UPGMA is used as well to refer to this method as *unweighted pair-group method using arithmetic averages.*

This method is identical to the unweighted pair-group average method, except that in the computations, the size of the respective clusters (i.e., the number of objects contained in them) is used as a weight. Thus, this method (rather than the previous method) should be used when the cluster sizes are suspected to be greatly uneven.

The dissimilarity between clusters is calculated using **average** values. Unfortunately, there are many ways of calculating an average. The most common (and recommended if there is no reason for using other methods) is UPGMA - **Unweighted Pair-Groups Method Average**. The average distance is calculated from the distance between each point in a cluster and all other points in another cluster. The two clusters with the lowest average distance are joined together to form the new cluster.

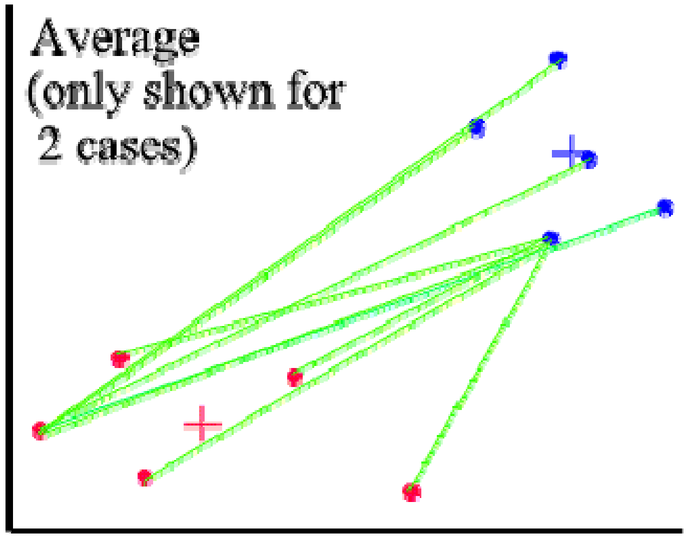


Figure 7: Average linkage graph

The + signs mark the centers of the two clusters.

3.3.4.4 Centroid linkage method

The centroid of a cluster is the average point in the multidimensional space defined by the dimensions. In a sense, it is the center of gravity for the respective cluster. In this method, the distance between two clusters is determined as the difference between centroids.

This method is identical to the previous one, except that weighting is introduced into computation to take into consideration differences in cluster sizes. There are other methods based on CENTROID and MEDIAN averages. Centroid, or UPGMC (Unweighted Pair-Groups Method Centroid), uses the group centroid as the average. The centroid is defined as the centre of a cloud of points. A problem with the centroid method is that some switching and reversal may take place, for example as the agglomeration proceeds some cases may need to be switched from their original clusters. This makes interpretation of the dendrogram quite difficult.

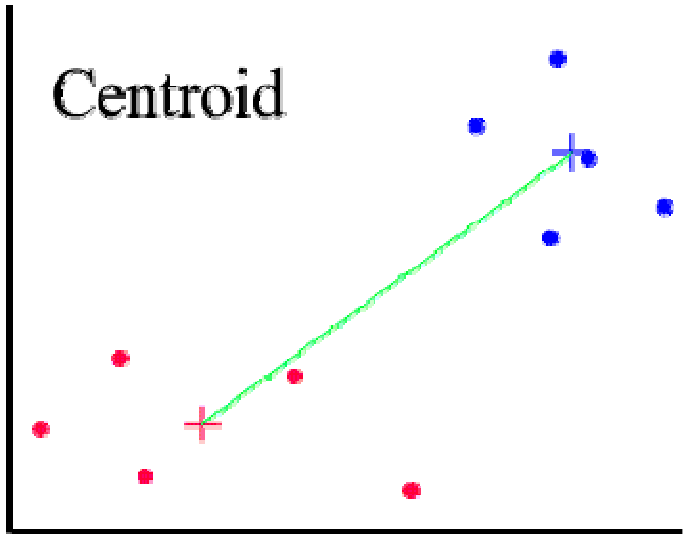


Figure 9: centroid linkage graph

The + signs mark the centres of the two clusters.

3.3.5 Properties

Hierarchical clustering is the most commonly used clustering strategy for gene expression analysis at the moment. The biggest advantage is that aside: from a choice of the amalgamation rule and the type of similarity distance measurement, no further parameters have to be specified. The result is a reordered set of genes and/or experiments, where similar vectors are close to each other in the tree structure and the distance between vectors and clusters is encoded in the branch length of a sub tree. This not only allows estimation of the similarity of neighboring genes, but also of the distance between distant vectors. This is helpful if someone is more interested in distances rather than similarities between two or more investigated conditions.

Hierarchical clustering just rearranges the dataset to a new, better ordered set of data vectors, therefore clusters have to be specified by the user by selecting a subtree as a cluster. A second drawback is the computational complexity. Large datasets are difficult or impossible to calculate due to the vast amount of necessary memory for the similarity matrix and the calculation time needed. Datasets with more than 20.000 vectors are manageable just by very advanced computer hardware.

The software discussed in this paper is able to calculate Single Linkage, Complete Linkage and Unweighted Average Linkage Clustering on both the genes and the experiments.

3.4 Self Organizing Maps

3.4.1 Introduction

One of the most popular neural network models today is the principle of a Self-Organizing Map (SOM)[15-19], developed by professor Kohonen at the University of Helsinki. A SOM is basically a multidimensional scaling method, which thesis data from input space to a lower dimensional output space. The SOM algorithm is based on unsupervised competitive learning, which means that the training is entirely data-driven and needs no further information.

A SOM is formed of neurons located in a regular, usually 1- or 2-dimensional grid. Each neuron *i* of the SOM is represented by an n-dimensional weight or reference vector. The neurons of the map are connected to adjacent neurons by a neighborhood relation dictating the structure of the map. Usually the map topology is rectangle or hexagonal. The number of neurons determines the granularity of the resulting mapping, which affects the accuracy and the generalization capability of the SOM.

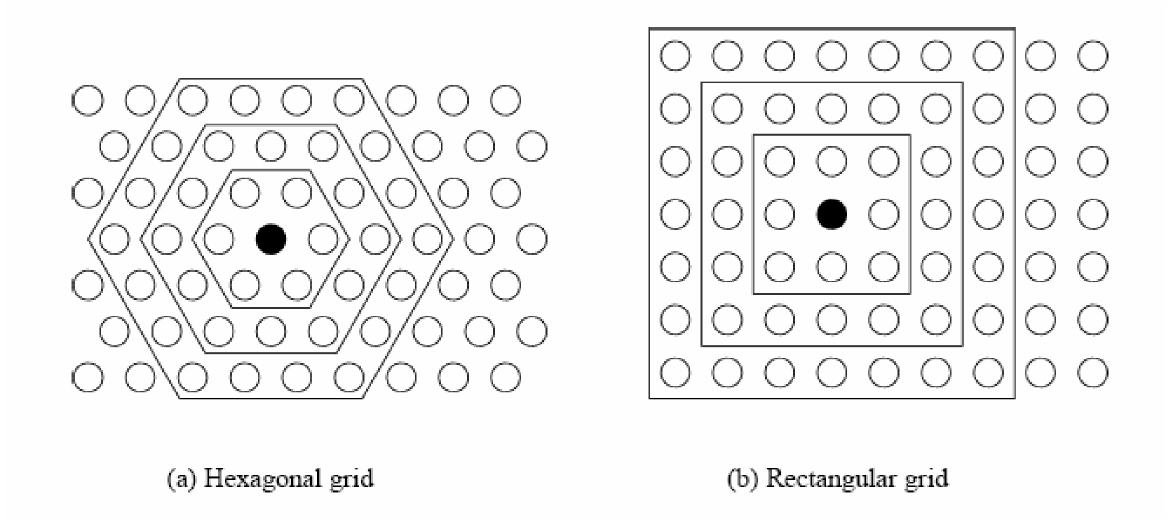


***Figure 10**: In the 2-dimensional case the neurons of the map can be arranged either on a rectangular or hexagonal lattice. Neighborhoods (size* **1**, **2** *and* **3**) *of the unit marked with black dot.*

3.4.2 Initialization

In the basic SOM algorithm, the topological relations and the number of neurons are fixed from the beginning. The number of neurons should be the number of clusters expected, with the neighborhood size controlling the smoothness and generalization of the mapping. Before the training phase, initial values are given to a weight vector, defined for each neuron. The SOM is robust regarding the initialization, but properly accomplished, it allows the algorithm to converge faster to a better solution. The two following initialization procedures are used:

- Random initialization, where the weight vectors are initialized with small random values between the minimum and maximum values of the vector.
- Random Gene initialization, where the weight vectors are initialized with random sample vectors from the training dataset.

3.4.3 SOM - Architecture

Lattice of neurons ('nodes') accepts and responds to set of input signals. Responses compared; 'winning' neuron selected from lattice. Selected neuron activated together with 'neighbourhood' neurons. Adaptive process changes weights to more closely resemble inputs

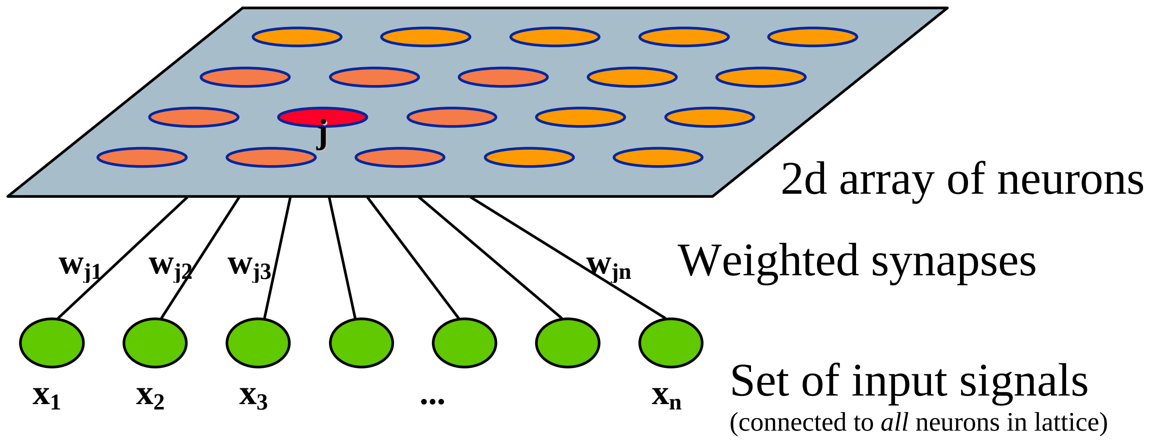


Figure 11: SOM-Lattice of neurons

3.4.4 Self Organizing Maps Algorithm:

1. Randomly initialise all weights
2. Select input vector **x** = [x1, x2, x3, … , xn]
3. Compare **x** with weights **w**j for each neuron j to determine winner
4. Update winner so that it becomes more like **x**, together with the winner's *neighbours*
5. Adjust parameters: learning rate & 'neighbourhood function'

6. Repeat from (2) until the map has converged (i.e. no noticeable changes in the weights) or pre-defined no. of training cycles have passed.

Initialisation

(i) Randomly initialise the weight vectors **w**j for all nodes j:

Input vector

(ii) Choose an input vector **x** from the training set:

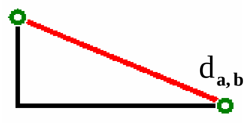
Finding a Winner

(iii) Find the best-matching neuron w(**x**), usually the neuron whose weight vector has smallest Euclidean distance from the input vector **x:**

The winning node is that which is in some sense ‘closest’ to the input vector

‘Euclidean distance’ is the straight line distance between the data points, if they were plotted on a (multi-dimensional) graph.

Euclidean distance between two vectors **a** and **b**, **a** = (a1,a2,…,an), **b** = (b1,b2,…bn), is calculated as:



$$d_{\mathbf{a},\mathbf{b}} = \sqrt{\sum_i (a_i - b_i)^2}$$

Weight Update-SOM Weight Update Equation

$\mathbf{w}j(t+1) = \mathbf{w}j(t) + \mu(t)\,\lambda\omega(\mathbf{x})(j,t)\,[\mathbf{x} - \mathbf{w}j(t)]$

“The weights of every node are updated at each cycle by adding

Current learning rate × Degree of neighbourhood with respect to winner × Difference between current weights and input vector to the current weights”

3.5 Principal Component Analysis

3.5.1 Introduction

Principal Component Analysis (PCA) [24-26], also known as Singular Value Decomposition (SVD) is an exploratory multivariate statistical technique that allows the identification of key variables (or combinations of variables) in a multidimensional data set that best explains the differences between observations. Given *m* observations (experiments) on *n* variables (genes), the goal of PCA is to reduce the dimensionality of the data matrix by finding *r*<=*n* new variables. These *r* principal components account together for as much of the variance in the original *n* variables as possible while remaining mutually uncorrelated and orthogonal.

3.5.2 Mathematical background

Consider *m* observations on *n* random variables represented by the matrix X. D is a distance matrix of the input matrix X. Let P denote a *(m x m)* matrix of unknown coefficients such that the quadratic form PTDP is maximized subject to the constraint pTp = I. This is equivalent to maximizing the Lagrangean expression

$$\mathbf{\Phi} = \mathbf{P}^{T}\mathbf{D}\mathbf{P} - \lambda\mathbf{I}(\mathbf{P}^{T}\mathbf{P} - \mathbf{I})$$

Differentiating with respect to P and setting the equation to zero we are receiving

$$\frac{\partial \Phi}{\partial \mathrm{P}} = 2\mathrm{D} - 2\lambda \mathrm{P} = 0$$

$$(\mathrm{D}-\lambda \mathrm{I})\mathrm{P}=0$$

The normal equations yield estimates for Eigenvalues and Eigenvectors. To compute the principal components, the *m* Eigenvalues and their corresponding Eigenvectors are calculated from the *(m x m)* distance matrix D using for example Singular Value Decomposition (SVD). When D is nonsingular, all latent roots are strictly positive and each. Eigenvector defines a principal component.

SVD methods are based on the following theorem of linear algebra: Any *(n x m)* matrix A whose number of rows *n* is greater than or equal to its number of columns *m,* can be written as the product of a *(n x m)* column-orthogonal matrix **U,** a *(m x m)* diagonal matrix W with positive or zero elements (the singular values), and the transpose of an *(m x m)* orthogonal matrix V.

$$\mathbf{A} = \mathbf{U}\mathbf{W}\mathbf{V}^{T}$$

SVD now explicitly constructs orthonormal bases for the nullspace and range of a matrix. Specifically, the columns of U whose same-numbered elements *Wj* are nonzero are an orthonormal set of basis vectors that span the range; the columns of V whose same-numbered elements *Wj* are zero are an orthonormal basis for the nullspace.

The matrices U and V are each orthogonal in the sense that their columns are orthonormal.

$$U^TU = V^TV=VV^T=1$$

The vectors of U contain our Eigenvectors and the diagonal elements of W contain the corresponding Eigenvalues. Now the Eigenvectors of U are ordered regarding the value of their corresponding Eigenvalues. Each Eigenvector defines a principal component. Principal Component 1 (PC1) is the Eigenvector with the greatest Eigenvalue; PC2 is the Vector with the 2nd largest Eigenvalue and so on.

The new ordered U matrix is the requested matrix P of equation; W contains the Eigenvalues 00.1. Since U is an orthonormal matrix, it can be seen as a Transformation matrix, which transforms a vector from the input space into the space spanned by the Principal Components.

$$Y=XU$$

Each component can be viewed as a weighted sum of conditions, where the coefficients of the Eigenvectors are the weights. The decision of gene *i* along the axis defined by the jth principal component is:

$$y_{ij} = \sum_{t=1}^{m} x_{it} u_{tj}$$

Where u_{tj} is the tth coefficient for the jth principal component; X_{if} is the expression measurement for gene *i* under the tth condition. Y represents the data in terms of principal components and is a rotation of the data: from the original space of observations to a new space with principal component axes (PC Space).

