Clustering Analysis of Microarray Gene Expression Data by Splitting Algorithm

Ruye Wang

Engineering Department
Harvey Mudd College
Claremont, CA 91711

Jet Propulsion Laboratory
M/S 126-347
4800 Oak Grove Dr.
Pasadena, CA 91109

Lucas Scharenbroich

Jet Propulsion Laboratory
M/S 126-347
4800 Oak Grove Dr.
Pasadena, CA 91109

Christopher Hart

California Institute of Technology
M/C 156-29
Pasadena, CA 91125

Barbara Wold

California Institute of Technology
M/C 156-29
Pasadena, CA 91125

Eric Mjolsness

University of California, Irvine
Institute for Genomics and Bioinformatics
School of Information & Computer Science
Irvine, CA 92697

Abstract

A clustering method based on recursive bisection is introduced for analyzing microarray gene expression data. Either or both dimensions for the genes and the samples of a given microarray dataset can be classi£ed in an unsupervised fashion. Alternatively, if certain prior knowledge of the genes or samples is available, a supervised version of the clustering analysis can also be carried out. Either approach may be used to generate a partial or complete binary hierarchy, the dendrogram, showing the underlying structure of the dataset. Compared to other existing clustering methods used for microarray data analysis (such as the hierarchical, K-means, and self-organizing map methods), the method presented here has the advantage of much improved computational ef£ciency while retaining effective separation of data clusters under a distance metric, a straightforward parallel implementation, and useful extraction and presentation of biological information. Clustering results of both synthesized and experimental microarray data are presented to demonstrate the performance of the algorithm.

1 Introduction

Various clustering analysis methods (both supervised and unsupervised), well known in the £elds of statistical pattern recognition and arti£cial neural networks, have been applied to the analysis of the microarray gene expression data. The typical unsupervised methods include bottom-up hierarchical clustering Eisen et al. (1998) and K-means clustering Tavazoie et al. (1999). All of these methods treat the genes or the samples in the microarray dataset as points (vectors) in a high dimensional feature space and classify them into clusters according to their locations and relevant distances (similarity) in that space.

In particular, the hierarchical clustering method, which is widely used in microarray data analysis, is a bottom-up process which keeps merging pairs of data points or groups of points closest to each other in the feature space until eventually all data points are merged into a single group. In addition, a tree structure, the dendrogram, can be obtained from the clustering process to present the hierarchical classi£cation graphically. While the hierarchy is generated automatically, the groups of genes need to be interpreted by a biologist to produce a biologically relevant clustering. From the computational point of view, the closest-£rst merging operation is inherently a sequential process.

Here, a top-down splitting approach for unsupervised clustering of the microarray data is presented. As with other clustering methods used for this purpose, the top-

Email addresses: ruye_wang@hmc.edu (Ruye Wang), Lucas.Scharenbroich@jpl.nasa.gov (Lucas Scharenbroich), hart@caltech.edu (Christopher Hart), woldb@caltech.edu (Barbara Wold), emj@uci.edu (Eric Mjolsness).

down partitioning approach has also been known in other £elds such as statistical pattern recognition, and it has been used for supervised clustering of microarray data (Brown et al. (2000), Xiong et al. (2000), Zhang et al. (2001), etc.). However, unlike these supervised approaches, which all make use of some prior knowledge of the genes or the samples assumed to be available, here it is assumed that other than the gene expression data in the microarray dataset, there is no additional information available about either the genes or the samples. The purpose of the analysis is simply to £nd the underlying structure of the dataset in terms of the similarities of the genes and/or the samples. In addition, similar to the hierarchical clustering approach, the top-down split algorithm can also generate a binary dendrogram tree to visually present the clustering results. This can be done for both the genes and the samples. Compared to other existing clustering methods, the top-down splitting algorithm possesses certain advantages including much improved ef£ciency. Moreover, computationally, the top-down splitting algorithm lends itself naturally to parallelization. The consecutive splitting operations can be carried out in parallel by different processing units of a multiprocessor computer system, thereby speeding up the computational time tremendously.

Principal component analysis (PCA) has been widely used in clustering analysis (Alter et al. (2000), S. Raychaudhuri (2000)). A small number of eigen-features can be obtained by the PCA method to represent most of the variability in the dataset, and thereby tremendously reduce the dimensionality of the feature space. For this reason, PCA is also widely used in the analysis of large-scale microarray gene data. A small number of eigen-genes can be obtained to represent the large number of genes, or, on the other hand, the genes can be represented by a space spanned by a small number of eigen-features. For example, PCA is used in Ben-Hur and Guyon (2003) to detect stability of clustering results.

However, in classi£cation (including clustering), a feature selection method (PCA or others) is more effective if it is adaptive to a small number (2 or 3) of classes instead of *all* classes. The features selected are most relevant to separating the few, most dissimilar classes, making the separation of less distinct classes easier at a later stage. This idea of adaptive PCA feature selection is implemented in the splitting algorithm presented here.

In the following, the basic methods of the top-down splitting algorithm will be discussed in detail in section 2, various advantages of the algorithm will be summarized in section 3, and the performance of the algorithm will be compared with that of other algorithms, in particular the bottom up hierarchical clustering method. Finally, in section 4, the clustering results generated by applying the top-down split algorithm to two experimental microarray data sets will be presented.

2 The top-down split clustering algorithm

It is assumed in the following that the expression levels of K genes from L samples (different experimental conditions, tissue samples, time courses, etc.) are obtained from the microarray data. The clustering analysis of this 2D data array can be carried out in either or both of the two dimensions. We can treat the genes as K data points (vectors) in an N-dimensional feature space $X_k = [x_1, \cdots, x_L]^T$ ($k = 1, \cdots, L$), or the samples as N data points in an L-dimensional feature space $Y_n = [y_1, \cdots, y_N]^T$ ($n = 1, \cdots, N$). In either case, the data points are partitioned recursively to form a set of clusters containing similar genes or similar samples, and a binary hierarchical structure is generated along the way to reveal the relationship (similarity and difference) between the clusters. This process can be carried out all the way until each cluster contains one data point and a complete dendrogram tree of the points is obtained. The clustering algorithm discussed below assumes each data point is a gene, although the discussion is equally valid for the clustering of the samples.

2.1 Top-down recursive bisection

Each bisection of a group of K data points is carried out in the following steps:

(1) **Obtain a subset of features**

The dimensions of a microarray are usually high (minimally, thousands of genes from tens of samples) for a clustering application. To reduce the dimensionality of the feature space for computational ef£ciency, an adaptive feature selection/extraction process is needed to obtain a small subset of features most relevant to a group of data points so that they can be optimally separated into two subgroups. To do so, we £rst obtain and sort the variances of all N features:

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_N$$

We then choose the M ($M < N$) features corresponding to the M largest variances to preserve most of the energy (representing information of separability) contained in the dataset. Speci£cally, M is so chosen that the ratio $\sum_{i=1}^{M} \sigma_i / \sum_{i=1}^{N} \sigma_i$ representing the percentage of energy preserved is greater than some speci£ed threshold (e.g., 90%).

(2) **Project data to a 1D space**

The partitioning is to be carried out in only one particular dimension corresponding to a certain feature, which could be either the best feature (with maximum variance σ) among the M original features chosen above, or a linear combination of these M features. Appending principal component analysis (PCA) method, we £nd the eigenvector corresponding to the largest eigen-

value of the covariance matrix of the data in the M-dimensional space:

$$\Sigma_x = \frac{1}{K}\sum_{i=1}^{K}(X_i - \mu_x)(X_i - \mu_x)^T$$

where μ_x is the mean of all points:

$$\mu_x = \frac{1}{K}\sum_{i=1}^{K} X_i$$

All K data points in the N-dimensional space are projected onto this line, which contains the maximal possible information in a single dimension along which the data points are to be separated into two groups. This step can be considered as a feature extraction effort following the feature selection in the previous step.

(3) **Partition all data points into two subgroups**

Sort all K data points projected on the direction (from smallest or most negative, to largest), and partition them into two parts. Among all $K-1$ possible ways for the partitioning, we choose the one corresponding to the maximal between-class distance de£ned as

$$p_1(\mu_1 - \mu_0)^2 + p_2(\mu_2 - \mu_0)^2 = p_1 p_2(\mu_1 - \mu_2)^2$$

where μ_i, K_i and $p_i \triangleq K_i/(K_1 + K_2)$ are the mean, number of points, and proportion of each of the two groups ($i = 1, 2$), and $\mu_0 = p_1\mu_1 + p_2\mu_2$ is the mean of all $K = K_1 + K_2$ data points.

