A new algorithm for hybrid hierarchical clustering with visualization and the bootstrap

Mark J. van der Laan and Katherine S. Pollard

Abstract

We propose a hybrid clustering method, Hierarchical Ordered Partitioning And Collapsing Hybrid (HOPACH), which is a hierarchical tree of clusters. The methodology combines the strengths of both partitioning and agglomerative clustering methods. At each node, a cluster is partitioned into two or more smaller clusters with an enforced ordering of the clusters. Collapsing steps uniting the two closest clusters into one cluster can be used to correct for errors made in the partitioning steps. We implement a version of HOPACH which optimizes a measure of clustering strength, such as average silhouette, at each partitioning and collapsing step. An important benefit of a hierarchical tree is that one can look at clusters at increasing levels of detail. We propose to visualize the clusters at any level of the tree by plotting the distance matrix corresponding with an ordering of the clusters and an ordering of elements within the clusters. A final ordered list of elements is obtained by running down the tree completely. The bootstrap can be used to establish the reproducibility of the clusters and the overall variability of the followed procedure. The power of the methodology compared to current algorithms is illustrated with simulated and publicly available cancer gene expression data sets.

Key words: Cluster analysis, hierarchical tree, bootstrap, gene expression.
Abbreviated title: HOPACH clustering
AMS classification: primary 62H30, secondary 62H10

1 Introduction

Exploratory data methods, such as cluster analysis, are being widely applied to the high dimensional data structures produced by today's researchers. Advances in technology have vastly altered the type and amount of data collected in fields such as molecular

[0]Mark J. van der Laan is Professor in Biostatistics and Statistics, University of California, Berkeley. Katherine S. Pollard is a doctoral candidate in Biostatistics in the School of Public Health, Division of Biostatistics, University of California, Berkeley. This research has been supported by a grant from the Life Sciences Informatics Program with industrial partner biotech company Chiron Corporation. Author for correspondence: Mark J. van der Laan, Div. of Biostatistics, Univ. of California, School of Public Health, Earl Warren Hall #7360, Berkeley, CA 94720-7360

biology, meteorology, medicine, astronomy, and marketing. As the means for collecting and storing ever larger amounts of data develop, it is essential to have good methods for identifying patterns, selecting variables of interest and building models in high-dimensional settings. For example, our methodological research has focused on gene expression data analysis where the number of genes far exceeds the number of samples. In this paper, we present a new hierarchical clustering algorithm called HOPACH that has been developed to address some of the short comings of currently available methods for clustering gene expression data. Since similar issues are likely to arise in other high dimensional data settings, the HOPACH algorithm will be applicable for analysis of data bases in many fields.

1.1 Clustering methods

Clustering is a type of unsupervised learning in which a set of elements is separated into homogeneous groups. Suppose we are interested in clustering p elements $\mathbf{x}_j$, $j \in \{1, \ldots, p\}$ and that each element $\mathbf{x}_j$ is an n dimensional vector $(x_{1j}, \ldots, x_{nj})^T$. Let $d(\mathbf{x}_j, \mathbf{x}_{j'})$ denote the dissimilarity between elements j and j' and let $\mathbf{D}$ be the $p \times p$ symmetric matrix of dissimilarities. Typical choices of dissimilarity include Euclidean distance, 1 minus correlation, 1 minus absolute correlation and 1 minus cosine-angle. Clustering routines (either implicitly or explicitly) map a distance matrix $\mathbf{D}$ into p cluster labels. Graph theoretic, model-based and non-parametric approaches have been implemented.

The non-parametric clustering methods can be separated into partitioning and hierarchical algorithms. Partitioning methods, such as Self-Organizing Maps (Törönen *et al.* (1999)), Partitioning Around Medoids (Kaufman & Rousseeuw (1990)), and K-Means, identify a user specified number of clusters and assign each element to a cluster. Hierarchical methods involve constructing a tree of clusters in which the root is a single cluster containing all the elements and the leaves each contain only one element. Hierarchical methods are used when it is of interest to look at clusters at a range of levels of detail, including the final level of the tree, which is an ordered list of the elements. Such a list, in which nearby elements are similar, is often more helpful to the subject-matter scientist than a collection of large, unordered groups. A hierarchical tree can be divisive (*i.e.*: built from the top down by recursively partitioning the elements) or agglomerative (*i.e.*: built from the bottom up by recursively combining the elements). DIANA (Kaufman & Rousseeuw (1990)) is an example of a divisive hierarchical algorithm, while AGNES (Kaufman & Rousseeuw (1990)) and **Cluster** (Eisen *et al.* (1998)) are examples of agglomerative hierarchical algorithms. Agglomerative methods can be employed with different types of linkage. In average linkage methods, the distance between two clusters is the average of the dissimilarities between the points in one cluster and the points in the other cluster. In single linkage methods (nearest neighbor methods), the dissimilarity between two clusters is the smallest dissimilarity between a point in the first cluster and a point in the second cluster.