The variance accounted for by each of the components is its associated Eigenvalue; it is the variance of a component over all genes. Consequently, the Eigenvectors with large Eigenvalues are the ones that contain most of the information; Eigenvectors with small Eigenvalues are uninformative.

3.5.3 Properties

Principal Component Analysis can be used to retrieve the basic patterns of gene expression contained in a given study. It eliminates the noise part of the dataset and concentrates on the most variant aspects of the investigated observation. PCA can also be applied to study clusters of genes from other calculations in PC space. If the clusters are well self-contained, they are usually better than clusters that are spread across the PC Space. It has to be mentioned at this place, that normalizing data adjustments remove a lot of the variation in the dataset and therefore impair the PCA. Normalization tends to thesis the genes/experiments on a circular or spherical shape in the PC Space. Since this is an exact mathematical calculation, no parameters have to be specified.

Chapter 4
Implementation Issues

Clustering is widely used in gene expression data analysis. By grouping genes together based on the similarity between their gene expression profiles, functionally related genes may be found. Such a grouping suggests the function of presently unknown genes.

The Clustering is a collection of numerical routines that implement the clustering algorithms that are most commonly used. The routines can be applied both to genes and to arrays. The clustering algorithms are:

- Hierarchical clustering (pairwise centroid-, single-, complete-, and average-linkage);
- k-means clustering;
- Self-Organizing Maps;
- Principal Component Analysis.

To measure the similarity or distance between gene expression data, eight distance measures are available:

- Pearson correlation;
- Absolute value of the Pearson correlation;
- Uncentered Pearson correlation (equivalent to the cosine of the angle between two data vectors) ;
- Absolute uncentered Pearson correlation (equivalent to the cosine of the smallest angle between two data vectors);
- Spearman's rank correlation;
- Kendall's *T;*
- Euclidean distance;
- Harmonically summed Euclidean distance;
- City-block distance.

4.1 Distance Function

In order to cluster gene expression data into groups with similar genes or microarrays, we should first define what exactly we mean by *similar*. In the C Clustering Library, eight distance functions are available to measure similarity, or conversely, distance [27]:

'c'	Pearson correlation coefficient;
'a'	Absolute value of the Pearson correlation coefficient;
'u'	Uncentered Pearson correlation (equivalent to the cosine of the angle between two data vectors);
'x'	Absolute uncentered Pearson correlation;
's'	Spearman's rank correlation;
'k'	Kendall's *T;*
'e'	Euclidean distance;
'h'	Harmonically summed Euclidean distance;
'b'	City-block distance.

The first six of these distance measures are related to the correlation coefficient, while the remaining three are related to the Euclidean distance. The characters in front of the distance measures are used as mnemonics to be passed to various routines in the Clustering.

One of the properties one would like to see in a distance function is that it satisfies the triangle inequality:

$$d\left(\underline{u},\underline{v}\right) \leq d\left(\underline{u},\underline{w}\right) + d\left(\underline{w},\underline{v}\right) \text{ for all } \underline{\mathrm{u}}, \underline{\mathrm{v}}, \underline{\mathrm{w}}.$$

In everyday language, this equation means that the shortest distance between two points is a straight line. Correlation-based distance functions usually define the distance *d* in terms of the correlation *r* as

$$d = 1 - r;$$

All correlation-based similarity measures are converted to a distance using this definition. Note that this distance function does not satisfy the triangle inequality. As an example, try

$\underline{u}$= (1, 0,-1);

$\underline{v}$ = (1, 1, 0);

$\underline{w}$ = (0, 1, 1);

Using the Pearson correlation, we find d (u, w) = 1.8660 while d (u, v) + d (v, w) = 1.6340. None of the distance functions based on the correlation coefficient satisfy the triangle in-equality; this is a general characteristic of the correlation coefficient. The Euclidean distance and the city-block distance, which are metrics, do satisfy the triangle inequality. The correlation-based distance functions are sometimes called semi-metric.

4.2 Data Handling

The input to the distance functions contains two arrays and two row or column indices, instead of two data vectors. This makes it easier to calculate the distance between two columns in the gene expression data matrix. If the distance functions would require two vectors, we would have to extract two columns from the matrix and save them in two vectors to be passed to the distance function. In order to specify if the distance between rows or between columns is to be calculated, each distance function has a flag *transpose*. If *transpose* ==0, then the distance between two rows is calculated. Otherwise, the distance between two columns is calculated.

4.3 Weighting

For most of the distance functions available in the Clustering, a weight vector can be applied. The weight vector contains weights for the elements in the data vector. If the weight for element i is *Wi*, then that element is treated as if it occurred *Wi* times in the data. The weights do not have to be integers. For the Spearman rank correlation and Kendall's *T*, discussed below, the weights do not have a well-defined meaning and are therefore not implemented.

4.4 Missing Values

Often in microarray experiments, some of the data values are missing. In the distance functions, we therefore use an additional matrix *mask* which shows which data values are missing. If *mask* [i] [j] ==0, then data [i] [j] is missing, and is not included in the distance calculation.

4.5 The Pearson Correlation Coefficient

The Pearson correlation [11] coefficient is defined as

$$r = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{\sigma_x}\right)\left(\frac{y_i - \bar{y}}{\sigma_y}\right)$$

in which *x*, yare the sample mean of *X* and *y* respectively, and *(lx*, *(ly* are the sample standard deviation of *x* and *y*. The Pearson correlation coefficient is a measure for how well a straight line can be fitted to a scatterplot of *x* and *y*. If all the points in the scatterplot lie on a straight line, the Pearson correlation coefficient is either +1 or -1, depending on whether the slope of line is positive or negative. If Pearson correlation coefficient is equal to zero, there is no correlation between *x* and *y*. The *Pearson distance* is then defined as

$$d_{\mathrm{P}} \equiv 1 - r.$$

As the Pearson correlation coefficient lies between -1 and 1, the Pearson distance lies between 0 & 2. The Pearson correlation automatically centers the data by subtracting the mean, and normalizes them by dividing by the standard deviation. While such normalization may be useful in some situations (e.g., when clustering gene expression levels directly instead of gene expression ratios), information is being lost in this step. In particular, the magnitude of changes in gene expression is being ignored. This is in fact the reason that the Pearson distance does not satisfy the triangle inequality.

4.6 Absolute Pearson Correlation

By taking the absolute value of the Pearson correlation, we find a number between zero and one. If the absolute value is one, all the points in the scatter plot lie on a straight line with either a positive or a negative slope. If absolute value is equal to zero, there is no correlation between *x* and *y*. The distance is defined as usual as

$$d_{\mathrm{A}} \equiv 1 - |r|\,,$$

where *r* is the Pearson correlation coefficient. As the absolute value of the Pearson correlation coefficient lies between 0 and 1, the corresponding distance lies between 0 and 1 as well. In the context of gene expression experiments, note that the absolute correlation is equal to one if the gene expression data of two genes microarray have a shape that is either exactly the same or exactly opposite. The absolute correlation coefficient should therefore be used with care.

4.7 Uncentered Correlation (cosine of the angle)

In some cases, it may be preferable to use the *uncentered correlation* instead of the regular Pearson correlation coefficient. The uncentered correlation is defined as

$$r_{\mathrm{U}} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i}{\sigma_x^{(0)}}\right)\left(\frac{y_i}{\sigma_y^{(0)}}\right),$$

where

$$\sigma_x^{(0)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2};$$

$$\sigma_y^{(0)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} y_i^2}.$$

This is the same expression as for the regular Pearson correlation coefficient, except that the sample means *x*, *y* are set equal to zero. The uncentered correlation may be appropriate if there is a zero reference state. For instance, in the case of gene expression data given in terms of log-ratios, a log-ratio equal to zero corresponds to the green and red signal being equal, which means that the experimental manipulation did not affect the gene expression.
The distance corresponding to the uncentered correlation coefficient is defined as

$$d_{\mathrm{U}} \equiv 1 - r_{\mathrm{U}},$$

where ru is the uncentered correlation. As the uncentered correlation coefficient lies between -1 and 1, the corresponding distance lies between 0 and 2.
The uncentered correlation is equal to the cosine of the angle of the two data vectors in n-dimensional space, and is often referred to as such. (From this viewpoint, it would make more sense to define the distance as the arc cosine of the uncentered correlation coefficient).

4.8 Absolute uncentered correlation

As for the regular Pearson correlation, we can define a distance measure using the absolute value of the uncentered correlation:

$$d_{\mathrm{AU}} \equiv 1 - \left|r_{\mathrm{U}}\right|,$$

where ru is the uncentered correlation coefficient. As the absolute value of the uncentered correlation coefficient lies between 0 and 1, the corresponding distance lies between 0 and 1 as well. Geometrically, the absolute value of the uncentered correlation is equal to the cosine between the supporting lines of the two data vectors (i.e., the angle without taking the direction of the vectors into consideration).

4.9 Spearman rank correlation

The Spearman rank correlation is an example of a non-parametric similarity measure. It is useful because it is more robust against outliers than the Pearson correlation.
To calculate the Spearman rank correlation, we replace each data value by their rank if we would order the data in each vector by their value. We then calculate the Pearson correlation between the two rank vectors instead of the data vectors.

Weights cannot be suitably applied to the data if the Spearman rank correlation is used, especially since the weights are not necessarily integers. The calculation of the Spearman rank correlation in the C Clustering Library therefore does not take any weights into consideration. As in the case of the Pearson correlation, we can define a distance measure corresponding to the Spearman rank correlation as

$$d_{\mathrm{S}} \equiv 1 - r_{\mathrm{S}},$$

Where r_S is the Spearman rank correlation.

4.10 Kendall's *T*

Kendall's *T* is another example of a non-parametric similarity measure. It is similar to the Spearman rank correlation, but instead of the ranks themselves only the relative ranks are used to calculate *T.* As in the case of the Spearman rank correlation, the weights are ignored in the calculation. We can define a distance measure corresponding to Kendall's *T* as

$$d_{\mathrm{K}} \equiv 1 - \tau.$$

As Kendall's *T* is defined such that it will lie between -1 and 1, the corresponding distance will be between 0 and 2.

4.11 Euclidean distance

The Euclidean distance is a true metric, as it satisfies the triangle inequality. The Euclidean distance is defined as

$$d = \sum_{i=1}^{n} (x_i - y_i)^2 .$$

In this formula, the expression data *Xi* and *Yi* are subtracted directly from each other. We should therefore make sure that the expression data are properly normalized when using the Euclidean distance, for example by converting the measured gene expression levels to log-ratios. Unlike the correlation-based distance functions, the Euclidean distance takes the magnitude of the expression data into account.

4.12 Harmonically summed Euclidean distance

The harmonically summed Euclidean distance is a variation of the Euclidean distance, where the terms for the different dimensions are summed inversely (similar to the harmonic mean):

$$d = \left[\frac{1}{n} \sum_{i=1}^{n} \left(\frac{1}{x_i - y_i} \right)^2 \right]^{-1} .$$

The harmonically summed Euclidean distance is more robust against outliers compared to the Euclidean distance. Note that the harmonically summed Euclidean distance is *not* a metric. For example, consider

$$\underline{u} = (1,0) ;$$
$$\underline{v} = (0,1) ;$$
$$\underline{w} = (1,1) .$$

This yields $d(\underline{u}, \underline{v}) = 1$ while $d(\underline{u}, \underline{w}) + d(\underline{w}, \underline{v}) = 0$.

4.13 City-block distance

The city-block distance, alternatively known as the Manhattan distance, is related to the Euclidean distance. Whereas the Euclidean distance corresponds to the length of the shortest path between two points, the city-block distance is the sum of distances along each dimension:

$$d = \sum_{i=1}^{n} |x_i - y_i| .$$

This is equal to the distance you would have to walk between two points in a city, where you have to walk along city blocks. The city-block distance is a metric, as it satisfies the triangle inequality. As for the Euclidean distance, the expression data are subtracted directly from each other, and we should therefore make sure that they are properly normalized.

4.14 Calculating the distance between clusters

In the hierarchical clustering methods, the distance matrix between all genes or microarrays is first calculated, and at successive steps of the algorithm the new distance matrix is calculated from the previous distance matrix. In some cases, however, we would like to calculate the distance between clusters directly, given their members. For this purpose, the function clusterdistance can be used. This function can also be used to calculate the distance between two genes and microarrays by defining two clusters consisting of one gene/microarray each. The distance between two clusters can be defined in several ways. The distance between the arithmetic means of the two clusters is used in pairwise centroid-linkage clustering and in k-means clustering. For the latter, the distance between the medians of the two clusters can be used alternatively. The shortest pairwise distance between elements of the two clusters is used in pairwise single-linkage clustering, while the longest pairwise distance is used in pairwise maximum-linkage clustering. In pairwise average-linkage clustering, the distance between two clusters is defined as the average over the pairwise distances.

Prototype

double clusterdistance (int *nrows*, int *ncolumns*, double *data*, int** *mask*, double *weight*[], int *n1*, int *n2*, int *index1*[], int *index2*[], char *dist*, char *method*, int *transpose*);**
returns the distance between two clusters.

Arguments

- **int *nrows*;**
  The number of rows in the *data* matrix, equal to the number of genes in the gene expression experiment.

- `int` *ncolumns* ;
  The number of columns in the data matrix, equal to the number of microarrays in the gene expression experiment.
- `double**` *data* ;
  The data array containing the gene expression data. Genes are stored row-wise, while microarrays are stored column-wise. Dimension: `[`*nrows*`][`*ncolumns*`]`.
- `int**` *mask* ;
  This array shows which elements in the *data* array, if any, are missing. If *mask*`[i][j]==0`, then *data*`[i][j]` is missing. Dimension: `[`*nrows*`][`*ncolumns*`]`.
- `double` *weight*`[]` ;
  The weights that are used to calculate the distance. Dimension: `[`*ncolumns*`]` if *transpose*`==0`; `[`*nrows*`]` if *transpose*`==1`.
- `int` *n1* ;
  The number of elements in the first cluster.
- `int` *n2* ;
  The number of elements in the second cluster.
- `int` *index1*`[]` ;
  Contains the indeces of the elements belonging to the first cluster. Dimension: `[`*n1*`]`.

4.15 Random Number Generator

- `int` *index2*`[]` ;
  Contains the indeces of the elements belonging to the second cluster. Dimension: `[`*n2*`]`.
- `char` *dist* ;
  Specifies which distance measure is used. See Chapter 2 [Distance functions], page 2.
- `char` *method*;
  Specifies how the distance between clusters is defined:

'a'	Distance between the two cluster centroids (arithmetic mean);
'm'	Distance between the two cluster centroids (median);
's'	Shortest pairwise distance between elements in the two clusters;
'x'	Longest pairwise distance between elements in the two clusters;
'v'	Average over the pairwise distances between elements in the two clusters.

- `int` *transpose* ;
  If *transpose*`==0`, the distances between rows in the data matrix are calculated. Otherwise, the distances between columns are calkulated.

The random number generator in the Clustering is used to initialize the *k*means clustering algorithm and Self-Organizing Maps (SOMs), as well as to randomly select a gene or microarray in the calculation of a SOM. This random number generator needs two seeds for initialization, for which we used the standard C random number generator srand. We initialize srand with the epoch time in seconds. The first two random numbers generated by srand are then used as seeds for the ranlib random number generator.