A binary tree structure (in either depth or breadth-£rst order) is generated by carrying the above steps for the bisection recursively at each node of the tree until each leaf node contains only one data point.

2.2 Supervised version

In some situations, certain additional information about the genes or the samples may be available which can be utilized to better classify the data. For example, the tissue types (e.g., cancer vs. normal) of the samples in the microarray may be known. In this case, the above steps can be modi£ed to take advantage of the additional information. First, an alternative criterion for feature selection can be used:

$$J = \frac{s_b}{s_w} = \frac{\sum_k p_k(m_k - m_o)^2}{\sum_k p_k \sigma_k}$$

where s_w and s_b are, respectively, the within-class and between-class scatter measures de£ned as

$$s_w = \sum_k p_k \sigma_k, \qquad s_b = \sum_k p_k (m_k - m_o)^2$$

The summation is over all known classes with mean m_k, variance σ_k, and *a priori* probability p_k (e.g., proportion of the samples belonging to the kth class), and m_o is the overall mean of all data points. The feature used for partitioning can be either the original feature with maximum J value, or the top PCA component obtained from the M features corresponding to the M largest J values. The sample points are then partitioned according to their known class identities. One particular criterion Zhang et al. (2001) can be used to optimally partition the data according to a criterion based on an entropy impurity measure:

$$\Delta I = I_p - (K_1 I_1 + K_2 I_2)/K$$

where I_p, I_1 and I_2 are the impurity measures of the whole group, the left subgroup and right subgroup, respectively, which are de£ned as

$$I = -\sum_k p_k log(p_k)$$

For example, if $k = 2$, the impurity (entropy or uncertainty) $I = 0$ when $p_1 = 0$ and $p_2 = 1$, while I reaches a maximum value when $p_1 = p_2 = 1/2$. The partitioning corresponding to the largest ΔI value ensures the smallest impurities I_1 and I_2 of the two resulting subgroups, i.e., they each contain most of the data points belonging to one or few classes.

2.3 Special treatment of points close to boundary

The dataset to be subdivided at each node of the tree is not necessarily linearly separable, i.e., it may be impossible to partition the feature space by a hyper-plane (perpendicular to the direction onto which all data points are projected) without cutting a group of tightly clustered points into two parts, even though the partitioning steps discussed above guarantee certain optimality.

To identify the data points which may be misclassi£ed by the bisection, a buffer zone around the partitioning hyper-plane is de£ned according to the between-class distances used previously to £nd the maximum distance. On either side of the partitioning plane, another plane is found corresponding to a distance slightly smaller than (e.g., 0.9 of) the maximum distance. Each data point in this buffer zone between these two planes is classi£ed into either of the two classes determined pre-

viously, according to its Euclidean distances to its closest neighbors in the two classes. Here all N features are used to £nd the Euclidean distances, as the data points inside the buffer zone cannot be well separated along the direction previously chosen and additional in formation from other dimensions is needed to determine to which of the classes they belong.

As the result of such a special treatment of the data points in the buffer zone, the feature space is no longer partitioned rigidly by a hyper-plane. Instead, the space is separated by a ¤exible surface which allows the data points close to the boundary the freedom to choose their identities according to their nearest neighbors' identities.

2.4 Truncation of the top-down hierarchy

When needed, the bisection clustering described above can be carried out recursively (in either depth-£rst or breadth-£rst order) to build a downward growing binary tree until eventually each terminating or leaf node contains only one data point. In this case, a complete hierarchical structure, a binary dendrogram, is constructed by which the relationship among all the data points (individual genes) in terms of their similarities is represented. This can reveal some underlying structures of the dataset.

Alternatively, it may not be necessary to carry the bisection all the way to the end to get a complete dendrogram, if it is of interest to £nd the possible clusters formed by similar genes. In this case the process of the top-down tree construction can be truncated by terminating the bisection early according to certain criterion. Each of the leaf nodes of such an incomplete tree represents a cluster of similar genes. The relationship among these clusters may still be revealed by the partial dendrogram tree formed before the termination of the splitting process.

The truncation can be determined in various ways. If the total number of clusters is known *a priori*, or it can be estimated by other means, the breadth-£rst bisection recursion can be terminated when the number of nodes in the tree exceeds the known number of clusters. Alternatively, if it is not desirable to have any cluster containing fewer than a certain number of data points, the recursion (in either depth or breadth-£rst order) along a branch can be terminated whenever the end node of a branch contains fewer data points than the smallest size desired. However, for reasons to be discussed below, one may want to carry the bisection a few levels further down before terminating the process. For example, to carry the process two levels lower than that determined by a speci£ed number of clusters K, one could terminate the process after $4K$ nodes are obtained.

The top-down recursive bisection described above can be followed by a £nal clean-up and correction process carried out in a bottom-up fashion to merge the nodes at the lowest levels of the hierarchy. This process may be needed in some unusual situations (depending on the speci£c dataset) where the tree structure generated by the top-down bisection may be unbalanced, so that some nodes reach the minimum cluster size or contain single data point much earlier than others, thereby causing the corresponding tree branches to terminate at levels much higher than others. When such an unbalanced tree is truncated, the sizes of the leaf nodes could vary drastically. To ensure that the £nal clustering results re¤ect objectively the actual dataset, independent of the particular order of the bisection operations (the order of the tree nodes, either depth or breadth £rst), a bottom-up merge back process following the top-down bisection process will allow the opportunity to readjust the clustering according to the similarities between all clusters of the leaf nodes and thereby to correct the potential problem of diverse cluster sizes. Moreover, the merge back process will also correct any possible misclassi£cations made in previous bisection process (although this is unlikely due to the ¤exible partitioning surface).

In the merge back process, the two classes nearest to each other are merged to form a single class and this operation continues until the number of classes is reduced to reach the desired number. While Euclidean distance (squared) between the means of two classes

$$D_E(i,j) = (\mu_i - \mu_j)^T(\mu_i - \mu_j) \tag{1}$$

could be used to determine the two classes closest to each other, other distance metrics may be more suitable to represent the similarity between classes containing multi-points. Typically, Mahalanobis distance

$$D_M(i,j) = (X - \mu_i)^T\Sigma_i^{-1}(X - \mu_i) \tag{2}$$

and Bhattacharyya distance

$$D_B(i,j) = (\mu_i - \mu_j)^T(\Sigma_i + \Sigma_j)^{-1}(\mu_i - \mu_j) \tag{3}$$

can be more properly used to represent, respectively, the similarity between a point and a group of points and the similarity between two groups. Here μ_i and Σ_i are the mean vector and covariance matrix of the ith class. Note that the Euclidean distance D_E is directly affected by a scaling transform ($y = cx$) of the dataset, while both distances D_N and D_B are invariant with respect to scaling, as they are both normalized by the covariance matrix.

However, some degenerate cases need to be considered when some of the classes contain few or even a single point, a likely situation to encounter while merging the leaf nodes of the tree structure. Specifically, when the number of data points in a class is smaller than the dimension of the feature space, the covariance matrix does not have full rank and its inverse does not exist. In particular, if a class contains only one point, its covariance matrix is zero, and the corresponding distances become infinity ($D_M = \infty$ if $\Sigma_i = 0$, $D_B = \infty$ if $\Sigma_i = \Sigma_j = 0$), although it is still meaningful to have some distance defined to measure the similarity between two single points. In order to measure in general the similarity between points and classes mixed together under all circumstances, a new distance is defined as:

$$D_{new} = (\mu_i - \mu_j)^T(\alpha\Sigma_0 + \Sigma_i + \Sigma_j)^{-1}(\mu_i - \mu_j) \tag{4}$$

where Σ_0 is the covariance matrix of the entire dataset involved, and α is a scaling factor. When one or both of Σ_i and Σ_j are zero, the distance becomes the Euclidean distance (squared) between the two mean vectors, scaled by the overall covariance. The distance so defined unifies all three distances D_E, D_M and D_B to cover all cases including degenerate ones. Note that the distance metric so defined is always invariant with respect to scaling, as the Euclidean distance between the means is still normalized by the overall covariance Σ_0 even when both $\Sigma_i = \Sigma_j = 0$. Also note that the new distance has a smooth transition between all three distances.