1.2 Bootstrap

Most clustering routines, including the methods proposed in this paper, fit in the general statistical framework for gene clustering presented in van der Laan & Bryan (2001). In this formal framework, the underlying target subset (with cluster labels) is defined as a deterministic function $S(\mu, \Sigma)$ of the population mean and covariance matrix (or any other smooth function of the true data generating distribution). The bootstrap is used to estimate the distribution of the estimated target subset $S(\mu_n, \Sigma_n)$, obtained by substituting the empirical mean and covariance. In particular, the authors propose to run the bootstrap with fixed cluster centers and to report for each element (e.g.: each gene) the cluster-specific proportion of times it falls in that cluster. A corresponding cluster probability plot allows one to inspect the cluster reproducibility visually and numerically. Cluster probabilities can also be used to obtain a sensible ordering of elements within clusters so that weakly clustered elements can be filtered out. The authors also prove consistency of the subset estimates and asymptotic validity of the bootstrap under the assumption that the sample size converges faster to infinity than the logarithm of the number of elements. In this paper, we will illustrate how the bootstrap methodology of van der Laan & Bryan can applied with HOPACH.

1.3 Gene expression analysis

New technologies are allowing researchers to monitor the expression of thousands of genes simultaneously. By comparing gene expression profiles across cells that are at different stages in some process, in distinct pathological states, or under different experimental conditions, we gain insight into the roles and interactions of various genes. For example, one can compare healthy cells to cancerous cells within subjects in order to learn which genes tend to be over (or under) expressed in the diseased cells; regulation of such differentially expressed genes could produce effective cancer treatment or prophylaxis (DeRisi *et al.* (1996)). Groups of differentially expressed genes which are *significantly correlated with each other* are particularly interesting, since such genes might be part of the same causal mechanism. In addition to identifying interesting clusters of genes, researchers often want to find subgroups of samples which share a common gene expression profile. Ross *et al.* (2000), for example, use microarray data to classify sixty human cancer cell lines.

A typical gene expression experiment results in an observed data matrix X whose columns are n copies of a p-dimensional vector of gene expression measurements. Consider, for example, a population of cancer patients from which we take a random sample of n patients, each of whom contributes p gene expression measurements. For microarrays, each measurement is a ratio, calculated from the intensities of two floures-cently labeled mRNA samples (*e.g.*: tumor and healthy tissues) cohybridized to arrays spotted with known cDNA sequences. Gene chips produce similar data, except each element is a quantitative expression level rather than a ratio. The methods presented in this paper can be applied to both types of data, but we will focus on microarrays. Data preprossessing may include background subtraction, combining data from replicated spots representing the same cDNA sequence, normalization, log transformation,

and truncation.A typical next step is to select an interesting subset of the genes (*e.g.*: based on a cut-off value for mean expression). Throughout this paper, we assume that preprossessing and subsetting have already been completed.

Approaches to gene expression data analysis rely heavily on results from cluster analysis. Hierarchical clustering of genes has been popularized by Eisen *et al.*, who apply an agglomerative hierarchical clustering algorithm (**Cluster**) to the empirical correlation matrix. Kaufman & Rousseeuw also describe and implement hierarchical clustering algorithms (*e.g.*: DIANA, AGNES) in FORTRAN and Splus which can be used to cluster gene expression data. Two serious drawbacks of these methods are that the ordering of the genes has a random component (and is therefore non-unique) and that the splits at each node are forced to be binary, while there is no biological reason for this choice. These two factors often result in clustering results in which clearly similar elements are not ordered near each other. Furthermore, once an error is made in one of these algorithms, it is impossible to correct it further down (or up) the tree. These issues motivated the development of HOPACH.

Researchers are not only interested in clustering genes, but also in clustering or classifying samples based on similarity of gene expression patterns. Thus, while gene clustering results are of biological interest themselves, they serve an additional purpose if we can use them to aid in the task of clustering samples. The simulation and data analysis illustrate, in particular, that a sensible strategy for clustering the samples is to first cluster genes and then cluster samples for each cluster of genes separately (possibly iterating this process). These results can again be visualized in a reordered data matrix, where the ordering of samples is produced by applying HOPACH to the transposed data matrix. This approach can reveal underlying structure in the data which may not be apparent when the samples are clustered using all genes. For a more formal treatment of this subject we refer the reader to Pollard & van der Laan (2002a), where we extend the statistical framework of van der Laan & Bryan (2001) to include clustering of both genes and samples by defining a simultaneous clustering parameter which is a composition of a mapping for clustering genes and a mapping for clustering samples.