4.16 The Distance Matrix

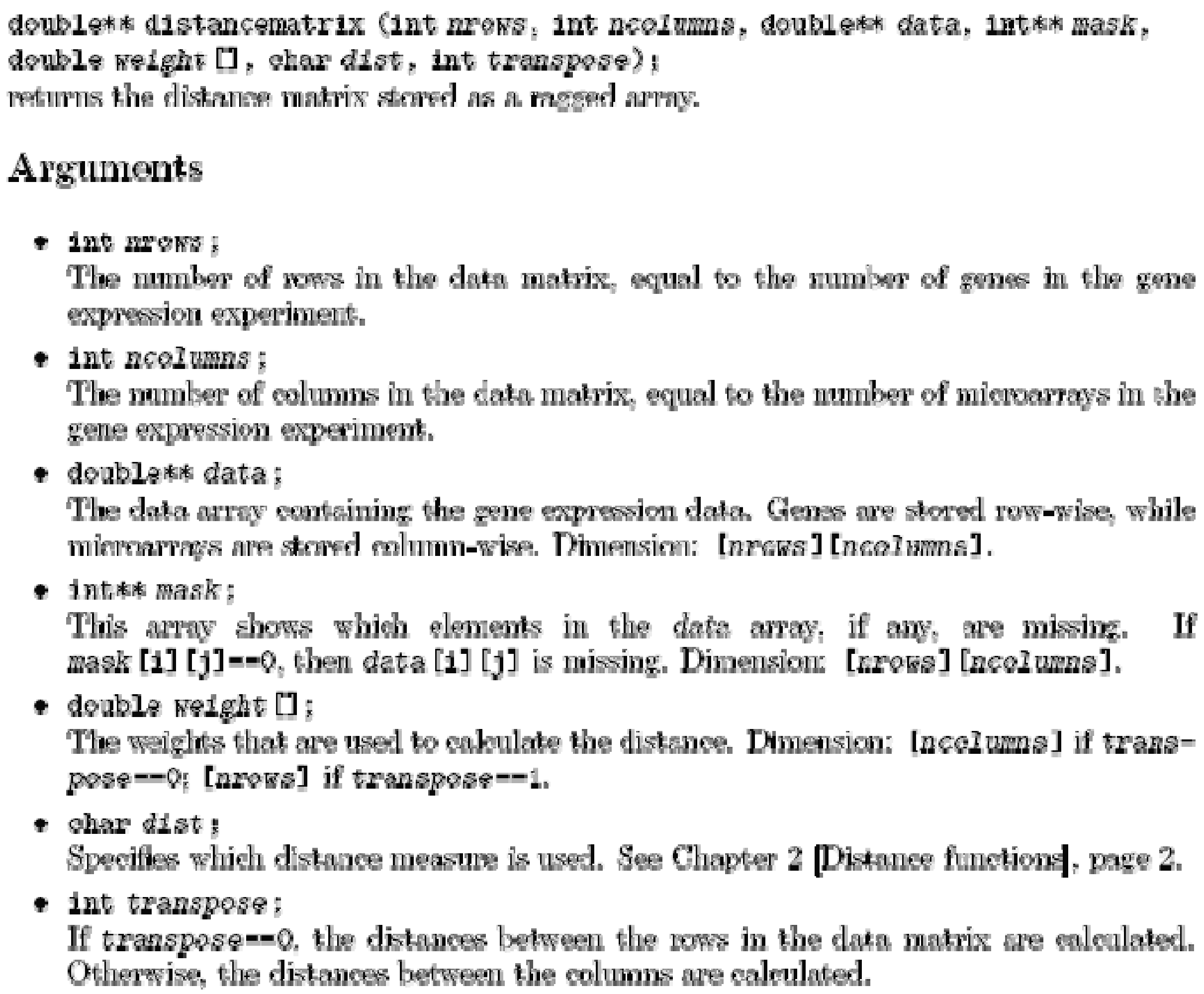
The first step in clustering problems is usually to calculate the distance matrix. This matrix contains all the distances between the items that are being clustered. As the distance functions are symmetric, the distance matrix is also symmetric. Furthermore, the elements on the diagonal are zero, as the distance of an item to itself is zero. The distance matrix can therefore be stored as a ragged array, with the number of columns in each row equal to the (zero-offset) row number. The distance between items i and j is stored in location
[i] [j] if $j < i$, in [j] [i] if $j > i$, while it is zero if $j = i$. Note that the first row of the distance matrix is empty. It is included for computational convenience, as including an empty row requires minimal storage.

4.17 Partitioning Algorithms

Partitioning algorithms divide items into k clusters such that the sum of distances over the items to their cluster centers is minimal. The number of clusters k is specified by the user. In the C Clustering Library, three partitioning algorithms are available:

- k-means clustering
- k-medians clustering
- k-medoids clustering

Prototype


`double** distancematrix (int nrows, int ncolumns, double** data, int** mask, double weight [], char dist, int transpose);`
returns the distance matrix stored as a ragged array.

Arguments

- `int nrows ;`
  The number of rows in the data matrix, equal to the number of genes in the gene expression experiment.
- `int ncolumns ;`
  The number of columns in the data matrix, equal to the number of microarrays in the gene expression experiment.
- `double** data ;`
  The data array containing the gene expression data. Genes are stored row-wise, while microarrays are stored column-wise. Dimension: `[nrows][ncolumns]`.
- `int** mask ;`
  This array shows which elements in the *data* array, if any, are missing. If `mask[i][j]==0`, then `data[i][j]` is missing. Dimension: `[nrows][ncolumns]`.
- `double weight [] ;`
  The weights that are used to calculate the distance. Dimension: `[ncolumns]` if *transpose*==0; `[nrows]` if *transpose*==1.
- `char dist ;`
  Specifies which distance measure is used. See Chapter 2 [Distance functions], page 2.
- `int transpose ;`
  If `transpose==0`, the distances between the rows in the data matrix are calculated. Otherwise, the distances between the columns are calculated.

These algorithms differ in how the cluster center is defined. In k-means clustering, the cluster center is defined as the mean data vector averaged over all items in the cluster. Instead of the mean, in k-medians clustering the median is calculated for each dimension in the data vector. Finally, in k-medoids clustering the cluster center is defined as the item which has the smallest sum of distances to the other items in the cluster. This clustering algorithm is suitable for cases in which the distance matrix is known but the original data matrix is not available, for example when clustering proteins based on their structural similarity.

The expectation-maximization (EM) algorithm is commonly used to find the partitioning into k groups. The first step in the EM algorithm is to create k clusters and randomly assign items (genes or microarrays) to them. We then iterate:

- Calculate the centroid of each cluster;
- For each item, determine which cluster centroid is closest;
- Reassign the item to that cluster.

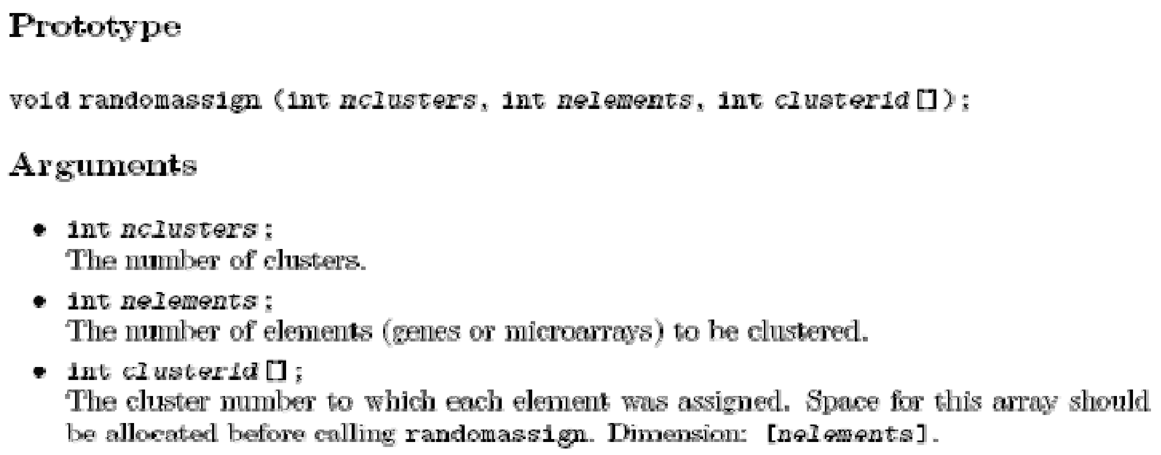
The iteration is stopped if no further item reassignments take place.

As the initial assignment of items to clusters is done randomly, usually a different clustering solution is found each time the EM algorithm is executed. To find the optimal clustering solution, the k-means algorithm is repeated many times, each time starting from a different initial random clustering. The sum of distances of the items to their cluster center is saved for each run, and the solution with the smallest value of this sum will be returned as the overall clustering solution.

How often the EM algorithm should be run depends on the number of items being clustered. As a rule of thumb, we can consider how often the optimal solution was found. This number is returned by the partitioning algorithms as implemented in this library. If the optimal solution was found many times, it is unlikely that better solutions exist than the one that was found. However, if the optimal solution was found only once, there may well be other solutions with a smaller within-cluster sum of distances.

4.17.1 Initialization

The k-means algorithm is initialized by randomly assigning items (genes or microarrays) to clusters. Special care should be taken to ensure that no empty clusters are produced. This is done by first choosing *k* items randomly and assigning each of them to a different cluster. The remaining items are then randomly assigned to clusters. Each cluster is thus guaranteed to contain at least one item.



Prototype

```
void randomassign (int nclusters, int nelements, int clusterid[]);
```

Arguments

- `int nclusters;`
  The number of clusters.
- `int nelements;`
  The number of elements (genes or microarrays) to be clustered.
- `int clusterid[];`
  The cluster number to which each element was assigned. Space for this array should be allocated before calling `randomassign`. Dimension: `[nelements]`.

4.18 Finding the Cluster Centroid

The centroid of a cluster can be defined in different ways. For k-means clustering, the centroid of a cluster is defined as the mean over all items in a cluster for each dimension separately. For robustness against outliers, in k-medians clustering the median is used instead of the mean. In k-medoids clustering, the cluster centroid is the item with the smallest sum of distances to the other items in the cluster. The C Clustering Library provides routines to calculate the cluster mean, the cluster median, and the cluster medoid.

4.18.1 Finding the cluster mean

The routine getclustermean calculates the centroids of the clusters by calculating the mean for each dimension separately over all items in a cluster. Missing data values are not included in the calculation of the mean. Whether the cluster means have a missing value is stored in an array *cmask.* If for cluster i the data values for dimension *j* are missing for all items, then *cmask* [i] [j] (or *cmask* [j] [i] if *transpose==1)* is set equal to zero. Otherwise, it is set equal to one.

Prototype

```
void getclustermean (int nclusters, int nrows, int ncolumns, double** data,
int** mask, int clusterid[], double** cdata, int** cmask, int transpose);
```

Arguments

- `int nclusters;`
  The number of clusters.
- `int nrows;`
  The number of rows in the data matrix, equal to the number of genes in the gene expression experiment.
- `int ncolumns;`
  The number of columns in the data matrix, equal to the number of microarrays in the gene expression experiment.
- `double** data;`
  The data array containing the gene expression data. Genes are stored row-wise, while microarrays are stored column-wise. Dimension: `[nrows][ncolumns]`.
- `int** mask;`
  This array shows which elements in the data array, if any, are missing. If `mask[i][j]==0`, then `data[i][j]` is missing. Dimension: `[nrows][ncolumns]`.
- `int clusterid[];`
  The cluster number to which each item belongs. Each element in this array should be between 0 and `nclusters-1` inclusive. Dimension: `[nrows]` if `transpose==0`, or `[ncolumns]` if `transpose==1`.
- `double** cdata;`
  This matrix stores the centroid information. Space for this matrix should be allocated before calling getclustermean. Dimension: `[nclusters][ncolumns]` if *transpose==0* (row-wise clustering), or `[nrows][nclusters]` if `transpose==1` (column-wise clustering).
- `int** cmask;`
  This matrix stores which values in cdata are missing. If `cmask[i][j]==0`, then `cdata[i][j]` is missing. Space for cmask should be allocated before calling getclustermean. Dimension: `[nclusters][ncolumns]` if `transpose==0` (row-wise clustering), or `[nrows][nclusters]` if `transpose==1` (column-wise clustering).

- `int transpose;`
  This flag indicates whether row-wise (gene) or column-wise (microarray) clustering is being performed. If transpose==0, rows (genes) are being clustered. Otherwise, columns (microarrays) are being clustered.

4.18.2 Finding the Cluster Median

The routine getclustermedian calculates the centroids of the clusters by calculating the median for each dimension separately over all items in a cluster. Missing data values are not included in the calculation of the median. Whether the cluster medians have a missing value is stored in an array *cmask.* If for cluster i the data values for dimension *j* are missing for all items, then *cmask* [i] [j] (or *cmask* [j] [i] if *transpose* ==1) is set equal to zero. Otherwise, it is set equal to one. Calculating the median may take significantly longer than calculating the mean.

Prototype

```
void getclustermedian (int nclusters, int nrows, int ncolumns, double** data,
int** mask, int clusterid[] , double** cdata, int** cmask, int transpose);
```

Arguments

- `int nclusters;`
  The number of clusters.
- `int nrows;`
  The number of rows in the data matrix, equal to the number of genes in the gene expression experiment.
- `int ncolumns;`
  The number of columns in the data matrix, equal to the number of microarrays in the gene expression experiment.
- `double** data;`
  The data array containing the gene expression data. Genes are stored row-wise, while microarrays are stored column-wise. Dimension: `[nrows][ncolumns]`.
- `int** mask;`
  This array shows which elements in the data array, if any, are missing. If `mask[i][j]==0`, then `data[i][j]` is missing. Dimension: `[nrows][ncolumns]`.
- `int clusterid[];`
  The cluster number to which each item belongs. Each element in this array should be between 0 and `nclusters-1` inclusive. Dimension: `[nrows]` if `transpose==0`, or `[ncolumns]` if `transpose==1`.
- `double** cdata;`
  This matrix stores the centroid information. Space for this matrix should be allocated before calling `getclustermedian`. Dimension: `[nclusters][ncolumns]` if `transpose==0` (row-wise clustering), or `[nrows][nclusters]` if `transpose==1` (column-wise clustering).
- `int transpose;`
  This flag indicates whether row-wise (gene) or column-wise (microarray) clustering is being performed. If `transpose==0`, rows (genes) are being clustered. Otherwise, columns (microarrays) are being clustered.

4.18.3 Finding the Cluster Medoid

The cluster medoid is defined as the item which has the smallest sum of distances to the other items in the cluster. The getclustermedoid routine calculates the cluster centroids, given to which cluster each item belongs. The centroid is defined as the item with the smallest sum of distances to the other items.

Prototype

```
void getclustermedoid(int nclusters, int nelements, double** distance, int
clusterid[], int centroids[], double errors[]);
```

Arguments

- `int nclusters;`
  The number of clusters.
- `int nelements;`
  The total number of elements that are being clustered.
- `double** distmatrix;`
  The distance matrix. The distance matrix is symmetric and has zeros on the diagonal. To save space, the distance matrix is stored as a ragged array. Dimension: `[nelements][]` as a ragged array. The number of columns in each row is equal to the row number (starting from zero). Accordingly, the first row always has zero columns.
- `int clusterid[];`
  The cluster number to which each element belongs. Dimension: [*nelements*].
- `int centroid[];`
  For each cluster, the element of the item that was determined to be its centroid. Dimension: [*nclusters*].
- `int errors[];`
  For each cluster, the sum of distances between the items belonging to the cluster and the cluster centroid. Dimension: [*nclusters*].

4.19 The EM Algorithm

The EM algorithm as implemented in the Clustering first randomly assigns items to clusters using randomassign, followed by iterating to find a clustering solution with a smaller within-cluster sum of distances. During the iteration, first we find the centroids of all clusters, where the centroids are defined in terms of the mean, the median, or the medoid. The distances of each item to the cluster centers are calculated, and we determine for each item which cluster is closest. We then reassign the items to their closest clusters, and recalculate the cluster centers.