2.6 Estimate number of clusters

In unsupervised classification such as clustering analysis there is little *a priori* information regarding the dataset, such as roughly how many clusters exist in the dataset and how separable they are. In some cases, the top-down split algorithm discussed above may provide some clue for such information based on the distribution of the distance between the two subgroups at each tree node. In a typical situation where a set of clearly separable clusters exist in the feature space, the number of clusters in the dataset may be estimated from the histogram of all the between-class distances at the tree nodes, such as the one shown in Fig. 1. Usually the larger distances close to the right end of the histogram are from the top level nodes in the tree (close to the root), while the smaller distances to the left of the histogram are from the nodes close to the bottom (the leaves) of the tree. (However, note that there does not exist a strict monotonic relationship between the distance and the depth of the node.) Moreover, once the bisection is carried out enough times to reach the inside of a cluster of points, the distances between the two subgroups in the cluster (intra-cluster) become drastically smaller than those between clusters (inter-cluster). Consequently in the distance histogram there typically exists a dense peak composed of a large number of intra-cluster distances, many more than the number of inter-cluster distances in a much coarse distribution towards the right end of the histogram. The number of clusters can therefore be roughly estimated

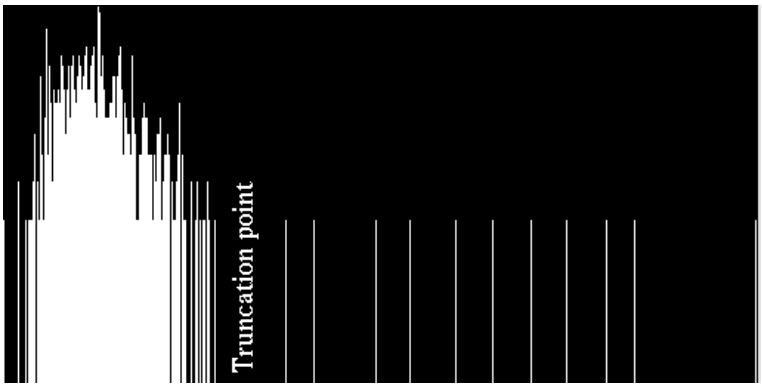

Fig. 1. Split distance histogram with one gap. The transition to widely spaced, singular entries marks the beginning of the cluster count.

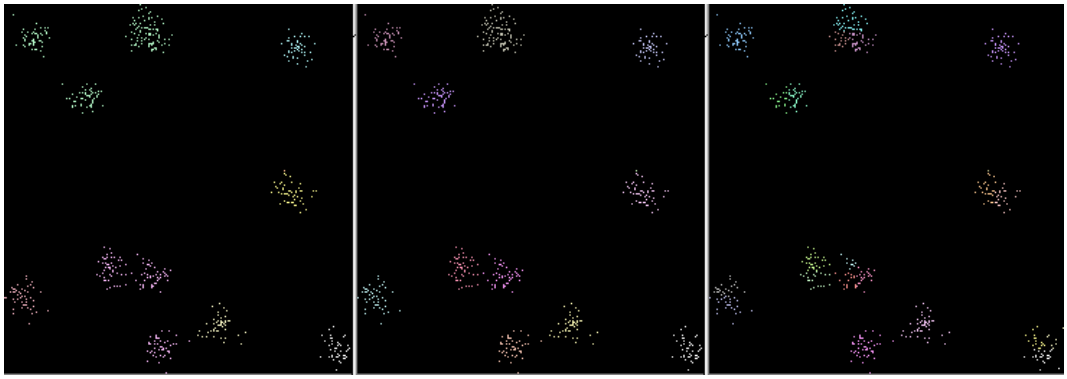

Fig. 2. Under, proper, and over classi£cations of clusters. Underestimating leads to local 'superclusters' while overestimating causes bisection along a cluster's principle axis.

by £nding the point that separates the two different regions in the histogram. The number of splits to the right of this point correspond to the bisections needed to partition the data to reach the individual points inside the clusters (middle panel of Fig.2). The dataset will be either under classi£ed if the top-down split process is terminated too early (left panel of Fig.2), or over classi£ed if it is terminated too late (right panel of Fig.2).

Moreover, the histogram of distances may also re¤ect some more complicated structure of the dataset. For example there may exist multi-level structure in the dataset, such as a cluster of clusters of data points in the space. In this case, there may exist two gaps in the histogram representing the two levels of clusters. If the top-down splitting process is terminated at a distance corresponding to the £rst gap on the right of the histogram (truncation point 1 in Fig.3), the leaf nodes of the resulting tree will represent the top level clusters (left panel of Fig. 4), while if the splitting process is terminated at a distance corresponding to the second gap on the left (truncation point 2 in Fig.3), each leaf node will represent a group of data points forming a lower level cluster (right panel of Fig. 4).

Fig. 3. Split distance histogram with two gaps

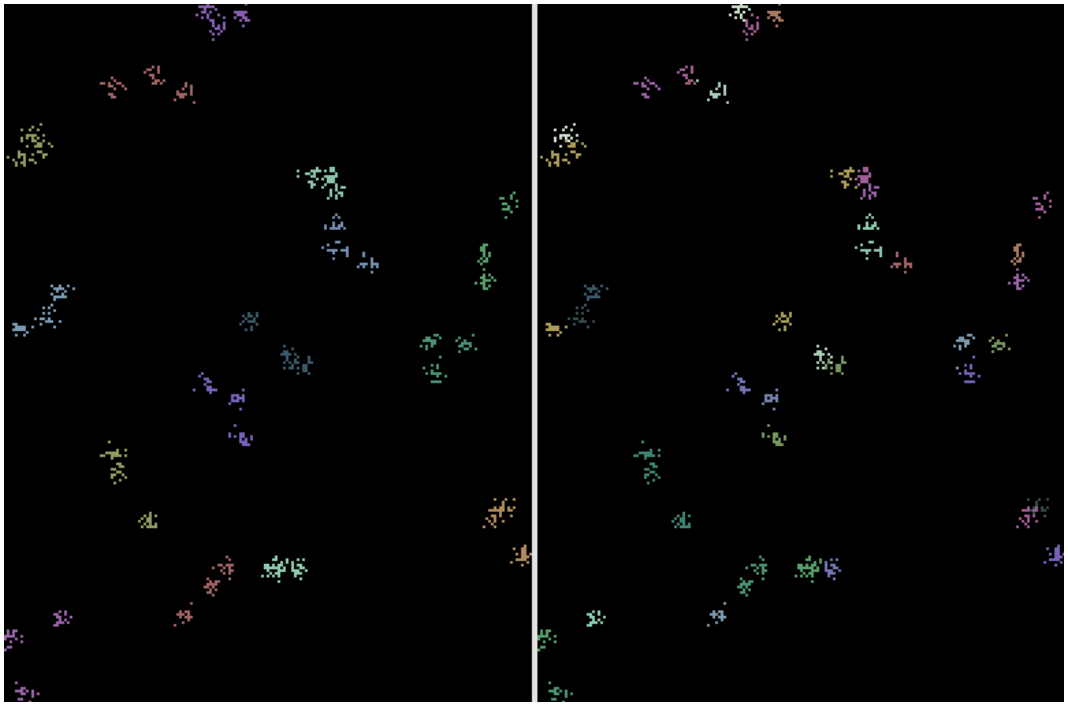

Fig. 4. Shallow (left) and deep (right) clusterings. The shallow clustering successfully £nds 'clusters of clusters'.

2.7 Clustering of the samples in microarray

Sometimes it is of interest to classify the samples (of different tissues obtained under different experimental conditions or times) from a study, while treating the genes as the features. The algorithm discussed above can be applied, in exactly the same fashion, to the same microarray data, with its two dimensions swapped. However, as there are in general many more genes (now treated as features) in the microarray data than the samples ($K \gg N$), the dimensions of the covariance matrix (needed for the PCA feature selection) can be very high (K by K), causing two potential problems: (1) the computation of its eigenvalues/vectors may be very time consuming, and (2) memory may become limiting. In this case, one could always

limit the dimension of the feature space by using only a manageable number of features. Due to the two step feature selection/extraction process discussed above, it is likely that the essential information is preserved in the small number of features actually used.

2.8 Display of the dendrogram

The dendrogram generated by the bisection process can be easily visualized based on the hierarchical structure recursively built during the top-down process of the algorithm. Starting from the root, the binary tree can be trivially drawn, also top-down and recursively, until eventually each leaf node contains a single gene, or a cluster of similar genes. In exactly the same fashion, the dendrogram tree for the experiments can be drawn for the other dimension of the microarray.

3 Advantages of the method

The above method of partitioning the data points has some general advantages.

First, by splitting the dataset from top down, the clusters may be identi£ed much more quickly than if a bottom-up merging algorithm is used as the data points inside the clusters no longer need to be classi£ed. The top-down splitting is especially effective when the dataset contains a large number of genes which form a small number of clusters of similar genes. For example, if the dataset contains many thousands of genes which form only a few tens of clusters, the clusters can be obtained quickly after only a few top-down splitting steps, whereas a bottom-up process will have to work through many unnecessary merging steps to £nally reach the cluster level.

Second, it is not unusual for a microarray to have a large number (thousands or tens of thousands) of genes from a large number (tens or hundreds) of samples from different experimental conditions, tissues, time points, and so on. It is, in general, impossible to £nd a small set of features spanning a subspace in which all data points are well separated. However, with the top-down split algorithm, at each iteration only a subset of the data points are subdivided at a node of the tree (and in general, the lower level a node is at, the smaller the group of points to be subdivided). It is therefore possible to £nd a small subset of features (or their linear combinations) containing suf£cient information to separate the particular subset of points. As this adaptive feature selection can effectively reduce the number of features needed for the bisection at each node, the amount of computation can be drastically reduced. However, this adaptive feature selection cannot be used in a bottom-up merging algorithm because all features are needed for computing the

inter-group distances to determine which two groups of points to merge.

Third, due to the adaptive feature selection aspect of the splitting method, certain biological insights regarding the genes and samples under study may be revealed. For example, if the clustering is done to the genes, one may learn from the features adaptively selected for separating the genes into two groups which experimental conditions or tissue samples are most signi£cant for distinguishing between the two groups. Similarly, if the clustering is done to the samples, one may learn which genes are responsible for causing two different types of tissues. Such biological insights may provide more valuable information than the simple classi£cation of the genes.

Fourth, in general, a group of K points can be partitioned into two subgroups in $2^{K-1}-1$ different ways. Moreover, the computation in the potentially high dimensional feature space can be very time consuming. By projecting the feature space to a single dimension while still keeping a large percentage of the total energy (representing separability information) preserved, the computational complexity is drastically reduced due to the reduction of both dimensionality (from high to one dimensional space) and the number of possible bisections (from exponential $O(2^{K-1})$ to linear $O(K)$). The optimal partitioning can be easily identi£ed.

Finally, it is desirable to put the genes in the microarray in a certain order after clustering analysis so that the most similar genes are arranged more closely to each other than those less similar. In general, there are 2^{K-1} different ways to arrange the K leaf nodes of a binary tree in a linear order. A separate step for £nding the optimal (Eisen et al. (1998), Biedl et al. (2001)) ordering of the genes is usually needed after the clustering analysis. The top-down split algorithm provides an easy way to make such an ordering without any extra step, as the recursive bisection of the gene groups is always carried out according to some prominent feature (or linear combination of some prominent features) based on which all genes are sorted. In other words, the clustering process always systematically rearranges the genes during the top-down recursion. The only effort needed for the optimal ordering of the leaf nodes of the complete dendrogram by the end of the clustering is to decide the order of the two subgroups at each node, depending on which of them is more similar to their parent node's sibling (Alon et al. (1999), Rose et al. (1990)). In fact, the gene array reordered by the clustering algorithm looks very smooth with only a few discontinuities at the boundaries between major subgroups.

4 Testing and comparison with other algorithms

To test the effectiveness of the top-down splitting algorithm (TSplit), it was run over a large number of synthetic datasets and compared to the well-known K-means algorithm, as well as the XCluster algorithm (Eisen et al. (1998), Sherlock (1999)),

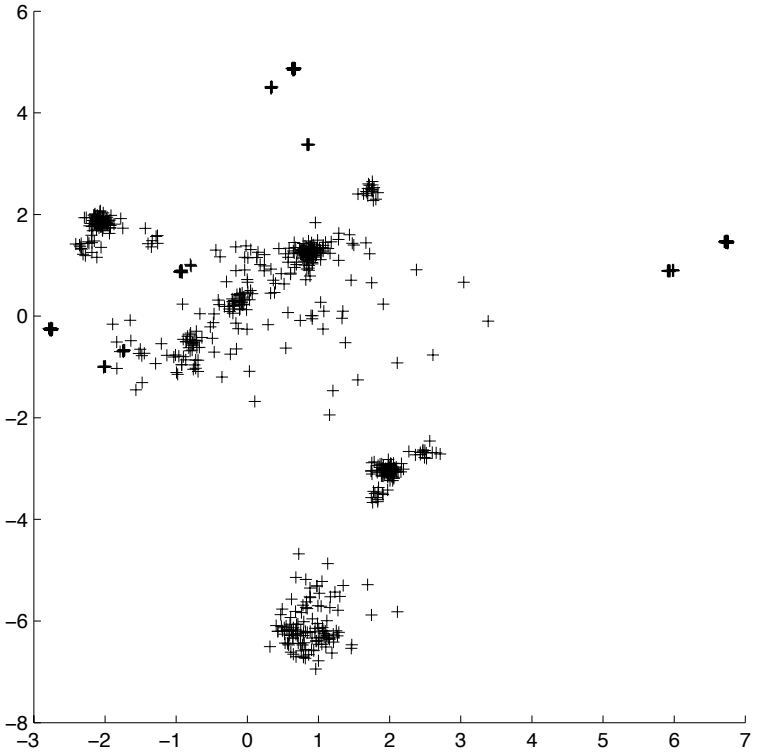

Fig. 5. Example of data in PCA space. There is a mix of tight and diffuse clusters of various sizes

XCluster was chosen because it is a tree-based clustering algorithm which is commonly used in the bioinformatics community and, due to its bottom-up nature, provides good contrast to TSplit's top-down technique.

4.1 Synthetic data

The synthetic data used for the comparison tests is composed of a mixture of spherical Gaussian probability distribution functions. The Gaussians are constructed in a hierarchical manner so as to produce a 7500 point dataset with 15 clusters each composed of three smaller clusters, for a total of 45 clusters in all. A typical example of the synthetic data, projected into 2D PCA space, can be seen in Figure 5.

The datasets are parameterized by number of datapoints, dimensionality and variance ratio. The variance ratio represents the ratio between variances in successive levels of the hierarchy. A ratio less than 1.0 causes the clusters to become tighter in lower levels. In contrast, a ratio above 1.0 forces the clusters to become more diffuse. It has been empirically observed that all the clustering algorithms at our disposal begin to degrade in performance when the ratio exceeds 1.0 and fail completely at a ratio of 2.0 (Hart et al. (2003b)). The datasets which show a range of values means the variance ratio was chosen independently for each branch of the

Linear Assignment Comparisons of Synthetic Data						
		K-means				
Dimensions	Ratio	avg.	std.	Tavazoie	XCluster/Agglom	TSplit
3	0.1	0.6353	0.0497	0.3727	0.9999	0.9521
3	0.3	0.6945	0.0535	0.3324	0.8380	0.8923
3	0.5	0.6806	0.0334	0.4567	0.6445	0.6447
3	1.0	0.1997	0.0048	0.1993	0.1869	0.1695
3	2.0	0.0951	0.0010	0.0963	0.0929	0.0887
10	0.1	0.6923	0.0497	0.4011	1.0000	0.8151
10	0.3	0.6673	0.0441	0.5157	1.0000	0.9775
10	0.5	0.7808	0.0352	0.5585	0.9785	0.9025
10	1.0	0.7721	0.0309	0.6684	0.1513	0.3913
10	2.0	0.1679	0.0020	0.1725	0.0333	0.0617
30	0.1 - 1.0	0.7981	0.0353	0.4127	0.7991	0.9833
30	0.3 - 2.0	0.5967	0.0373	0.2315	0.0951	0.0955

Table 1: Linear Assignment scoring over parameterized datasets. Scores fall in the range of 0.0 (no points matching) to 1.0 (all points matching) when compared against the ground truth of the datasets.

hierarchy from that range inclusive.

The K-means algorithm was run with £ve different random seeds for choosing the initial placement of means and also by the method described in Tavazoie et al. (1999) of placing means maximally distant from one another. The scores in the tables represent the average score as well as the standard deviation over the £ve trials. The XCluster/Agglom and TSplit algorithms are deterministic and their scores are placed in the last two columns, respectively. All the algorithms used the Euclidean distance metric.

To compare cluster outputs two objective metrics are used (Hart et al. (2003a)):

(1) a "Linear Assignment" (LA) scoring function, which optimizes a linear function of a permutation matrix to £nd the best correspondence between two clustering outputs using a matching algorithm Gabow (1973), and (2) the Normalized Mutual Information (NMI) (Forbes (1995)). These comparison experiments were performed using the MLX package of machine learning algorithms and scoring methods (Hart et al. (2003a)).