All items are first reassigned before recalculating the cluster centroids. This has two consequences:

- If unchecked, clusters may become empty if all their items are reassigned. For k-means and k-medians clustering, the EM routine therefore keeps track of the number of items in each cluster at all times, and prohibits an item to be reassigned to a different cluster if that would cause its current cluster to become empty. For k-medoids clustering, such a check is not needed, as the item that functions as the cluster centroid has a zero distance to itself, and would therefore not be reassigned to a different cluster anyway.
- In principle, the order in which items are reassigned to clusters does not matter. However, since we force an item to stay in a cluster if it is the last remaining item, for k-means and k-medians clustering we need to randomize the order anyway to ensure that not always the same items are forced to stay in a cluster. For k-medoids clustering, no such randomization is needed.

The EM algorithm terminates when no further reassignments take place. We noticed, however, that for some sets of initial cluster assignments, the EM algorithm fails to converge due to the same clustering solution reappearing periodically after a small number of iteration steps. In the EM algorithm as implemented in the Clustering, the occurrence of such periodic solutions is checked for. After a given number of iteration steps, the current clustering result is saved as a reference. By comparing the clustering result after each subsequent iteration step to the reference state, we can determine if a previously encountered clustering result is found. In such a case, the iteration is halted. If after a given number of iterations the reference state has not yet been encountered, the current clustering solution is saved to be used as the new reference state. Initially, ten iteration steps are executed before resaving the reference state. This number of iteration steps is doubled each time, to ensure that periodic behavior with longer periods can also be detected.

4.19.1 Finding the Optimal Solution

K-means and k-medians

The optimal solution is found by executing the EM algorithm repeatedly and saving the best clustering solution that was returned by it. This can be done automatically by calling the routine kcluster. This procedure first initializes ranlib's random number generator. The routine to calculate the cluster centroid and the distance function are selected based on the arguments passed to kcluster.

The EM algorithm is then executed repeatedly, saving the best clustering solution that was returned by these routines. In addition, kcluster counts how often the EM algorithm found this solution. If it was found many times, we can assume that there are no other solutions possible with a smaller within-cluster sum of distances. If, however, the solution was found only once, it may well be that better clustering solutions exist.

Prototype

```
void kcluster (int nclusters, int nrows, int ncolumns, double** data, int**
mask, double weight[], int transpose, int npass, char method, char dist, int
clusterid[], double** cdata, double* error, int* ifound);
```

Arguments

- `int` *nclusters*;
  The number of clusters *k*.
- `int` *nrows*;
  The number of rows in the data matrix, equal to the number of genes in the gene expression experiment.

- `int` *ncolumns*;
  The number of columns in the data matrix, equal to the number of microarrays in the gene expression experiment.
- `double**` *data*;
  The data array containing the gene expression data. Genes are stored row-wise, while microarrays are stored column-wise. Dimension: `[`*nrows*`][`*ncolumns*`]`.
- `int**` *mask*;
  This array shows which elements in the *data* array, if any, are missing. If *mask*`[i][j]==0`, then *data*`[i][j]` is missing. Dimension: `[`*nrows*`][`*ncolumns*`]`.
- `double` *weight*`[]`;
  The weights that are used to calculate the distance. Dimension: `[`*ncolumns*`]` if *transpose*`==0`; `[`*nrows*`]` if *transpose*`==1`.
- `int` *transpose*;
  This flag indicates whether row-wise (gene) or column-wise (microarray) clustering is being performed. If *transpose*`==0`, rows (genes) are being clustered. Otherwise, columns (microarrays) are being clustered.
- `int` *npass*;
  The number of times the EM algorithm should be run. If *npass* `> 0`, each run of the EM algorithm uses a different (random) initial clustering. If *npass* `== 0`, then the EM algorithm is run with an initial clustering specified by *clusterid*. For *npass* `> 0`, items are reassigned to clusters in a randomized order. Since the cluster centroids are recalculated only once after all items have been considered for reassignment, the order of item reassignment is relevant only when the last item in a cluster is about to be reassigned to a different cluster. To prevent clusters from becoming empty, such reassignments are not allowed; which items are reassigned may therefore depend on the order in which items are considered. For *npass*`==0`, the EM algorithm is run only once, without randomizing the order in which items are reassigned to clusters, using

the initial clustering as specified by *clusterid*. The order in which items are reassigned is identical to the order in which items are given in the data matrix.

- `char method;`
  Specifies whether the arithmetic mean (*method*=='a') or the median (*method*=='m') should be used to calculate the cluster center.
- `char dist;`
  Specifies which distance function should be used. The character should correspond to one of the distance functions that are available in the C Clustering Library. See Chapter 2 [Distance functions], page 2.
- `int clusterid[];`
  This array will be used to store the cluster number to which each item was assigned by the clustering algorithm. Space for `clusterid` should be allocated before calling `kcluster`. If *npass*==0, then the contents of *clusterid* on input is used as the initial assignment of items to clusters; on output, *clusterid* contains the optimal clustering solution found by the EM algorithm. Dimension: [*nrows*] if *transpose*==0, or [*ncolumns*] if *transpose*==1.
- `double** cdata;`
  This matrix stores the centroid information. Space for *cdata* should be allocated before

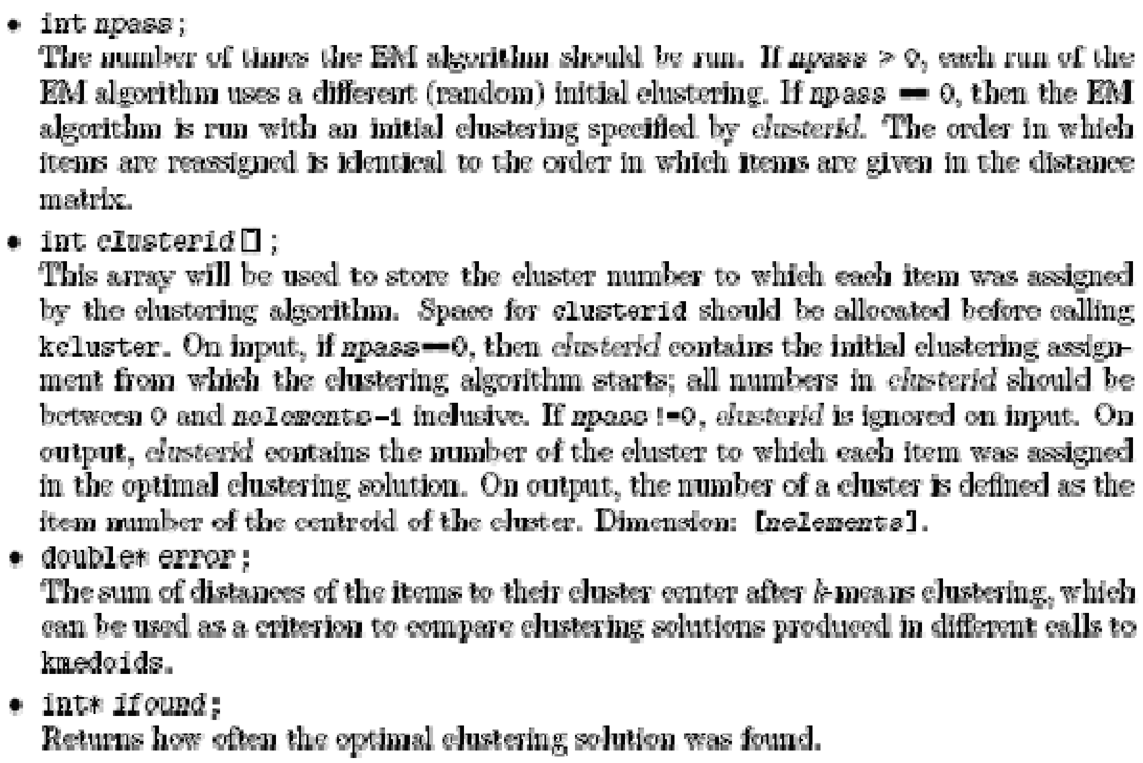
  calling `kcluster`. Dimension: [*nclusters*][*ncolumns*] if *transpose*==0 (row-wise clustering), or [*nrows*][*nclusters*] if *transpose*==1 (column-wise clustering).
- `double* error;`
  The sum of distances of the items to their cluster center after k-means clustering, which can be used as a criterion to compare clustering solutions produced in different calls to `kcluster`.
- `int* ifound;`
  Returns how often the optimal clustering solution was found.

4.19.2 k-medoids

The kmedoids routine performs k-medoids clustering on a given set of elements, using the distance matrix and the number of clusters passed by the user. Multiple passes are being made to find the optimal clustering solution, each time starting from a different initial clustering.

Arguments

- `int nclusters;`
  The number of clusters to be found.
- `int nelements;`
  The number of elements to be clustered.
- `double** distmatrix;`
  The distance matrix. The distance matrix is symmetric and has zeros on the diagonal. To save space, the distance matrix is stored as a ragged array. Dimension: [*nelements*][] as a ragged array. The number of columns in each row is equal to the row number (starting from zero). Accordingly, the first row always has zero columns.


- `int npass;`
  The number of times the EM algorithm should be run. If *npass* > 0, each run of the EM algorithm uses a different (random) initial clustering. If npass == 0, then the EM algorithm is run with an initial clustering specified by *clusterid*. The order in which items are reassigned is identical to the order in which items are given in the distance matrix.
- `int clusterid[];`
  This array will be used to store the cluster number to which each item was assigned by the clustering algorithm. Space for `clusterid` should be allocated before calling `kcluster`. On input, if *npass*==0, then *clusterid* contains the initial clustering assignment from which the clustering algorithm starts; all numbers in *clusterid* should be between 0 and `nelements-1` inclusive. If *npass* !=0, *clusterid* is ignored on input. On output, *clusterid* contains the number of the cluster to which each item was assigned in the optimal clustering solution. On output, the number of a cluster is defined as the item number of the centroid of the cluster. Dimension: `[nelements]`.
- `double* error;`
  The sum of distances of the items to their cluster center after *k*-means clustering, which can be used as a criterion to compare clustering solutions produced in different calls to `kmedoids`.
- `int* ifound;`
  Returns how often the optimal clustering solution was found.

4.20 Choosing the Distance Measure

Whereas all eight distance measures are accepted for k-means, k-medians, and k-medoids clustering, using a distance measure other than the Euclidean distance or city-block distance with k-means or k-medians is in a sense inconsistent. When using the distance measures based on the Pearson correlation, the data are effectively normalized when calculating the distance. However, no normalization is applied when calculating the centroid in the *k*means or k-medians algorithm. From a theoretical viewpoint, it is best to use the Euclidean distance for the k-means algorithm, and the city-block distance for k-medians.

4.21 Hierarchical Clustering

In hierarchical clustering there are various methods.

4.21.1 Hierarchical clustering methods

Hierarchical clustering methods are inherently different from the k-means clustering method. In hierarchical clustering methods, gene expression data are described in terms of a tree structure. While the existence of such a tree structure may be debatable, the hierarchical clustering methods are quite popular in the analysis of gene expression data.

The first step in hierarchical clustering is to calculate the distance matrix, specifying all the distances between the items to be clustered. Next, we create a node by joining the two closest items. Subsequent nodes are created by pairwise joining of items or nodes based on the distance between them, until all items belong to the same node. A tree structure can then be created by retracing which items and nodes were merged. Unlike the EM algorithm, which is used in k-means clustering, the complete process of hierarchical clustering is deterministic.

Several flavors of hierarchical clustering exist, which differ in how the distance between subnodes is defined in terms of their members. In the C Clustering Library, pairwise single, maximum, average, and centroid linkage are available.

- In pairwise single-linkage clustering, the distance between two nodes is defined as the shortest distance among the pairwise distances between the members of the two nodes.
- In pairwise maximum-linkage clustering, alternatively known as pairwise complete linkage clustering, the distance between two nodes is defined as the longest distance among the pairwise distances between the members of the two nodes.
- In pairwise average-linkage clustering, the distance between two nodes is defined as the average over all pairwise distances between the elements of the two nodes.
- In pairwise centroid-linkage clustering, the distance between two nodes is defined as the distance between their centroids. The centroids are calculated by taking the mean over all the elements in a cluster. As the distance from each newly formed node to existing nodes and items need to be calculated at each step, the computing time of pairwise centroid-linkage clustering may be significantly longer than for the other hierarchical clustering methods. Another peculiarity is that (for a distance measure based on the Pearson correlation), the distances do not necessarily increase when going up in the clustering tree, and may even decrease. This is caused by an inconsistency between the centroid calculation and the distance calculation when using the Pearson correlation: Whereas the Pearson correlation effectively normalizes the data for the distance calculation, no such normalization occurs for the centroid calculation.

For pairwise single-, complete-, and average-linkage clustering, the distance between two nodes can be found directly from the distances between the individual items. Therefore, the clustering algorithm does not need access to the original gene expression data, once the distance matrix is known. For pairwise centroid-linkage clustering, however, the centroids of newly formed subnodes can only be calculated from the original data and not from the distance matrix.

Straightforward implementation of pairwise single-linkage clustering. The clustering result produced by this algorithm is identical to the clustering solution found by the conventional single-linkage algorithm. The single-linkage hierarchical clustering algorithm implemented in this library can be used to cluster large gene expression data sets, for which conventional hierarchical clustering algorithms fail due to excessive memory requirements and running time.

The treecluster routine described below implements pairwise single-, complete, average-, and centroid-linkage clustering. A pointer *distmatrix* to the distance matrix can be passed as one of the arguments to treecluster; if this pointer is NULL, the treecluster routine will calculate the distance matrix from the gene expression data using the arguments data, *mask, weight,* and *dist.* For pairwise single-, complete-, and average-linkage clustering, the treecluster routine ignores these four arguments if *distmatrix* is given, as the distance matrix by itself is sufficient for the clustering calculation. For pairwise centroid-linkage clustering, the arguments data, *mask, weight,* and *dist* are always needed, even if *distmatrix* is available.

The treecluster routine will complete faster if it can make use of a previously calculated distance matrix passed as the *distmatrix* argument. Note, however, that newly calculated distances are stored in the distance matrix, and its elements may be rearranged during the clustering calculation. Therefore, in order to save the original distance matrix, it should be copied before treecluster is called. The memory that was allocated by the calling routine for the distance matrix will not be deallocated by treecluster, and should be deallocated by the calling routine after treecluster returns. If *distmatrix* is NULL, however, treecluster takes care both of the allocation and the deallocation of memory for the distance matrix. In that case, treecluster may fail if not enough memory can be allocated for the distance matrix, in which case treecluster returns o.

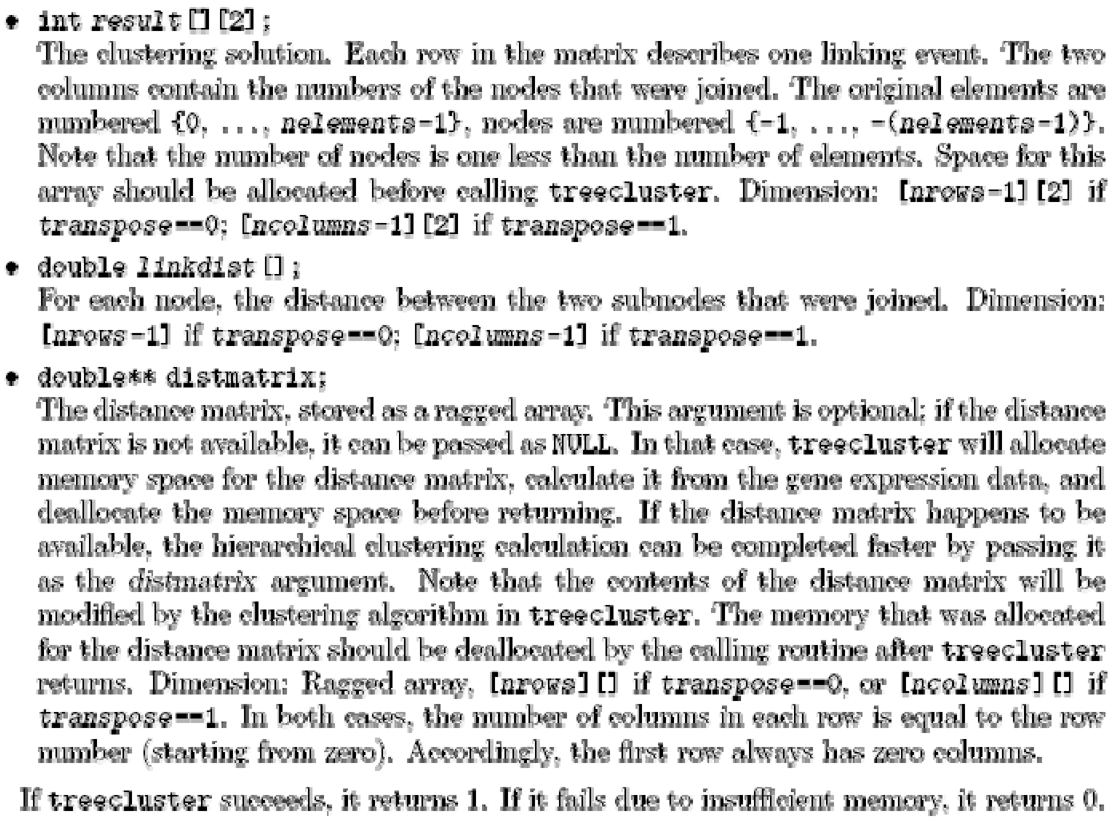
Prototype

`int treecluster (int nrows, int ncolumns, double** data, int** mask, double weight[], int transpose, char dist, char method, int result[][2], double linkdist[], double** distmatrix);`

Arguments

- `int nrows;`
  The number of rows in the *data* matrix, equal to the number of genes in the gene expression experiment.
- `int ncolumns;`
  The number of columns in the *data* matrix, equal to the number of microarrays in the gene expression experiment.
- `double** data;`
  The data array containing the gene expression data. Genes are stored row-wise, while microarrays are stored column-wise. Dimension: `[nrows][ncolumns]`.
- `int** mask;`
  This array shows which elements in the *data* array, if any, are missing. If `mask[i][j]==0`, then `data[i][j]` is missing. Dimension: `[nrows][ncolumns]`.

- `double weight[];`
  The weights that are used to calculate the distance. Dimension: `[ncolumns]` if *transpose*==0; `[nrows]` if *transpose*==1.
- `int transpose;`
  This flag indicates whether row-wise (gene) or column-wise (microarray) clustering is being performed. If *transpose*==0, rows (genes) are being clustered. Otherwise, columns (microarrays) are being clustered.
- `char dist;`
  Specifies which distance measure is used. See Chapter 2 [Distance functions], page 2.
- `char method;`
  Specifies which type of hierarchical clustering is used:
  - `'s'`: pairwise single-linkage clustering
  - `'m'`: pairwise maximum- (or complete-) linkage clustering
  - `'a'`: pairwise average-linkage clustering
  - `'c'`: pairwise centroid-linkage clustering



- `int result[][2];`
  The clustering solution. Each row in the matrix describes one linking event. The two columns contain the numbers of the nodes that were joined. The original elements are numbered {0, ..., *nelements*-1}, nodes are numbered {-1, ..., -(*nelements*-1)}. Note that the number of nodes is one less than the number of elements. Space for this array should be allocated before calling `treecluster`. Dimension: `[`*nrows*`-1][2]` if *transpose*==0; `[`*ncolumns*`-1][2]` if *transpose*==1.
- `double linkdist[];`
  For each node, the distance between the two subnodes that were joined. Dimension: `[`*nrows*`-1]` if *transpose*==0; `[`*ncolumns*`-1]` if *transpose*==1.
- `double** distmatrix;`
  The distance matrix, stored as a ragged array. This argument is optional; if the distance matrix is not available, it can be passed as `NULL`. In that case, `treecluster` will allocate memory space for the distance matrix, calculate it from the gene expression data, and deallocate the memory space before returning. If the distance matrix happens to be available, the hierarchical clustering calculation can be completed faster by passing it as the *distmatrix* argument. Note that the contents of the distance matrix will be modified by the clustering algorithm in `treecluster`. The memory that was allocated for the distance matrix should be deallocated by the calling routine after `treecluster` returns. Dimension: Ragged array, `[`*nrows*`][]` if *transpose*==0, or `[`*ncolumns*`][]` if *transpose*==1. In both cases, the number of columns in each row is equal to the row number (starting from zero). Accordingly, the first row always has zero columns.

If `treecluster` succeeds, it returns 1. If it fails due to insufficient memory, it returns 0.

4.21.2 Cutting a hierarchical clustering tree

The tree structure generated by the hierachical clustering routine treecluster can be further analyzed by dividing the genes or microarrays into *n* clusters, where *n* is some positive integer less than or equal to the number of elements that were clustered. This can be achieved by ignoring the top *n* - 1 linking events in the tree structure, resulting in *n* separated subnodes. The elements in each subnode are then assigned to the same cluster. The routine cuttree determines to which cluster each element is assigned, based on the hierarchical clustering result stored in the tree structure.

Prototype

`void cuttree (int nelements, int tree[][2], int nclusters, int clusterid[]);`

Arguments

- `int` *`nelements`*`;`
  The number of elements whose clustering results are stored in the tree hierarchical clustering result.
- `int tree[][2];`
  The hierarchical clustering solution. Each row in the matrix describes one linking event. The two columns contain the numbers of the nodes that were joined. The original elements are numbered `{0, ..., `*`nelements`*`-1}`, nodes are numbered `{-1, ..., -(`*`nelements`*`-1)}`. Note that the number of nodes is one less than the number of elements. The `cuttree` routine performs some error checking of the structure of the tree argument in order to avoid segmentation faults. However, errors in the structure of tree that would not result in segmentation faults will in general not be detected. Dimension: `[`*`nelements`*`-1][2]`.
- `int` *`nclusters`*`;`
  The desired number of clusters. The number of clusters should be positive, and less than or equal to *nelements*.
- `int` *`clusterid`*`[];`
  The cluster number to which each element is assigned. Memory space for *clusterid* should be allocated before `cuttree` is called. Dimension: `[`*`nelements`*`]`.

4.22 Self-Organizing Maps

4.22.1 Introduction

Self-Organizing Maps (SOMs) were invented by Kohonen to describe neural networks (see for instance Kohonen, 1997). Tamayo (1999) first applied Self-Organizing Maps to gene expression data.

SOMs organize items into clusters that are situated in some topology. Usually a rectangular topology is chosen. The clusters generated by SOMs are such that neighboring clusters in the topology are more similar to each other than clusters far from each other in the topology.
The first step to calculate a SOM is to randomly assign a data vector to each cluster in the topology. If genes are being clustered, then the number of elements in each data vector is equal to the number of microarrays in the experiment.

An SOM is then generated by taking genes one at a time, and finding which cluster in the topology has the closest data vector. The data vector of that cluster, as well as neighboring clusters is adjusted using the data vector of the gene under consideration. The adjustment is given by

$$\Delta \underline{x}_{\text{cell}} = \tau \cdot (\underline{x}_{\text{gene}} - \underline{x}_{\text{cell}}) .$$

The parameter τ is a parameter that decreases at each iteration step. We have used a simple linear function of the iteration step:

$$\tau = \tau_{\text{init}} \cdot \left(1 - \frac{i}{n}\right) ,$$

in which T init is the initial value of *T* as specified by the user, i is the number of the current iteration step, and *n* is the total number of iteration steps to be performed. While changes are made rapidly in the beginning of the iteration, at the end of iteration only small changes are made.

All clusters within a radius *R* are adjusted to the gene under consideration. This radius decreases as the calculation progresses as

$$R = R_{\max} \cdot \left(1 - \frac{i}{n}\right) ,$$

in which the maximum radius is defined as

$$R_{\max} = \sqrt{N_x^2 + N_y^2},$$

where *(Nx, Ny)* are the dimensions of the rectangle defining the topology.
The routine somcluster carries out the complete SOM algorithm. First it initializes the random number generator. The distance function to be used is specified by *dist.* The node data are then initialized using the ranlib random number generator. The order in which genes or microarrays are used to modify the SOM is also randomized. The total number of iterations is specified by *niter,* given by the user.

Prototype

```
void somcluster (int nrows, int ncolumns, double** data, int** mask, double
weight[], int transpose, int nxgrid, int nygrid, double inittau, int niter,
char dist, double*** celldata, int clusterid[][2]);
```

Arguments

- `int` *`nrows`*`;`
  The number of rows in the data matrix, equal to the number of genes in the gene expression experiment.
- `int` *`ncolumns`*`;`
  The number of columns in the data matrix, equal to the number of microarrays in the gene expression experiment.
- `double**` *`data`*`;`
  The data array containing the gene expression data. Genes are stored row-wise, while microarrays are stored column-wise. Dimension: `[nrows][ncolumns]`.
- `int**` *`mask`*`;`
  This array shows which elements in the *data* array, if any, are missing. If `mask[i][j]==0`, then `data[i][j]` is missing. Dimension: `[nrows][ncolumns]`.
- `double` *`weight`*`[];`
  The weights that are used to calculate the distance. Dimension: `[ncolumns]` if `transpose==0`; `[nrows]` if `transpose==1`.
- `int` *`transpose`*`;`
  This flag indicates whether row-wise (gene) or column-wise (microarray) clustering is being performed. If `transpose==0`, rows (genes) are being clustered. Otherwise, columns (microarrays) are being clustered.
- `int` *`nxgrid`*`;`
  The number of cells horizontally in the rectangular topology containing the clusters.
- `int` *`nygrid`*`;`
  The number of cells vertically in the rectangular topology containing the clusters.
- `double` *`inittau`*`;`
  The initial value for the parameter τ that is used in the SOM algorithm. A typical value for *inittau* is 0.02,
- `int` *`niter`*`;`
  The total number of iterations.
- `char` *`dist`*`;`
  Specifies which distance measure is used.
- `double***` *`celldata`*`;`
  The data vectors of the clusters in the rectangular topology that were found by the SOM algorithm. These correspond to the cluster centroids. The first dimension is the horizontal position of the cluster in the rectangle, the second dimension is the vertical position of the cluster in the rectangle, while the third dimension is the dimension along the data vector. The `somcluster` routine does not allocate storage space for the *celldata* array. Space should be allocated before calling `somcluster`. Alternatively, if *celldata* is equal to `NULL`, the `somcluster` routine allocates space for *celldata* and frees it before returning. In that case, `somcluster` does not return the data vectors of the clusters that were found. Dimension: `[nxgrid][nygrid][ncolumns]` if `transpose==0`, or `[nxgrid][nygrid][nrows]` if `transpose==1`.
- `int` *`clusterid`*`[][2];`
  Specifies the cluster to which a gene or microarray was assigned, using two integers

to identify the horizontal and vertical position of a cell in the grid for each gene or microarray. Gene or microarrays are assigned to clusters in the rectangular grid by determining which cluster in the rectangular topology has the closest data vector. Space for the *c1usterid* argument should be allocated before calling somcluster. If *c1usterid* is NULL, the somcluster routine ignores this argument and does not return the cluster assignments. Dimension: *[nrows]* if *transpose* ==0; [ncolumns] if *transpose* ==1.

4.23 Principal Component Analysis

Principal Component Analysis (PCA) is a widely used technique for analyzing multivariate data. In PCA, the data vectors are written as a linear sum over principal components. The number of principal components is equal to the number of dimensions of the data vectors.

The principal components are chosen such that they maximally explain the variance in the data vectors. For example, in case of 3D data vectors, the data can be represented as an ellipsoidal cloud of points in three dimensional spaces. The first principal component would be the longest axis of the ellipsoid, the second principal component would be the second longest axis of the ellipsoid, and the third principal component would be the shortest axis. In other words, the principal components are ordered by the amount of variance they explain.

Each data point can be reconstructed by a suitable linear combination of the principal components. However, in order to reduce the dimensionality of the data, usually only the most important principal components are used. The remaining variance present in the data is then regarded as unexplained variance. The principal components can be found by calculating the eigenvectors of the covariance matrix of the data. The corresponding eigenvalues determine how much of the variance present in the data is explained by each principal component.

The eigenvectors are found by calculating the singular value decomposition of the data matrix. For this purpose, we have included a routine to calculate the singular value de-composition of a matrix.

Prototype

`void svd (int nrows, int ncolumns, double** U, double S[], double** V, int* ierror);`
calculates the singular value decomposition $U_{\text{input}} = U_{\text{output}} \cdot S \cdot V^{T}$. Householder bidiagonalization and a variant of the QR algorithm are used.

Arguments

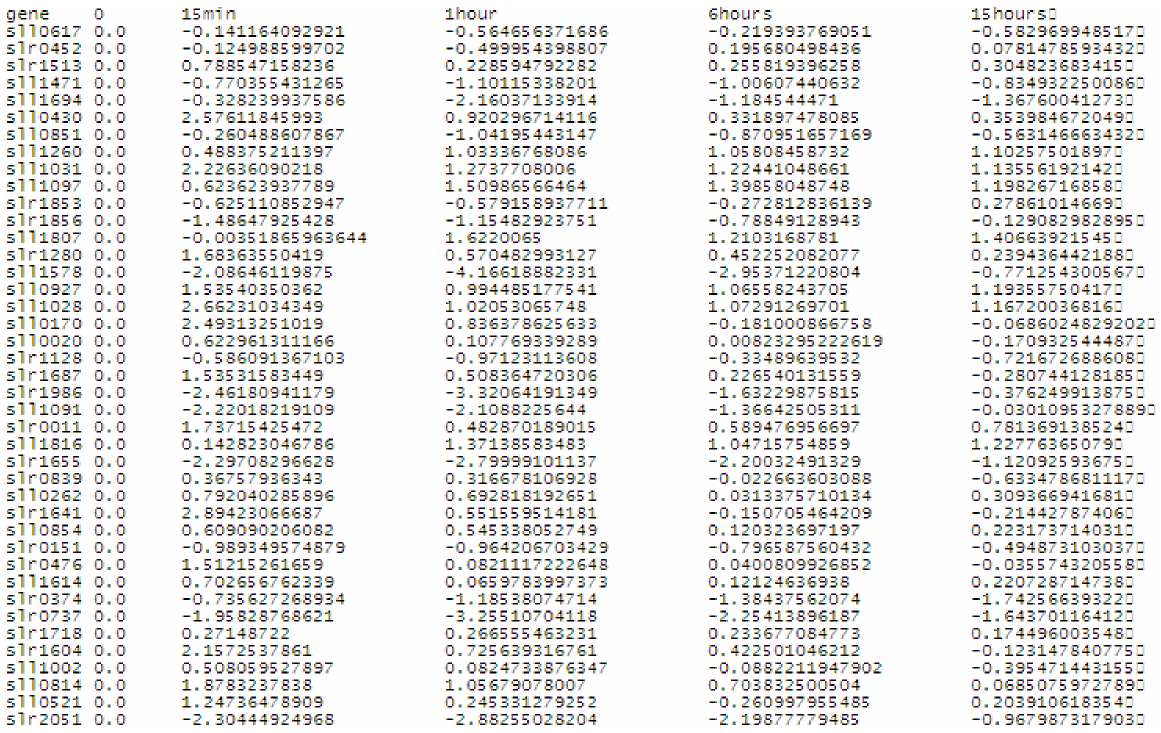
- `int nrows`
  The number of rows in U. The number of rows should be greater than or equal to the number of columns.
- `int ncolumns`
  The number of columns in U and the order of V.
- `double** U;`
  On input: U is the rectangular matrix to be decomposed. On output: The contents of U is replaced such that $U_{\text{input}} = U_{\text{output}} \cdot S \cdot V^{T}$. Dimension: `[nrows][ncolumns]`.
- `double* S;`
  The *ncolumns* non-negative singular values of the matrix U, unordered. If an error exit is made, the singular values should be correct for indices `{ *ierror, *ierror+1, ..., ncolumns-1}`. Dimension: `[ncolumns]`.