In addition, the full dendrograms were truncated into non-hierarchical clusterings

NMI Comparisons of Synthetic Data						
		K-means				
Dimensions	Ratio	avg	std	Tavazoie	XCluster/Agglom	TSplit
3	0.1	0.9212	0.0114	0.7937	0.9998	0.9795
3	0.3	0.9223	0.0142	0.7929	0.9260	0.9457
3	0.5	0.8074	0.0047	0.7645	0.7697	0.7675
3	1.0	0.3326	0.0016	0.3320	0.3090	0.2880
3	2.0	0.1532	0.0013	0.1532	0.1423	0.1303
10	0.1	0.9334	0.0109	0.7774	1.0000	0.9216
10	0.3	0.9271	0.0109	0.8845	1.0000	0.9918
10	0.5	0.9517	0.0085	0.8847	0.9903	0.9443
10	1.0	0.7863	0.0089	0.7595	0.1815	0.4913
10	2.0	0.2051	0.0018	0.2111	0.0150	0.0766
30	0.1 - 1.0	0.8705	0.0116	0.7227	0.8064	0.9802
30	0.3 - 2.0	0.8935	0.0119	0.5383	0.0020	0.0099

Table 2: Normalized Mutual Information scoring over parameterized datasets. Scores fall in the range of 0.0 (no points matching) to 1.0 (all points matching) when compared against the ground truth of the datasets.

in order to allow comparison with the ground truth 45 clusters of the synthetic data and also with each other. XCluster was augmented with an agglomeration heuristic (Hart et al. (2003b)) to convert its full dendrogram. This stands in contrast to the stopping criteria for TSplit, described previously, which terminated before constructing the full dendrogram.

A strong Linear Assignment score indicates that two clusterings are nearly identical. In terms of con£dence values, two random clusterings return a score of $\frac{1}{k}$, where k is the number of clusters. For the synthetic data with 45 clusters, a linear assignment score of 0.022 represents a mutually random pair of clusterings. This bound expectation holds in the case of equal number of clusters.

The second method of cluster comparison, Normalized Mutual Information, is computed by the formula

$$nmi(S, R) = 1 - \frac{H(S, R) - H(R)}{H(S)} \tag{5}$$

where $H(S)$ and $H(R)$ represent the information contained in the rows and columns

of the confusion matrix C computed by

$$H(X) = -\sum_i x_i \log(x_i) \tag{6}$$

and $H(S, R)$ is the joint-information computed by

$$H(X, Y) = H(C) = -\sum_j \sum_i c_{ij} \log(c_{ij}) \tag{7}$$

From Tables 1 and 2 two observations may be made. First, both TSplit and XCluster/Agglom consistently outperform K-means over the low-dimension, low-variance datasets with K-means scoring between 10% and 25% below the other methods.

Second, TSplit and XCluster/Agglom produce roughly comparable results through the low-dimensional datasets. However, in the high dimensional case (30 features), TSplit displays a clear improvement over XCluster/Agglom in both Linear Assignment and NMI. It is important to note that all the algorithms completely fail at a variance ratio of 2.0.

4.2 Stability Analysis

It is important to assess the stability of the clustering algorithm, in terms of how sensitive the clustering result is with respect to varying levels of perturbation in the data. If some small perturbation in the data would cause substantial changes in the clustering outcome, the algorithm is not stable and its results not reliable.

To examine the stability of the algorithm, two types of perturbation are considered. First, the dataset is randomly sub-sampled at various rates (>50%). Or, alternatively, different amount of Gaussian noise is added to the dataset. In either case, the clustering results between many trials are compared by Linear Assignment score based on the confusion matrices and a composite measure of stability is generated. This process is depicted in Figure 6.

4.2.1 Self-Consistency Tests

The dataset is first randomly sub-sampled at various rates $p \in P = \{0.6, 0.7, 0.8, 0.9\}$ ($|P| = 4$), and for each rate p, N=10 clustering trials are conducted. For each pair of two sub-sampling rates p and q (e.g., $p = 0.8,\ q = 0.7$ in Figure 6), a confusion matrix is obtained based on the common data points (the intersect $C = S_p \cap S_q$ of the two sub-samplings S_p and S_q). Then the Linear Assignment scores averaged over all combinations of the trials of the two sub-sampling rates is obtained as the corresponding element of a 4x4 stability matrix representing the consistency of the

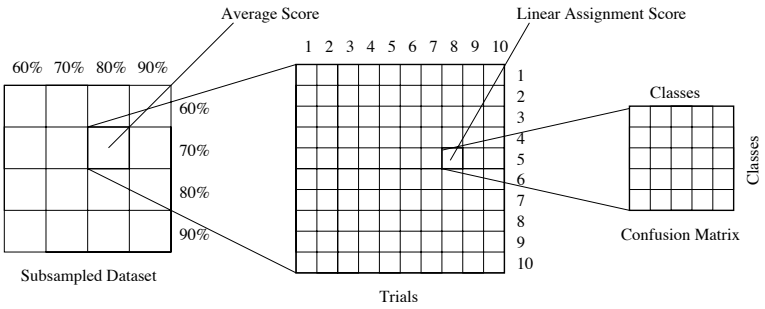

Fig. 6. Structure of the noise analysis. From left to right: (1) Each cell of the subsampling matrix is the average of (2) the trials matrix which is £lled with the Linear Assignment score of (3) the confusion matrix built from the intersection of any two runs.

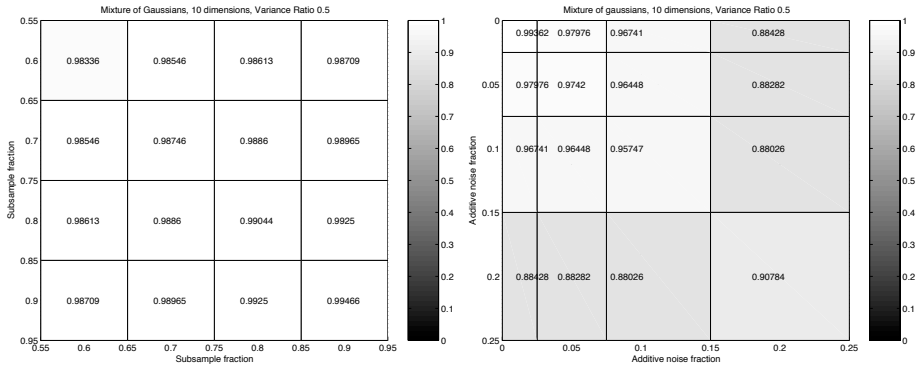

Fig. 7. Self-consistency at a variance ratio of 0.5 with subsampling (left) and added noise (right). Both tables exhibit high agreement (>96%) except for the greatest additive noise, which achieved >88% agreement.

clustering results across all sub-sampling rates. Such a stability matrix obtained from a 10-dimensional dataset is shown on the left of Figure 7.

In a similar fashion, the stability of the algorithm is also tested with different amounts of additive noise. The level of noise is measured in terms of the fraction of the standard deviation of the whole dataset. Four levels 1%, 5%, 10% and 20% of noise are added to the dataset. For comparison, adding 20% noise to the dataset is roughly equal in magnitude to the standard deviation of the clusters within the dataset. Same as the sub-sampling test, for each noise level N=10 trials are carried out. The resulting 4 by 4 stability matrix is shown on the right of Figure 7, with each element representing the linear assignment score computed based on a pair of two levels of noise added to the data, averaged over all trials for the two noise levels.

It is seen that most scores in the two matrices are greater than 90%, indicating that the clustering results across different sub-samplings or different levels of noise are highly consistent, i.e., the clustering algorithm is stable.

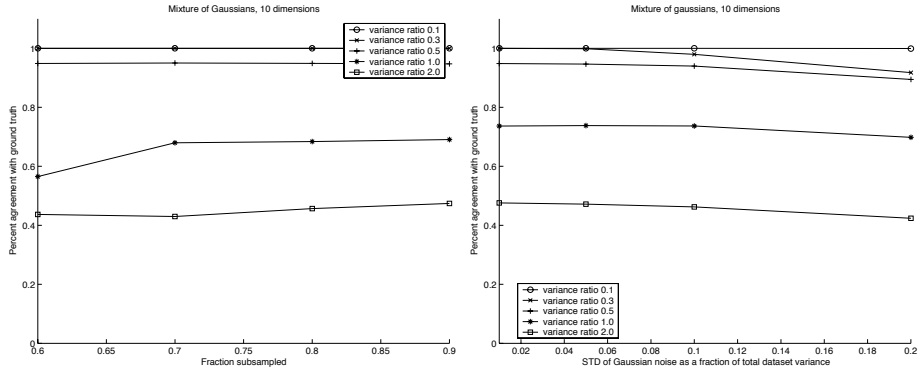

Fig. 8. Stability measured against ground truth with subsampling (left) and added noise (right). Both plots exhibit good consistency across perturbations, with the absolute scores increasing slightly as the subsampling fraction is increased (left) and decreasing slightly as more noise is added (right).