- `double** V;`
  The orthogonal matrix V of the decomposition. If an error exit is made, the columns of V corresponding to indices of correct singular values should be correct. Dimension: `[ncolumns][ncolumns]`.
- `int* ierror;`
  Error exit: `*ierror==0` for normal return, and `*ierror==k` if the k^{th} singular value has not been determined after 30 iterations.

Chapter 5
Experimental result

Clustering technique provide a computational environment for analyzing data from microarray experiments, or other genomic datasets. Clustering program organizes and analyzes the data in a number of different ways. The first step in using Cluster is to import data. Cluster input tables rows represent genes and columns represent samples or observations. For a simple timecourse, a Cluster input file would look like this:

5.1 Input file:



```
gene    0       15min                   1hour                   6hours                  15hours□
sll0617 0.0     -0.141164092921         -0.564656371686         -0.219393769051         -0.582969948517□
slr0452 0.0     -0.124988599702         -0.499954398807         0.195680498436          0.0781478593432□
slr1513 0.0     0.788547158236          0.228594792282          0.255819396258          0.304823683415□
sll1471 0.0     -0.770355431265         -1.10115338201          -1.00607440632          -0.834932250086□
sll1694 0.0     -0.328239937586         -2.16037133914          -1.184544471            -1.36760041273□
sll0430 0.0     2.57611845993           0.920296714116          0.331897478085          0.35398467204 9□
sll0851 0.0     -0.260488607867         -1.04195443147          -0.870951657169         -0.563146663432□
sll1260 0.0     0.488375211397          1.03336768086           1.05808458732           1.10257501897□
sll1031 0.0     2.22636090218           1.2737708006            1.22441048661           1.13556192142□
sll1097 0.0     0.623623937789          1.50986566464           1.39858048748           1.19826716858□
slr1853 0.0     -0.625110852947         -0.579158937711         -0.272812836139         0.27861014669□
slr1856 0.0     -1.48647925428          -1.15482923751          -0.78849128943          -0.129082982895□
sll1807 0.0     -0.00351865963644       1.6220065               1.2103168781            1.40663921545□
slr1280 0.0     1.68363550419           0.570482993127          0.452252082077          0.239436442188□
sll1578 0.0     -2.08646119875          -4.16618882331          -2.95371220804          -0.771254300567□
sll0927 0.0     1.53540350362           0.994485177541          1.06558243705           1.19355750417□
sll1028 0.0     2.66231034349           1.02053065748           1.07291269701           1.16720036816□
sll0170 0.0     2.49313251019           0.836378625633          -0.181000866758         -0.0686024829202□
sll0020 0.0     0.622961311166          0.107769339289          0.00823295222619        -0.170932544487□
slr1128 0.0     -0.586091367103         -0.97123113608          -0.33489639532          -0.721672688608□
slr1687 0.0     1.53531583449           0.508364720306          0.226540131559          -0.280744128185□
slr1986 0.0     -2.46180941179          -3.32064191349          -1.63229875815          -0.376249913875□
sll1091 0.0     -2.22018219109          -2.1088225644           -1.36642505311          -0.0301095327889□
slr0011 0.0     1.73715425472           0.482870189015          0.589476956697          0.781369138524□
sll1816 0.0     0.142823046786          1.37138583483           1.04715754859           1.22776365079□
slr1655 0.0     -2.29708296628          -2.7999101137           -2.20032491329          -1.12092593675□
slr0839 0.0     0.36757936343           0.316678106928          -0.022663603088         -0.633478681117□
sll0262 0.0     0.792040285896          0.692818192651          0.0313375710134         0.309366941681□
slr1641 0.0     2.89423066687           0.551559514181          -0.150705464209         -0.21442787406□
sll0854 0.0     0.609090206082          0.545338052749          0.12032369719 7         0.223173714031□
slr0151 0.0     -0.989349574879         -0.964206703429         -0.796587560432         -0.494873103037□
slr0476 0.0     1.51215261659           0.0821117222648         0.0400809926852         -0.0355743205558□
sll1614 0.0     0.702656762339          0.0659783997373         0.12124636938           0.220728714738□
slr0374 0.0     -0.735627268934         -1.18538074714          -1.38437562074          -1.74256639322□
slr0737 0.0     -1.95828768621          -3.25510704118          -2.25413896187          -1.64370116412□
slr1718 0.0     0.27148722              0.266555463231          0.233677084773          0.174496003548□
slr1604 0.0     2.1572537861            0.725639316761          0.422501046212          -0.123147840775□
sll1002 0.0     0.508059527897          0.0824733876347         -0.0882211947902        -0.395471443155□
sll0814 0.0     1.8783237838            1.05679078007           0.703832500504          0.0685075972789□
sll0521 0.0     1.24736478909           0.245331279252          -0.260997955485         0.203910618354□
slr2051 0.0     -2.30444924968          -2.88255028204          -2.19877779485          -0.967987317903□
```

Each row (gene) has an identifier that always goes in the first column. Each column (sample) has a label that is always in the first row; here the labels describe the time at which a sample was taken. The remaining cells in the table contain data for the appropriate gene and sample. Missing values are acceptable and are designated by empty cells.

5.2 Hierarchical Clustering:

The Hierarchical Clustering allows you to perform hierarchical clustering on your data. This is an incredibly powerful and useful method for analyzing all sorts of large genomic datasets.

Cluster currently performs four types of binary, average, complete, and centroid clustering. The basic idea is to assemble a set of items (genes or arrays) into a tree, where items are joined by very short branches if they are very similar to each other, and by increasingly longer branches as their similarity decreases.

Similarities/Distances:

The first choice that must be made is how “similarity” is to be defined. There are many ways to compute how similar two series of numbers are, and Cluster provides a small number of options. The most commonly used similarity metrics are based on Pearson correlation. The Pearson correlation coefficient between any two series of number

X={ X , X , , X N 1 2 K } and Y={ N Y ,Y , ,Y 1 2 K } is defined as

$$r = \frac{1}{N}\sum_{i=1,N}\left(\frac{X_i - \overline{X}}{\sigma_X}\right)\left(\frac{Y_i - \overline{Y}}{\sigma_Y}\right)$$

where X is the average of values in X, and X is the standard deviation of these values.

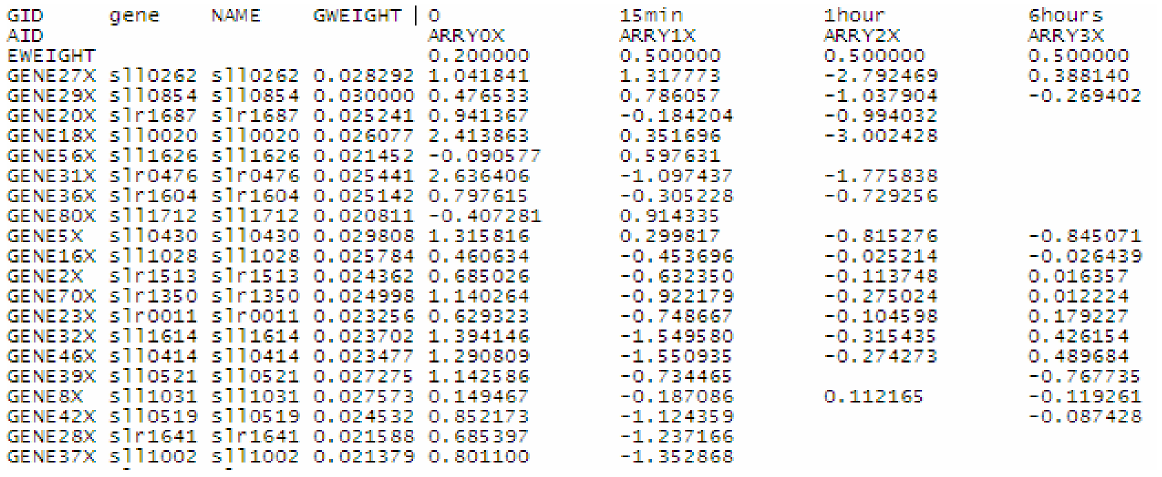
There are many ways of conceptualizing the correlation coefficient. Cluster provides two similarity metrics that are the absolute value of these two correlation functions, which consider two items to be similar if they have opposite expression patterns; the standard correlation coefficients consider opposite genes are being very distant.

Clustering process:

With any specified metric, the first step in the clustering process is to compute the distance between of all pairs of items to be clustered (e.g. the set of genes in the current dataset). Once this matrix of distances is computed, the clustering begins. The process used by Cluster is agglomerative hierarchical processing, which consists of repeated cycles where the two closest remaining items (those with the smallest distance) are joined by a node/branch of a tree, with the length of the branch set to the distance between the joined items. The two joined items are removed from list of items being processed replaced by a item that represents the new branch. The distances between this new item and all other remaining items are computed, and the process is repeated until only one item remains.

Output File:

The result of a clustering run is a tree or pair of trees (one for genes one for arrays). When Cluster joins two items, it randomly places one item on the top branch and the other on the bottom branch. It is possible to guide this process to generate the “best” ordering consistent with a given tree. This is done by using the GORDER (gene order) and EORDER (experiment order) parameters in the input file,



```
GID      gene    NAME    GWEIGHT | 0               15min           1hour           6hours
AID                                ARRY0X          ARRY1X          ARRY2X          ARRY3X
EWEIGHT                            0.200000        0.500000        0.500000        0.500000
GENE27X sll0262 sll0262 0.028292 1.041841        1.317773        -2.792469       0.388140
GENE29X sll0854 sll0854 0.030000 0.476533        0.786057        -1.037904       -0.269402
GENE20X slr1687 slr1687 0.025241 0.941367        -0.184204       -0.994032
GENE18X sll0020 sll0020 0.026077 2.413863        0.351696        -3.002428
GENE56X sll1626 sll1626 0.021452 -0.090577       0.597631
GENE31X slr0476 slr0476 0.025441 2.636406        -1.097437       -1.775838
GENE36X slr1604 slr1604 0.025142 0.797615        -0.305228       -0.729256
GENE80X sll1712 sll1712 0.020811 -0.407281       0.914335
GENE5X  sll0430 sll0430 0.029808 1.315816        0.299817        -0.815276       -0.845071
GENE16X sll1028 sll1028 0.025784 0.460634        -0.453696       -0.025214       -0.026439
GENE2X  slr1513 slr1513 0.024362 0.685026        -0.632350       -0.113748       0.016357
GENE70X slr1350 slr1350 0.024998 1.140264        -0.922179       -0.275024       0.012224
GENE23X slr0011 slr0011 0.023256 0.629323        -0.748667       -0.104598       0.179227
GENE32X sll1614 sll1614 0.023702 1.394146        -1.549580       -0.315435       0.426154
GENE46X sll0414 sll0414 0.023477 1.290809        -1.550935       -0.274273       0.489684
GENE39X sll0521 sll0521 0.027275 1.142586        -0.734465                       -0.767735
GENE8X  sll1031 sll1031 0.027573 0.149467        -0.187086       0.112165        -0.119261
GENE42X sll0519 sll0519 0.024532 0.852173        -1.124359                       -0.087428
GENE28X slr1641 slr1641 0.021588 0.685397        -1.237166
GENE37X sll1002 sll1002 0.021379 0.801100        -1.352868
```

Array group

```
NODE1X  ARRY1X  ARRY0X  1.000000
NODE2X  ARRY2X  NODE1X  0.261253
NODE3X  ARRY3X  NODE2X  0.140240
NODE4X  ARRY4X  NODE3X  -0.017618
```

Gene group

```
NODE1X  GENE12X GENE10X 1.000000
NODE2X  GENE72X GENE24X 1.000000
NODE3X  GENE73X GENE26X 1.000000
NODE4X  GENE80X GENE36X 1.000000
NODE5X  GENE1X  GENE0X  1.000000
NODE6X  GENE3X  NODE5X  1.000000
```

```
NODE7X  GENE84X NODE6X  1.000000
NODE8X  GENE83X NODE7X  1.000000
NODE9X  GENE82X NODE8X  1.000000
NODE10XGENE4X   NODE9X  1.000000
NODE11XGENE6X   NODE10X1.000000
NODE12XGENE79XNODE11X1.000000
NODE13XGENE78XNODE12X1.000000
NODE14XGENE11XNODE13X1.000000
NODE15XGENE76XNODE14X1.000000
```

5.3 K-mean Clustering:

K-means clustering is a simple, but popular, form of cluster analysis. The basic idea is that you start with a collection of items (e.g. genes) and some chosen number of clusters (k) you want to find. The items are initially randomly assigned to a cluster. K-means clustering proceeds by repeated application of a two-step process where:

1. The mean vector for all items in each cluster is computed
2. Items are reassigned to the cluster whose center is closest to the item

The parameters that control k-means clustering are

1. The number of clusters (K)
2. The maximum number of cycles

The output is simply an assignment of items to a cluster. The implementation here simply rearranges the rows and/or columns based on which cluster they were assigned to in the final cycle. Cluster also implements a slight variation on k-means clustering known as k-mediod clustering in which the median instead of the mean of items in a node are used.