4.2.2 Comparison with Ground Truth

While it is desirable for a clustering algorithm to be stable and generate consistent results, it is even more important for the algorithm to generate correct (as well as consistent) results. To test our algorithm in this regard, we further compared its clustering results against the ground truth of a synthetic dataset composed of £ve overlapping spherical Gaussians.

The clustering results for the datasets of different variance ratios (0.1, 0.3, 0.5,1.0 and 2.0) are shown in Figure 8. The dataset with variance ratio 2.0 is the most dif£cult one to classify for which our algorithm and all other clustering methods compared (K-means, XCluster/Agglom, etc.) performed poorly at this variance ratio, generating clustering results roughly equivalent to random assignment of classes (corresponding to a linear assignment score of 0.2). This performance can be treated as the baseline for measuring other clustering performances. Our clustering algorithm is applied to the dataset sub-sampled by different rates and with different amounts of noise added, and the results are compared with the ground truth in terms of the Linear Assignment as shown in Figure 8. It is seen, as expected, that the performance improves as sub-sampling rate increases (left graph), and worsens as the level of noise increases (right graph). However, the most important observation is that regardless of the absolute value of the scores, they are remarkably consistent across all different sub-sampling rates or different noise levels, and none of the curves cross others at any point, indicating that the algorithm is highly stable.

4.3 Visualization of the splitting process

Figure 9 shows the £tness function for the £rst four layers of the tree. The vertical bar indicates where the node was split. The dataset in this £gure is the 30000 point, 30 dimensional dataset with a variable variance ratio between 0.1 and 1.0.

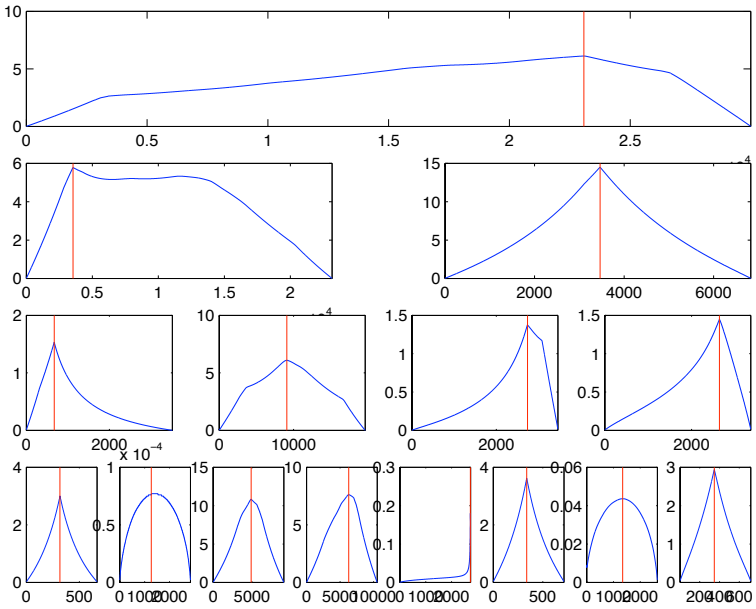

Fig. 9. The £tness function used to identify splitting points exhibits a discontinuity at the maximum when a clear point exists.

A soft, rounded peak in the £tness plot represents a splitting point where the two subsets are not well separated. A plot which has its maximum at a sharp peak means that the two clusters are well-separated.

4.4 Performance relative to XCluster/Agglom

Since it was shown in the previous section that TSplit produces clusterings of the same approximate quality of XCluster/Agglom, it is instructive to compare the execution time of the two algorithms.

A serial scalability test was performed by taking a random subset of N points from the 30,000 point dataset shown in Figure 5 and timing the execution for each algorithm to £nd 45 clusters. The summary of timings are shown in Figure 10. TSplit has a much smaller absolute execution time and also exhibits superior scaling relative to XCluster/Agglom.

4.5 Full Tree Decomposition

It is sometimes useful to view the full tree structure of a clustering since a human may be able to pick up patterns which cannot be extracted by the particular analysis method employed. TSplit can easily be run down to nodes of size one which produces a full tree decomposition of the dataset. Since XCluster also produces such a decomposition, the two algorithms may be visually compared.

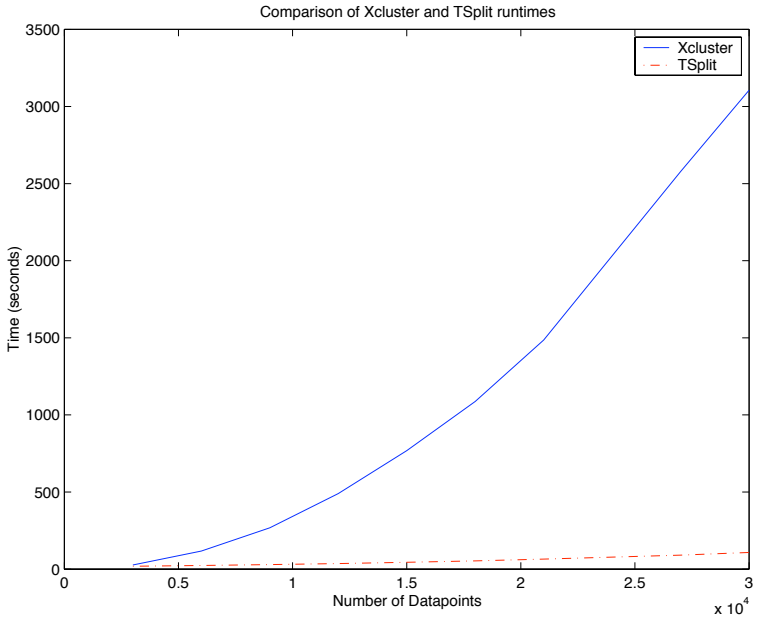

Fig. 10. Scaling of TSplit and XCluster/Agglom versus data size. TSplit grows at a much slower rate than XCluster/Agglom.

The following £gures show the decompositions of a small (99 point) example dataset from Eisen et al. (1998). An observation consistent over several different datasets is that TSplit appears to produce a more balanced splitting. The trees for TSplit and XCluster shown in Figure 11, on the left and right respectively, have depths of 7 and 10.

5 Parallelization

The top-down approach of the algorithm leads to a natural parallel implementation. In the section we detail our implementation strategy and address the major bottleneck in the serial algorithm.

5.1 Coarse partitioning

Since no information is shared between subtrees during construction, this method is trivially parallelizable. Our strategy is simple: begin with all processes attached to the root node and divide them proportionally between the children. This has the bene£t of providing more computation resources near the top of the tree where the nodes are most likely to contain a large number of data points. As the tree is split, the processes are diffused until each node has a single process attached to it. At this point, the splitting process continues in a serial manner.