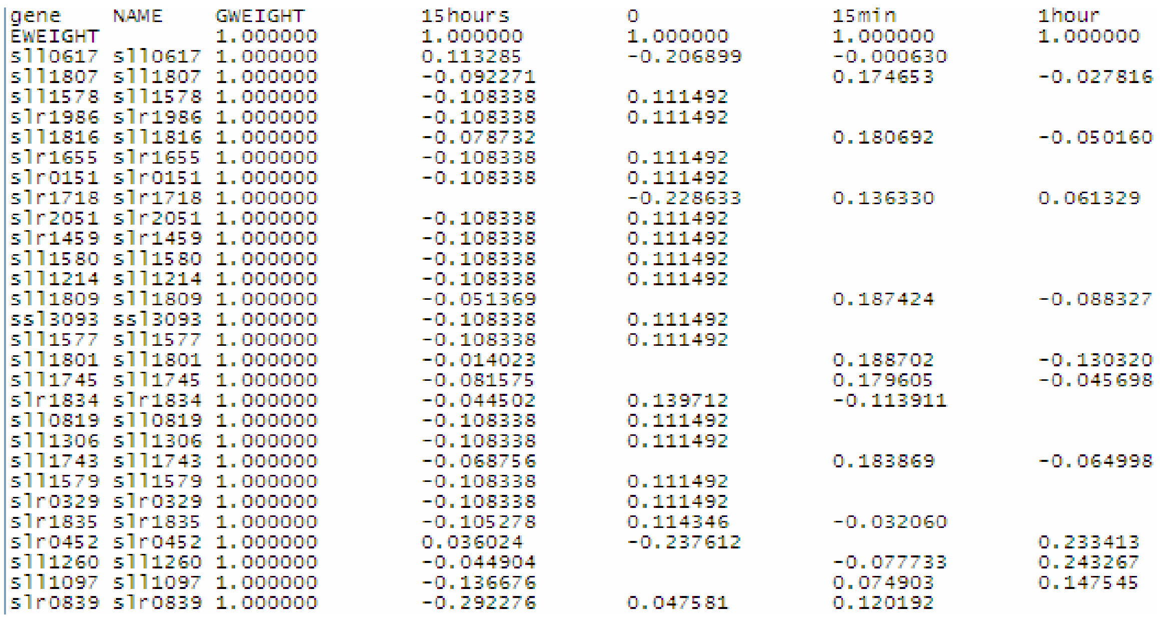
Output files

```
ARRAY    GROUP
15hours  0
0                 1
15min             2
1hour             3
6hours            4



GENE     GROUP
sll0617  0
sll1807  0
```

sll1743	0
sll1579	0
slr0329	0
slr1835	0
slr0452	1
sll1260	1
sll1097	1
slr0839	1
sll0144	2
sll0680	2
slr1350	2
sll0901	2
sll1327	2
sll1712	2
sll1694	3
sll0851	3
slr1128	3
sll0262	3
sll0854	3
slr0374	3
sll1802	3
sll0416	3
sll0519	4
ssl1533	4
sll1029	4
slr1963	4
slr0642	4
slr0208	4



gene	NAME	GWEIGHT	15hours	0	15min	1hour
EWEIGHT		1.000000	1.000000	1.000000	1.000000	1.000000
sll0617	sll0617	1.000000	0.113285	-0.206899	-0.000630	
sll1807	sll1807	1.000000	-0.092271		0.174653	-0.027816
sll1578	sll1578	1.000000	-0.108338	0.111492		
slr1986	slr1986	1.000000	-0.108338	0.111492		
sll1816	sll1816	1.000000	-0.078732		0.180692	-0.050160
slr1655	slr1655	1.000000	-0.108338	0.111492		
slr0151	slr0151	1.000000	-0.108338	0.111492		
slr1718	slr1718	1.000000		-0.228633	0.136330	0.061329
slr2051	slr2051	1.000000	-0.108338	0.111492		
slr1459	slr1459	1.000000	-0.108338	0.111492		
sll1580	sll1580	1.000000	-0.108338	0.111492		
sll1214	sll1214	1.000000	-0.108338	0.111492		
sll1809	sll1809	1.000000	-0.051369		0.187424	-0.088327
ssl3093	ssl3093	1.000000	-0.108338	0.111492		
sll1577	sll1577	1.000000	-0.108338	0.111492		
sll1801	sll1801	1.000000	-0.014023		0.188702	-0.130320
sll1745	sll1745	1.000000	-0.081575		0.179605	-0.045698
slr1834	slr1834	1.000000	-0.044502	0.139712	-0.113911	
sll0819	sll0819	1.000000	-0.108338	0.111492		
sll1306	sll1306	1.000000	-0.108338	0.111492		
sll1743	sll1743	1.000000	-0.068756		0.183869	-0.064998
sll1579	sll1579	1.000000	-0.108338	0.111492		
slr0329	slr0329	1.000000	-0.108338	0.111492		
slr1835	slr1835	1.000000	-0.105278	0.114346	-0.032060	
slr0452	slr0452	1.000000	0.036024	-0.237612		0.233413
sll1260	sll1260	1.000000	-0.044904		-0.077733	0.243267
sll1097	sll1097	1.000000	-0.136676		0.074903	0.147545
slr0839	slr0839	1.000000	-0.292276	0.047581	0.120192	

5.4 Self-Organizing Maps:

Self-Organizing Maps (SOMs) is a method of cluster analysis that are somewhat related to k-means clusterins. SOMs were invented in by Teuvo Kohonen in the early 1980s, and have recently been used in genomic analysis (see Chu 1998, Tamayo 1999 and Golub 1999 in references). The Tamayo paper contains a simple explanation of the methods. A more detailed description is available in the book by Kohonen, Self-Organizing Maps1997.

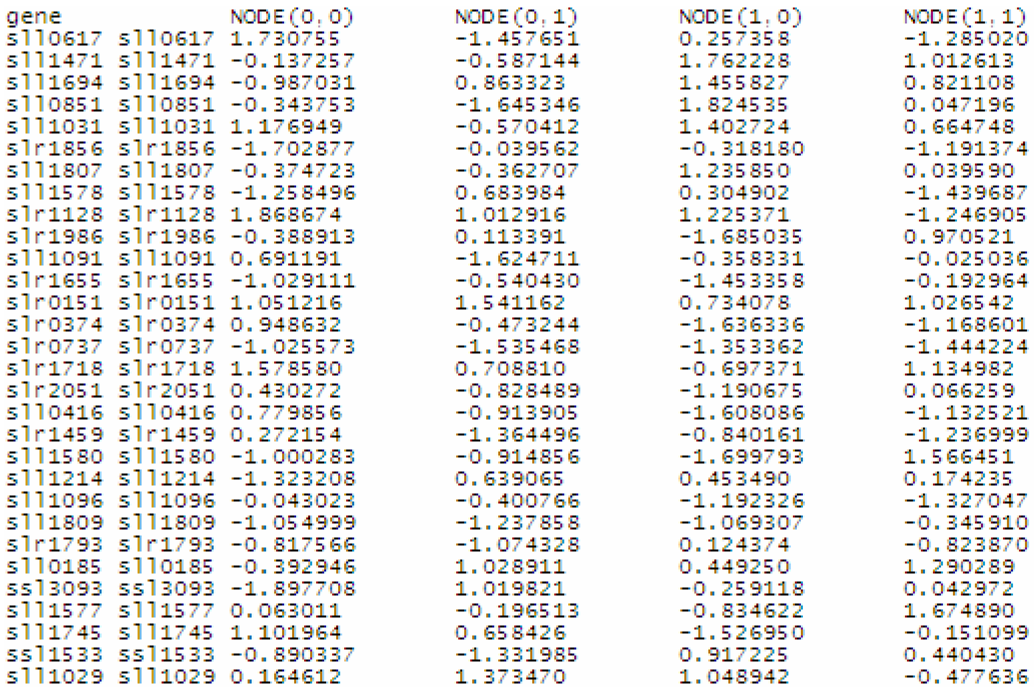
The current implementation varies slightly from that of Tamayo et al., in that it restricts the analysis one-dimensional SOMs along each axis, as opposed to a two-dimensional network. The one-dimensional SOM is used to reorder the elements on whichever axes are selected. The result is similar to the result of k-means clustering, except that, unlike kmeans, the nodes in a SOM are ordered. This tends to result in a relatively smooth transition between groups.

The options for SOMs are

1. Whether or not you will organize each axis,
2. The number of nodes for each axis (the default is the square-root of the number of items) and the number of iterations to be run.

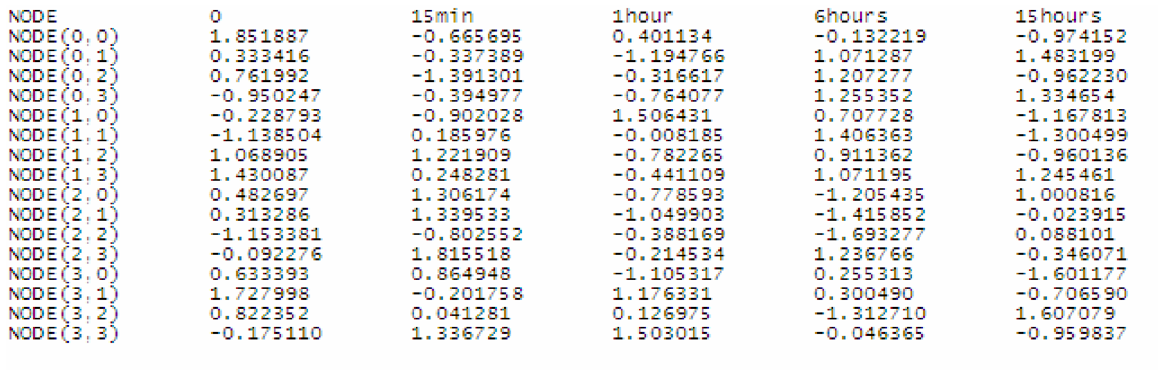
Output file

Array group:

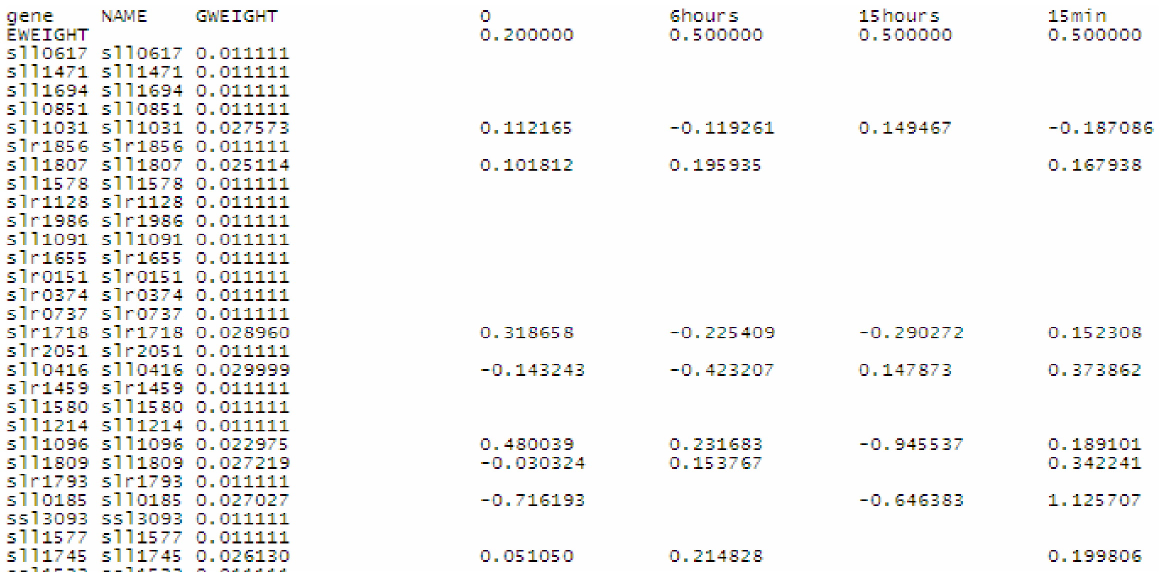


```
gene            NODE(0,0)       NODE(0,1)       NODE(1,0)       NODE(1,1)
sll0617 sll0617 1.730755        -1.457651       0.257358        -1.285020
sll1471 sll1471 -0.137257       -0.587144       1.762228        1.012613
sll1694 sll1694 -0.987031       0.863323        1.455827        0.821108
sll0851 sll0851 -0.343753       -1.645346       1.824535        0.047196
sll1031 sll1031 1.176949        -0.570412       1.402724        0.664748
slr1856 slr1856 -1.702877       -0.039562       -0.318180       -1.191374
sll1807 sll1807 -0.374723       -0.362707       1.235850        0.039590
sll1578 sll1578 -1.258496       0.683984        0.304902        -1.439687
slr1128 slr1128 1.868674        1.012916        1.225371        -1.246905
slr1986 slr1986 -0.388913       0.113391        -1.685035       0.970521
sll1091 sll1091 0.691191        -1.624711       -0.358331       -0.025036
slr1655 slr1655 -1.029111       -0.540430       -1.453358       -0.192964
slr0151 slr0151 1.051216        1.541162        0.734078        1.026542
slr0374 slr0374 0.948632        -0.473244       -1.636336       -1.168601
slr0737 slr0737 -1.025573       -1.535468       -1.353362       -1.444224
slr1718 slr1718 1.578580        0.708810        -0.697371       1.134982
slr2051 slr2051 0.430272        -0.828489       -1.190675       0.066259
sll0416 sll0416 0.779856        -0.913905       -1.608086       -1.132521
slr1459 slr1459 0.272154        -1.364496       -0.840161       -1.236999
sll1580 sll1580 -1.000283       -0.914856       -1.699793       1.566451
sll1214 sll1214 -1.323208       0.639065        0.453490        0.174235
sll1096 sll1096 -0.043023       -0.400766       -1.192326       -1.327047
sll1809 sll1809 -1.054999       -1.237858       -1.069307       -0.345910
slr1793 slr1793 -0.817566       -1.074328       0.124374        -0.823870
sll0185 sll0185 -0.392946       1.028911        0.449250        1.290289
ssl3093 ssl3093 -1.897708       1.019821        -0.259118       0.042972
sll1577 sll1577 0.063011        -0.196513       -0.834622       1.674890
sll1745 sll1745 1.101964        0.658426        -1.526950       -0.151099
ssl1533 ssl1533 -0.890337       -1.331985       0.917225        0.440430
sll1029 sll1029 0.164612        1.373470        1.048942        -0.477636
```

Gene group:



```
NODE          0           15min       1hour       6hours      15hours
NODE(0,0)     1.851887    -0.665695   0.401134    -0.132219   -0.974152
NODE(0,1)     0.333416    -0.337389   -1.194766   1.071287    1.483199
NODE(0,2)     0.761992    -1.391301   -0.316617   1.207277    -0.962230
NODE(0,3)     -0.950247   -0.394977   -0.764077   1.255352    1.334654
NODE(1,0)     -0.228793   -0.902028   1.506431    0.707728    -1.167813
NODE(1,1)     -1.138504   0.185976    -0.008185   1.406363    -1.300499
NODE(1,2)     1.068905    1.221909    -0.782265   0.911362    -0.960136
NODE(1,3)     1.430087    0.248281    -0.441109   1.071195    1.245461
NODE(2,0)     0.482697    1.306174    -0.778593   -1.205435   1.000816
NODE(2,1)     0.313286    1.339533    -1.049903   -1.415852   -0.023915
NODE(2,2)     -1.153381   -0.802552   -0.388169   -1.693277   0.088101
NODE(2,3)     -0.092276   1.815518    -0.214534   1.236766    -0.346071
NODE(3,0)     0.633393    0.864948    -1.105317   0.255313    -1.601177
NODE(3,1)     1.727998    -0.201758   1.176331    0.300490    -0.706590
NODE(3,2)     0.822352    0.041281    0.126975    -1.312710   1.607079
NODE(3,3)     -0.175110   1.336729    1.503015    -0.046365   -0.959837
```



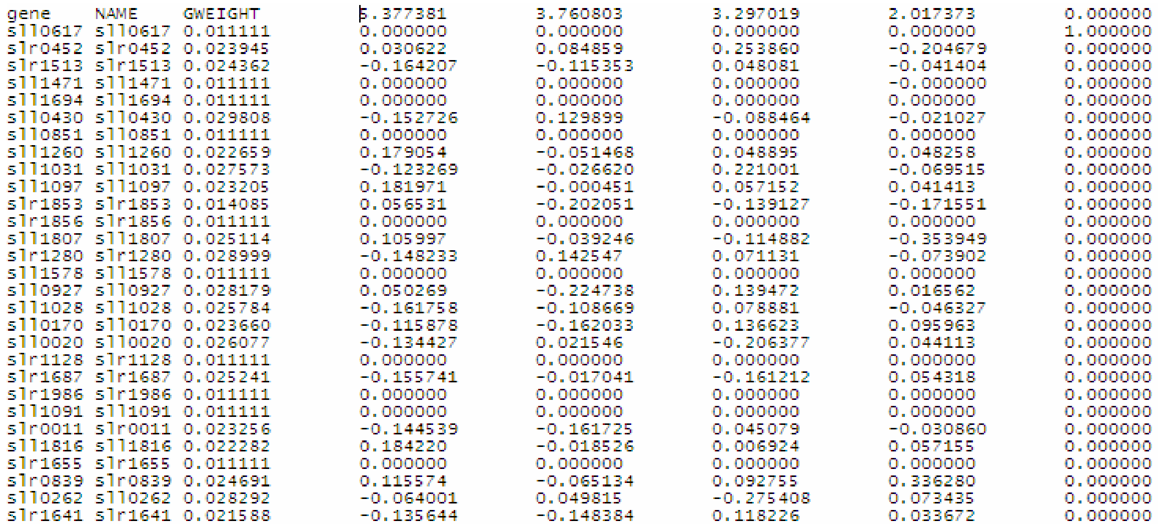
```
gene    NAME    GWEIGHT                 0               6hours          15hours         15min
EWEIGHT                                 0.200000        0.500000        0.500000        0.500000
sll0617 sll0617 0.011111
sll1471 sll1471 0.011111
sll1694 sll1694 0.011111
sll0851 sll0851 0.011111
sll1031 sll1031 0.027573                0.112165        -0.119261       0.149467        -0.187086
slr1856 slr1856 0.011111
sll1807 sll1807 0.025114                0.101812        0.195935                        0.167938
sll1578 sll1578 0.011111
slr1128 slr1128 0.011111
slr1986 slr1986 0.011111
sll1091 sll1091 0.011111
slr1655 slr1655 0.011111
slr0151 slr0151 0.011111
slr0374 slr0374 0.011111
slr0737 slr0737 0.011111
slr1718 slr1718 0.028960                0.318658        -0.225409       -0.290272       0.152308
slr2051 slr2051 0.011111
sll0416 sll0416 0.029999                -0.143243       -0.423207       0.147873        0.373862
slr1459 slr1459 0.011111
sll1580 sll1580 0.011111
sll1214 sll1214 0.011111
sll1096 sll1096 0.022975                0.480039        0.231683        -0.945537       0.189101
sll1809 sll1809 0.027219                -0.030324       0.153767                        0.342241
slr1793 slr1793 0.011111
sll0185 sll0185 0.027027                -0.716193                       -0.646383       1.125707
ssl3093 ssl3093 0.011111
sll1577 sll1577 0.011111
sll1745 sll1745 0.026130                0.051050        0.214828                        0.199806
```

5.5 Principal Component Analysis:

Cluster will perform principal component analysis on data. The output is very simple in this version and consists of two files that contain the principal components and the loadings of each gene on the principal components.