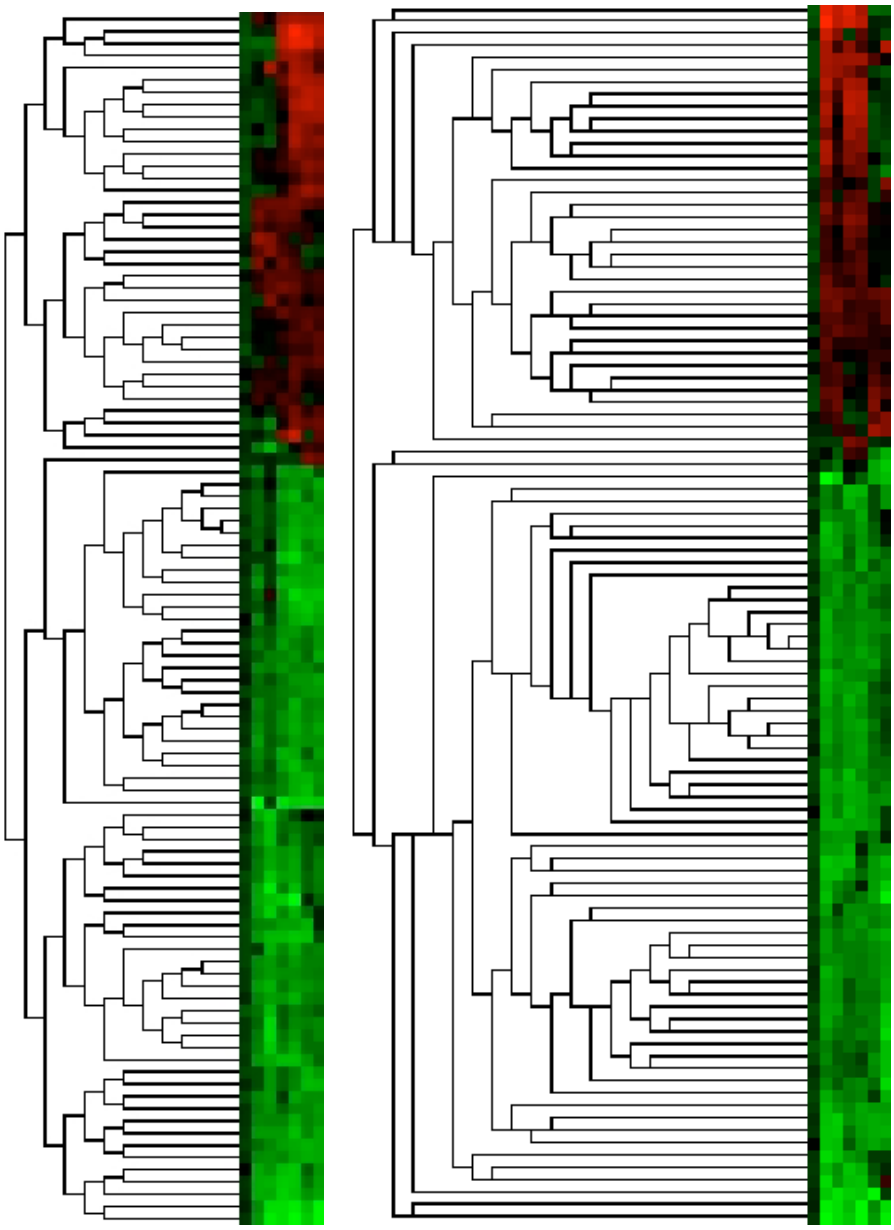

Fig. 11. TSplit (left) vs. XCluster (right) full dendrogram. TSplit's dendrogram is more balanced and thus more compact than XCluster.

The ease of parallelization of such a top-down technique stands in sharp contrast to a bottom-up strategy, which entails a non-trivial amount of communication between processes. On machines with slow or high-latency communication, the overhead may negate the bene£t of parallelizing. Our algorithm does not suffer a high communication overhead.

5.2 Fine partitioning

Each node may have several processes at its disposal and they should be used as ef£ciently as possible. Fortunately, the computations for determining the optimal splitting point of a node are amenable to parallelization as well.

The optimization process reduces to £nding the maximum of an array and returning its position. This is ef£ciently handled by scattering the data across the available processes and reducing the local maximum to the root node of the communication subgroup.

The bottleneck of the splitting process is the special treatment of points near the boundary. If this buffer zone is large (several thousand points), its evaluation can take over 99% of the runtime for the splitting process.

5.3 Merging subtrees

After the full tree has been built by each process, one is left with N subtrees which share a common root. For the algorithm to continue, these subtrees must be merged at the master process into a single tree.

Our solution to this problem is to serialize each subtree via a preorder traversal and send the full subtree to the master process in a single transfer. The subtree is merged into the full tree node-wise by £rst searching for the node's parent and linking it to the appropriate branch. The preorder traversal guarantees that a parent node will be merged in before any of its children.

A summary of the parallel splitting algorithm is contained in Table 5.3.

6 Experimental results

6.1 Scalability test

The parallel algorithm was run multiple times using different number of processors (1, 2, 4 and 8) and different data sizes (25000, 50000 and 100000 data points) to £nd its scalability. The results are summarized in Table 4.

Here the £rst column on the left shows the number of data points, the £rst row on top shows the number of processors used, and the values in the table represent the total execution time in second. The £rst three plots in Fig. 12 show how the

```
root ← all processes
Q ← root
while Q not empty
        node ← pop(Q)
        if ¬ valid node
                continue
        end if
        N_p ← number of processes in node
        split(node)
        if N_p = 1
                left child ← process
                right child ← process
        else
                N_l ← number of points in left child
                N_r ← number of points in right child
                p ← ⌈N_l / N_r⌉
                left child ← processes[1 ... p]
                right child ← processes[p+1 ... N_p]
        end if
        Q ← left
        Q ← right
end while
merge_trees()
```

Table 3: Pseudo-code for the parallel implementation of TSplit

	Number of Processors			
Datapoints	1	2	4	8
25000	126.3	66.7	52.2	29.0
50000	399.0	199.4	149.9	76.6
100000	1098.5	574.5	298.6	169.0

Table 4: Runtime (in seconds) of TSplit on dataset size versus number of processors. The speedup is nearly linear as the number of processors increases.

execution time reduces as the number of processor increases for the three datasets, and the last plot shows the speedup $S[n] = T[n]/T[1]$ as a function of the number of processors. As can be see in the £gure, the $S[n]$ scales almost linearly with the number of processors, and the ef£ciency $E = S[n]/n$ (the slope) increases toward 1 when the number of data points increases. All these results show that the parallel algorithm exhibits good scalability.

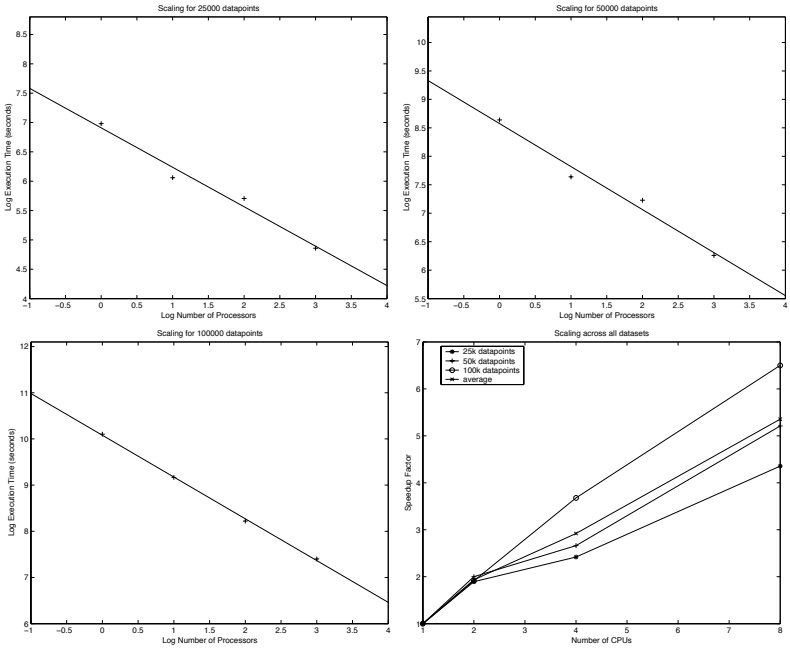

Fig. 12. Log-log plots of execution time vs. processors for 25,000 (top-left), 50,000 (top-right), and 100,000 (bottom-left) data points. The fourth plot shows the relative speedup for each plot plus the average over all three datasets.

6.2 Clustering of a yeast dataset

The splitting clustering algorithm was applied to the yeast *S. cerevisiae* dataset used in Eisen et al. (1998) as the second example. The upper and lower portions of the complete dendrogram generated by clustering analysis of both the 2,467 genes and the 79 samples is shown in Figures 13 through 14. To maximally display the expression levels, pseudo colors are used to show the high expression levels in red, low levels in blue, and middle levels in yellow.