Output files:

EIGVALUE	0	15min	1hour	6hours	15hours
5.377381	0.000000	-0.805762	0.387468	0.328947	0.303991
3.760803	0.000000	0.021068	0.649631	-0.011371	-0.759873
3.297019	0.000000	-0.133927	-0.519809	0.708132	-0.458705
2.017373	0.000000	-0.576513	-0.397048	-0.624671	-0.346081
0.000000	1.000000	0.000000	0.000000	0.000000	0.000000



```
gene    NAME    GWEIGHT         5.377381        3.760803        3.297019        2.017373        0.000000
sll0617 sll0617 0.011111        0.000000        0.000000        0.000000        0.000000        1.000000
slr0452 slr0452 0.023945        0.030622        0.084859        0.253860        -0.204679       0.000000
slr1513 slr1513 0.024362        -0.164207       -0.115353       0.048081        -0.041404       0.000000
sll1471 sll1471 0.011111        0.000000        0.000000        0.000000        -0.000000       0.000000
sll1694 sll1694 0.011111        0.000000        0.000000        0.000000        0.000000        0.000000
sll0430 sll0430 0.029808        -0.152726       0.129899        -0.088464       -0.021027       0.000000
sll0851 sll0851 0.011111        0.000000        0.000000        0.000000        0.000000        0.000000
sll1260 sll1260 0.022659        0.179054        -0.051468       0.048895        0.048258        0.000000
sll1031 sll1031 0.027573        -0.123269       -0.026620       0.221001        -0.069515       0.000000
sll1097 sll1097 0.023205        0.181971        -0.000451       0.057152        0.041413        0.000000
slr1853 slr1853 0.014085        0.056531        -0.202051       -0.139127       -0.171551       0.000000
slr1856 slr1856 0.011111        0.000000        0.000000        0.000000        0.000000        0.000000
sll1807 sll1807 0.025114        0.105997        -0.039246       -0.114882       -0.353949       0.000000
slr1280 slr1280 0.028999        -0.148233       0.142547        0.071131        -0.073902       0.000000
sll1578 sll1578 0.011111        0.000000        0.000000        0.000000        0.000000        0.000000
sll0927 sll0927 0.028179        0.050269        -0.224738       0.139472        0.016562        0.000000
sll1028 sll1028 0.025784        -0.161758       -0.108669       0.078881        -0.046327       0.000000
sll0170 sll0170 0.023660        -0.115878       -0.162033       0.136623        0.095963        0.000000
sll0020 sll0020 0.026077        -0.134427       0.021546        -0.206377       0.044113        0.000000
slr1128 slr1128 0.011111        0.000000        0.000000        0.000000        0.000000        0.000000
slr1687 slr1687 0.025241        -0.155741       -0.017041       -0.161212       0.054318        0.000000
slr1986 slr1986 0.011111        0.000000        0.000000        0.000000        0.000000        0.000000
sll1091 sll1091 0.011111        0.000000        0.000000        0.000000        0.000000        0.000000
slr0011 slr0011 0.023256        -0.144539       -0.161725       0.045079        -0.030860       0.000000
sll1816 sll1816 0.022282        0.184220        -0.018526       0.006924        0.057155        0.000000
slr1655 slr1655 0.011111        0.000000        0.000000        0.000000        0.000000        0.000000
slr0839 slr0839 0.024691        0.115574        -0.065134       0.092755        0.336280        0.000000
sll0262 sll0262 0.028292        -0.064001       0.049815        -0.275408       0.073435        0.000000
slr1641 slr1641 0.021588        -0.135644       -0.148384       0.118226        0.033672        0.000000
```

Chapter 6
Applications

The various applications of clustering techniques are listed below:

6.1 Biology

In biology have two main applications in the fields of computational biology and bioinformatics.

- In transcriptomics, clustering is used to build groups of genes with related expression patterns. Often such groups contain functionally related proteins, such as enzymes for a specific pathway, or genes that are co-regulated. High throughput experiments using expressed sequence tags (ESTs) or DNA microarrays can be a powerful tool for genome annotation, a general aspect of genomics.
- In sequence analysis, clustering is used to group homologous sequences into gene families. This is a very important concept in bioinformatics, and evolutionary biology in general. See evolution by gene duplication.

6.2 Marketing research

Cluster analysis is widely used in market research when working with multivariate data from surveys and test panels. Market researchers use cluster analysis methods to partition the general population of consumers into market segments and to better understand the relationships between different groups of consumers/potential customers.

- Segmenting the market and determining target markets
- Product positioning
- New product development
- Selecting test markets

6.3 Other applications

Social network analysis: In the study of social networks, clustering may be used to recognize communities within large groups of people.

Image segmentation: Clustering can be used to divide a digital image into distinct regions for border detection or object recognition.

Data mining: Many data mining applications involve partitioning data items into related subsets; the marketing applications discussed above represent some examples. Another common application is the division of documents, such as World Wide Web pages, into genres.

Chapter 7
Conclusions

To summarize, a clustering algorithm can be defined by a set of objects (e.g. documents) and a vague description of the set, A. The goal of clustering is to divide the object set into objects belonging to A and a second set not in A. So, in this clustering problem, one needs to determine first what features are relevant in describing objects in A (intra-cluster similarity) and second, what features distinguish objects in A from objects not belonging to A (inter-cluster similarity).

Alternatively, a cluster problem can also be formulated by a set of objects and a similarity or distance function. Here, the object set is divided into a number of subsets (clusters) that best reveal the structure of the object set; these subdivisions can take the form of partitions or a hierarchically organized taxonomy.

The Clusters should be highly internally homogeneous i.e. members being similar to one another and highly externally heterogeneous as well i.e. members should be unlike members of other clusters. Clustering algorithms are distinguished on the basis of a number of features such as unsupervised or supervised; divisive or agglomerative; incremental or partitioning; iterative or noniterative; single link, grouped average, or complete link clustering. Interestingly, no clustering algorithm has been shown to be notably better than others when producing the same number of clusters. However, one can experience that some choices seem to fit some kinds of data better than others and there have been “bakeoffs” between clustering approaches (comparing single link, complete link, Ward’s trace, centroid, and so forth) that suggest that some approaches are more reliable than others for generic data sets.

The subject of clustering evolved with the early attempts to categorize objects or phenomena on the basis of inherent similarities. The application of clustering techniques started in the discipline of astronomy and medicine and gradually permeated to almost all the fields of study. But the most wonderful application of clustering techniques was found in the area of information retrieval and thus, document clustering.

In recent times, the clustering techniques have migrated from data mining into text mining. In the case of text mining, clustering is used to segment a document collection into subsets; where the members of each subset are similar with respect to certain features. The use of this technique makes text mining different from a simple search engine. As such, retrieval of text references is not a difficult task but the problem is of the large volume of retrieved documents; the sheer volume makes it difficult for users to find relevant information. While searching, the user naturally moves from one document to another looking for dominant themes or similar documents in a collection and the application of clustering helps in this process further. But, clustering tries to enforce a structure onto naturally unstructured documents, here, the challenge is to find a method that is simple and efficient and which provides enough structure to reveal interesting information.

Clustering depends on the discovery of some measure of interdocument similarity. In this area, one approach is to represent them as vectors of equal length; where each component of a vector is associated with one of the unique content works in the document collection. Now, the vector component may indicate the frequency, normalized or not, of a word in the document. Single linkage hierarchical clustering is a commonly used method for the purpose. However it is too slow for even moderately large document collections. The reason is beginning with individual documents, single-linkage hierarchical clustering iteratively agglomerates the most similar pairs of clusters into a new cluster. So, when it goes for global consideration of all pair wise similarities at each stage of clustering, then it leads to extensive computer run times.

Clustering has emerged as a full-fledged discipline from the days of its evolution as an application tool. There has been positive growth of the literature in the area of clustering since the day of its evolution as a mere categorization tool. Evolution of clustering is typical as it has maintained its unique status along with its interdisciplinary nature during the process. It has been widely applied in the areas of science, engineering, health care, education, social sciences and business. In the recent years, there has been a paradigm shift in research, as clustering is now applied for the purpose of information retrieval and document clustering on WWW. The subject has to still grow towards the realization of its full potential.

Chapter 8
Bibliography

1. Armanino, C., Lanteri, S., Forina, M., Balsamo, A., Migliardi, M. and Cenderelli, G. (1989) Hirsutism: A multivariate approach of feature selection and classification Chemometrics and Intelligent Laboratory Systems, 5, 335-341

2. Barnett, V., Kay, R. and Sneath, P. H. A. (1979) A familiar statistic in an Fuzzy Sets and Systems, 1, 111-127

3. Cheng, Yizong and Fu, King-Sun (1985) Conceptual clustering in knowledge organization IEEE Transactions on Pattern Analysis and Machine Intelligence, 7, 592-598

4. Grimson, R. C. (1979) The clustering of disease Mathematical Biosciences, 46, 257-278

Chapter 9

References

1. Southern EM. Detection of specific sequences among DNA fragments separated by gel electrophoresis. J Mol Biol. 1975 Nov 5;98(3):503-17.

2. D'haeseleer P, Liang S, Somogyi R. Genetic network inference: from co-expression clustering to reverse engineering. Bioinformatics. 2000 Aug;16(8):707-26.

3. Topics in Modelling of Clustered Data
Author : Marc Aerts, Geert Molenberghs, Louise M. Ryan
Pub. Date: May 2002
Publisher: CRC Press

4. Clustering and Information Retrieval
Author: Weili Wu, Hui Xiong, Shashi Shekhar (Editor)
Pub. Date: December 2003
Publisher: Kluwer Academic Publishers

5. Fuzzy Sets in Information Retrieval and Cluster Analysis
Author: Sadaakio Miyamoto
Pub. Date: May 1990
Publisher: Kluwer Academic Publishers

6. Mixture Models: Inference and Applications to Clustering
Author: Geoffrey J. McLachlan, K. E. Basford
Pub. Date: January 2001
Publisher: Marcel Dekker

7. Fodor SP, Read JL, Pirrung MC, Stryer L, Lu AT, Solas D. Light-directed, spatially addressable parallel chemical synthesis. Science. 1991 Feb 15;251(4995):767-73.

8. DeRisi JL, Iyer VR, Brown PO. Exploring the metabolic and genetic control of gene expression on a genomic scale. Science. 1997 Oct 24;278(5338):680-6

9. Fuzzy Sets and Their Application to Clustering and Training
Author: Beatrice Lazzerini (Editor), Lakhmi C. Jain
Pub. Date: March 2000
Publisher: CRC Press

10. Claverie JM. Computational methods for the identification of differential and coordinated gene expression. Hum Mol Genet. 1999;8(10):1821-32. Review.

11. Press W, Teukolsky SA, Vetterling WT, Flannery BP. Numerical Recipes in C. The Art of Scientific Computing. Second Edition. Cambridge University Press.

Hierarchical clustering

12. Guenther Gedina, Praktische Methodenlehre Hintergrund-Material, lecture notes for "Praktische Methodenlehre", WS 1999/2000, University of Osnabrueck, Germany

13. Doug Fisher, Optimization and Simplification of Hierarchical Clusterings, KDD-95:118-123.

14. Wen X, Fuhrman S, Michaels GS, Carr DB, Smith S, Barker JL, Somogyi R. Large-scale temporal gene expression mapping of central nervous system development. Proc Natl Acad Sci U S A. 1998 Jan 6;95(1):334- 9.

Self-organizing maps

15. Vesanto J. Usisng SOM in Data Mining. Licentiate's thesis. Helsinki University of Technology, Department of Computer Science and Engineering. 2000 Apr 10.

16. Vesanto J. SOM-Based Data Visualization Methods. Helsinki University of Technology, Laboratory of Computer and Information Science. 1999 Nov 19.

17. Vesanto J. Data Mining Techniques Based on the Self-Organizing Map. Thesis for degree of Master of Science in Engineering. Helsinki University of Technology, Department of Engineering Physics and Mathematics. 1997 May 26.

18. Kohonen T, Hynninen J, Kangas J, Laaksonen J. SOM_PAK. The Self-Organizing Map Program Package. Manual Version 3.1. Helsinki University of Technology. Laboratory of Computer and Information Science. 1995 April 7.

19. Tamayo P, Slonim D, Mesirov J, Zhu Q, Kitareewan S, Dmitrovsky E, Lander ES, Golub TR. Interpreting patterns of gene expression with self-organizing maps: methods and application to hematopoietic differentiation. Proc Natl Acad Sci U S A. 1999 Mar 16;96(6):2907-12.

k-means

20. Zhexue Huang. A fast clustering algorithm to cluster very large categorical data sets in data mining. In Proc. SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery, 1997.

21. Zhexue Huang. Clustering large data sets with mixed numerical and categorical values. Proceedings of the First Pacific-Asia Conference on Knowledge Discovery and Data Mining, Singapore, Word Scientific, 1997.

22. Tavazoie S, Hughes JD, Campbell MJ, Cho RJ, Church GM. Systematic determination of genetic network architecture. Nat Genet. 1999 Jul;22(3):281-5.

23. Soukas A, Cohen P, Socci ND, Friedman JM. Leptin-specific patterns of gene expression in white adipose tissue. Genes Dev. 2000 Apr 15;14(8):963-80.

Principal component analysis

24. Alexander Basilevsky, Statistical factor analysis and related methods, Theory and Applications. Wiley series in probability and mathematical statistics. John Wiley & Sons.

25. Raychaudhuri S, Stuart JM, Altman RB. Principal components analysis to summarize microarray experiments: application to sporulation time series. Pac Symp Biocomput. 2000;:455-66.

26. Holter NS, Mitra M, Maritan A, Cieplak M, Banavar JR, Fedoroff NV. Fundamental patterns underlying gene expression profiles: simplicity from complexity. Proc Natl Acad Sci U S A. 2000 Jul 18;97(15):8409-14.

27. Claverie JM. Computational methods for the identification of differential and coordinated gene expression. Hum Mol Genet. 1999;8(10):1821-32. Review.