6.3 A supervised example

The clustering process can be considered as the training phase and the tree structure obtained can be used as a trained classi£er for other sample data with unknown identities. Moreover, biological/medical insights may be obtained from the

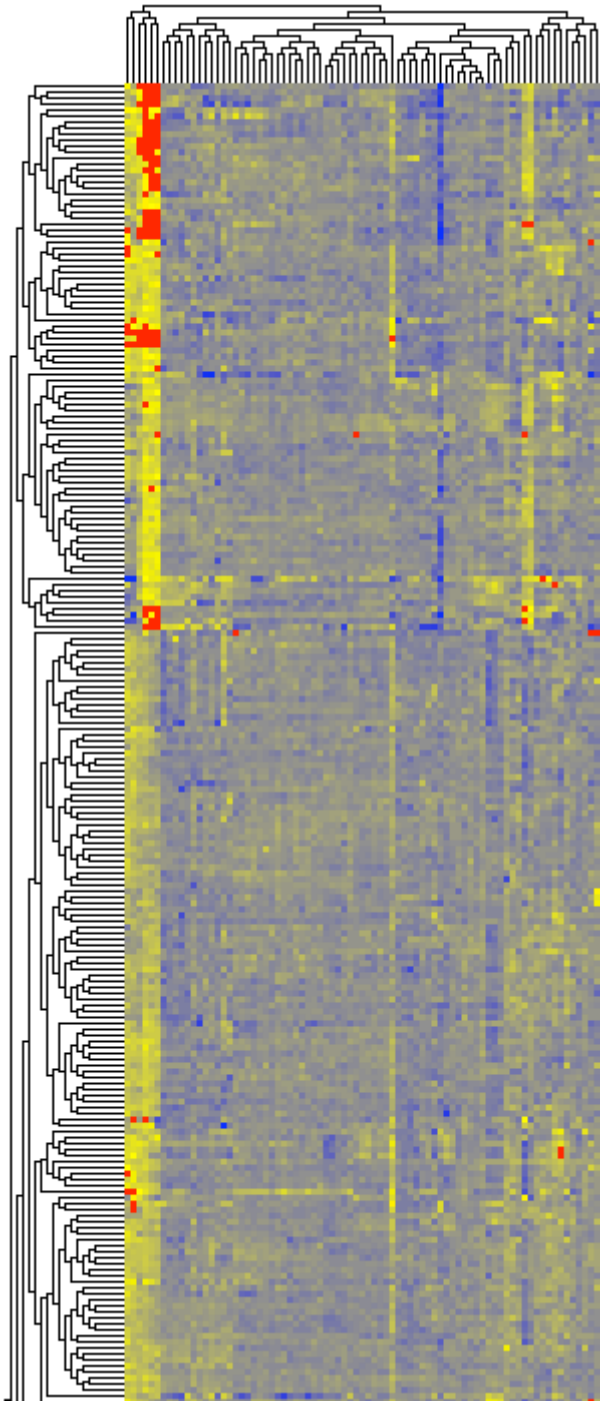

Fig. 13. Clustering result of yeast dataset (top segment)

tree such as which genes are associated with certain diseases.

The top-down split algorithm was applied to a set of microarray data containing 12,558 genes from 54 cancer tissue samples (Scho£eld et al. (2001)). Without any prior knowledge of the genes, the unsupervised version of the clustering analysis was used to obtain a complete dendrogram of the genes. However, as a clinical

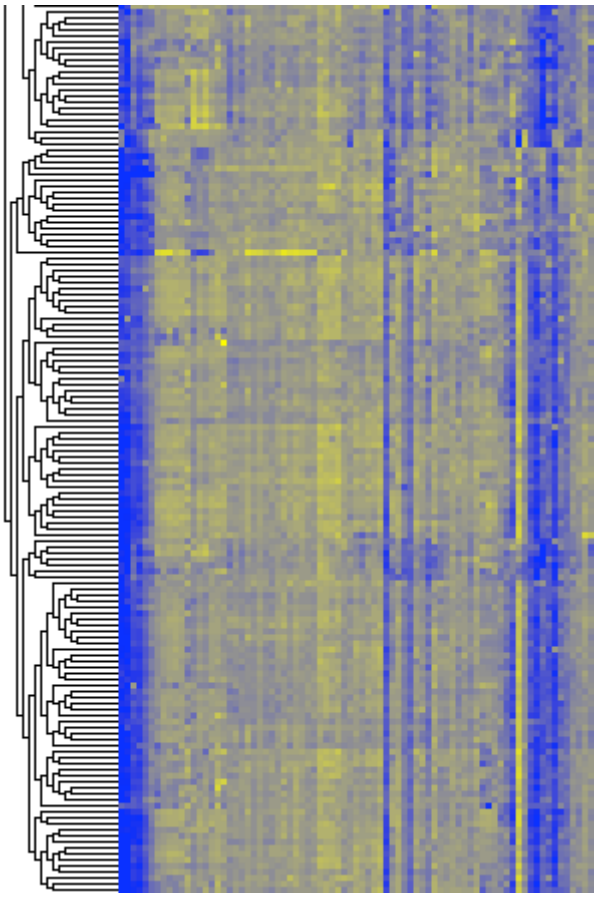

Fig. 14. Clustering result of yeast dataset (bottom segment)

classi£cation of each tissue (class 1 or class 2) was known, the supervised version of the algorithm could be used for the tissue clustering. At each tree node one of the 12,558 genes was selected as the best feature to partition the tissue samples according to their known classes.

The resulting dendrogram was able to fully separate the two cancer tissue classes using only three genes. At the very top level, most of the class 2 samples are separated from the rest (all class 1 plus two class 2 samples). In the next two levels, the remaining two class 2 samples are separated from the class 1 samples by two more genes.

7 Conclusion

We have presented an ef£cient, top-down clustering algorithm which may be applied as either an unsupervised or a supervised learning algorithm. Our testing shows it to be more ef£cient than comparable algorithms while still producing equivalent quality clusterings.

Future work will be focused toward automatically extracting information from the inter-cluster distance histograms and further investigation into the adaptive nature of the recursive partitioning.

References

Alon, U., Barkai, N., Notterman, D. A., Gish, K., Ybarra, S., Mack, D., Levine, A. J., Jun 1999. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. Proc Natl Acad Sci U S A 96 (12), 6745–50, (eng).

Alter, O., Brown, P. O., Botstein, D., Aug 2000. Singular value decomposition for genome-wide expression data processing and modeling. Proc Natl Acad Sci U S A 97 (18), 10101–6, (eng).

Ben-Hur, A., Guyon, I., 2003. Detecting stable clusters using principal componenet analysis. In: Brownstein, M., Kohodursky, A. (Eds.), Methods in Molecular Biology. Humana Press, pp. 159–182.

Biedl, T., Brejova, B., Demaine, E. D., Hamel, A. M., Vinar, T., April 2001. Optimal Arrangement of Leaves in the Tree Representing Hierarchical Clustering of Gene Expression Data. Tech. Rep. CS-2001-14, University of Waterloo.

Brown et al., Jan 2000. Knowledge-based analysis of microarray gene expression data by using support vector machines. Vol. 97. National Acadamy of Science USA.

Eisen, M. B., Spellman, P. T., Brown, P. O., Botstein, D., Dec 1998. Cluster analysis and display of genome-wide expression patterns. Proc Natl Acad Sci U S A 95 (25), 14863–8.

Forbes, A., 1995. Classi£cation-algorithm evaluation - 5 performance-measures based on confusion matrices. Journal of Clinical Monitoring 11 (3), 189–206.

Gabow, H., 1973. Implementation of algorithms for maximum matching on nonbipartite graphs. Ph.D. thesis, Stanford University.

Hart, C., Damle, S., Roden, J., Bornstein, B., Mjolsness, E., Wold, B., 2003a. A framework for leveraging cluster comparison to gain insights in gene expression patterns, in progress.

Hart, C., Mjolsness, E., Wold, B., 2003b. Private communication.

Rose, K., Gurewitz, E., Fox, G., 1990. Statistical-mechanics and phase-transitions in clustering. Physical Review Letters 65 (8), 945–948.

S. Raychaudhuri, J.M. Stuart, R. A., 2000. Principal components analysis to aummarize microarray experiments: Application to sporulationi time series. Paci£c Symposium of Biocomputing .

Scho£eld, D., Mjolsness, E., Buckley, J., Wold, B., Triche, T., 2001. Gene expression patterns in pediatric sarcomas as a predictor of clinical outcome. Modern Pathology 14 (1), 18A–18A.

Sherlock, G., 1999. Xcluster, http://genome-www.stanford.edu/~ sherlock/cluster.html.

Tavazoie, S., Hughes, J. D., Campbell, M. J., Cho, R. J., Church, G. M., 1999. Determination of genetic network architecture. (using k-means). Nature Genetics 22, 281–285.

Xiong, M., Jin, L., Li, W., Boerwinkle, E., 2000. Computational methods for gene expression-based tumor classi£cation. Biotechniques 29 (6), 1264–1268.

Zhang et al., 2001. Recursive Partitioning for Tumor Classi£cation with Gene Expression Microarray Data. Vol. 98. National Acadamy of Science USA.