1.4 Visualization

In addition to illustrating how hierarchical clustering can be applied to gene expression data, Eisen *et al.* provide a visual plot of the ordered p by n data matrix X. They apply their **Cluster** algorithm separately to both genes and samples to reorder the rows and columns of the data matrix for the plot. Each log ratio is represented by a spot on the red/green color scale. This plot allows one, to a certain extent, to visually inspect the strength of clusters. We feel that the visualization of clusters is an important contribution, but that plotting the data matrix might not show clustering patterns with respect to certain distances. For example, two vectors can be highly correlated, but have very different mean expression levels. Such vectors will be close using 1 minus correlation as the distance metric (details below), but their similarity will be difficult to "eyeball" in the data matrix. We propose that one should, in particular, visualize the corresponding distance matrix. Visualizing ordered distance matrices was proposed

by Sokal and has been used by others since (e.g.: Chen (2002)). We emphasize the usefulness of this method in high dimensional settings, such as gene expression data. Ordered distance matrix plots (and to a certain degree ordered data matrix plots) corresponding to different levels of a hierarchical tree help to determine the main clustering structures in the data set and provide information about the clusters, such as their strength and their similarity to each other. In this paper, we use a red-white color scale (rather than red-green) for improved visibility.

1.5 Overview

In this paper, we aim to build on the strengths of currently employed methods and to address some of their drawbacks. In Section 2, we describe our clustering method, Hierarchical Ordered Partitioning and Collapsing Hybrid (HOPACH), which creates a hierarchical tree of clusters with an enforced ordering of the clusters. Three important components of this method which are improvements over other hierarchical methods are 1) not restricting splits to be binary (which we see degrades performance in the simulation and data analysis), 2) ordering the clusters and elements within clusters deterministically, and 3) combining recursive partitioning with a collapsing step that allows erroneously separated groups of elements to be reunited. In addition, the methodology includes

- running down the tree to obtain an ordered list of genes,
- plotting the ordered distance matrix at any level of the tree,
- identifying the main clusters, and
- running the bootstrap to establish the reproducibility of the clusters.

Section 3 discusses methods for statistical inference based on applying the bootstrap to HOPACH clustering results. In Section 4, we apply HOPACH to simulated data in order to illustrate its performance relative to other clustering methods. By defining true clustering parameters, we are able to compare the parameters defined by different algorithms. We show that HOPACH is a consistent estimator of cluster labels and is more efficient than existing algorithms for realistically small sample sizes. In Section 5, we analyze a publicly available cancer data set consisting of gene expression from a variety of cell lines derived from different types of tumors. The simulations and data analysis demonstrate, in particular, that HOPACH is better able to identify clusters and to produce a sensible ordering of the elements being clustered than a number of currently employed methods.

2 Hierarchical Ordered Partitioning and Collapsing Hybrid

We present a new clustering method, Hierarchical Ordered Partitioning and Collapsing Hybrid (HOPACH), which applies partitioning and collapsing steps iteratively to create

a hierarchical tree whose final level is an ordered list of the elements. By combining the strengths of two celebrated approaches to clustering, we create a more flexible algorithm for finding patterns in data.

It is important to note that our methodology is a general approach which could be applied with any choice of partitioning algorithm. Commonly employed partitioning algorithms include K-Means and Self-Organizing Maps (SOM). To be concrete, however, we will henceforth assume that Partitioning Around Medoids (PAM) is being used. We refer to our particular implementation with PAM as the partitioning algorithm as HOPACH-PAM. Other possible algorithms would be HOPACH-KMEANS and HOPACH-SOM.

2.1 Partitioning Around Medoids (PAM)

Our HOPACH-PAM algorithm is based on the output of the clustering procedure PAM (Kaufman & Rousseeuw, 1990, chap. 2), which takes as input a dissimilarity matrix $\mathbf{D}$ based on any distance metric. Suppose we are clustering p elements $\mathbf{x}_j$, $j \in \{1, \ldots, p\}$, each an n dimensional vector $(x_{j1}, \ldots, x_{jn})^T$. Let $d(\mathbf{x}_j, \mathbf{x}_{j'})$ denote the dissimilarity between elements j and j'. Possible dissimilarities are:

$$\begin{aligned}
\text{Euclidean: } d(\mathbf{x}_j, \mathbf{x}_{j'}) &= \sum_{l=1}^{n} (x_{jl} - x_{j'l})^2 \\
\text{correlation: } d(\mathbf{x}_j, \mathbf{x}_{j'}) &= 1 - \rho_{jj'} \\
\text{absolute correlation: } d(\mathbf{x}_j, \mathbf{x}_{j'}) &= 1 - \mid \rho_{jj'} \mid \\
\text{cosine-angle: } d(\mathbf{x}_j, \mathbf{x}_{j'}) &= 1 - \rho^0_{jj'} \\
\text{absolute cosine-angle: } d(\mathbf{x}_j, \mathbf{x}_{j'}) &= 1 - \mid \rho^0_{jj'} \mid
\end{aligned}$$

where

$$\rho^0_{jj'} \equiv \frac{\sum_{l=1}^{n} x_{jl} x_{j'l}}{\sqrt{\sum_{l=1}^{n} x^2_{jl}} \sqrt{\sum_{l=1}^{n} x^2_{j'l}}}.$$

Each dissimilarity metric defines a different choice of what it means for two elements to be close. Euclidean distance requires that the vectors be in the same direction and of the same magnitude. We have found that Euclidean distance is useful for clustering patients based on their gene expression vectors when we want to group patients based on similar magnitude patterns of gene expression. Correlation distance requires that the vectors have similar shape, while absolute correlation extends this to include opposite shape (*i.e.*: a large, positive value in one vector can be accompanied by a large positive or negative value in the other vector). Absolute correlation can be used to group together genes which are similarly up *or* down regulated across samples. Cosine-angle distance is similar to correlation, except that the vectors are centered around their own means, rather than being re-scaled to have mean zero in the correlation calculation. It is of interest to note that $1 - \rho^0_{jj'}$ equals 0.5 times the squared Euclidean distance of the two vectors standardized to have Euclidean norm 1. This distance was used in

Eisen *et al.* (1998), and it has been our experience that it is a sensible choice in many applications.

The clustering procedure PAM produces as output a set of cluster centers or "medoids". Let K be the number of clusters (*i.e.*: the number of causal mechanisms we believe to be operating) and let $\mathbf{M} = (M_1, \ldots, M_K)$ denote any size K collection of the n elements $\mathbf{x}_j$. Given $\mathbf{M}$, we can calculate the dissimilarity $d(\mathbf{x}_j\ , M_k)$ of each element and each member of $\mathbf{M}$. For each element $\mathbf{x}_j$, we denote the minimum and minimizer by $\min_{k=1,\ldots,K} d(\mathbf{x}_j\ , M_k) = d_1(\mathbf{x}_j\ , \mathbf{M})$ and $\min^{-1}_{k=1,\ldots,K} d(\mathbf{x}_j\ , M_k) = l_1(\mathbf{x}_j\ , \mathbf{M})$. PAM selects the medoids $\mathbf{M}^*$ by minimizing the sum of such distances $\mathbf{M}^* = \min^{-1}_{\mathcal{M}} \sum_j d_1(\mathbf{x}_j\ , M)$. Each medoid M^*_k identifies a cluster, defined as the elements which are closer to this medoid than to any other. This clustering is captured by a vector of labels $l(\mathbf{X}, \mathbf{M}^*) = (l_1(\mathbf{x}_1, \mathbf{M}^*), \ldots, l_1(\mathbf{x}_p, \mathbf{M}^*))$. An attractive property of PAM is that the clusters are identified by the medoids, which are elements themselves, and it has been our experience that the medoids are stable representations of the clusters.

One can consider K as given or it can be data-adaptively selected, for example, by maximizing the average silhouette as recommended by Kaufman & Rousseeuw. The silhouette for a given element is calculated as follows. For each element j, calculate a_j which is the average dissimilarity of element j with other elements of its cluster:

$$a_j = \text{avg}\, d(\mathbf{x}_j\ , \mathbf{x}_{j'}), j' \in \{i : l_1(\mathbf{x}_i, M) = l_1(\mathbf{x}_j\ , M)\}.$$

For each element j and each cluster k to which it does not belong (that is, $k \neq l_1(\mathbf{x}_j\ , M)$), calculate b_{jk}, which is the average dissimilarity of element j with the members of cluster k:

$$b_{jk} = \text{avg}\, d(\mathbf{x}_j\ , \mathbf{x}_{j'}), j' \in \{i : l_1(\mathbf{x}_i, M) = k\}.$$

Let $b_j = \min_k b_{jk}$. The silhouette of element j is defined by the formula:

$$S_j(\mathbf{M}) = \frac{b_j - a_j}{\max(a_j, b_j)}.$$

Note that the largest possible silhouette is 1, which occurs only if there is no dissimilarity within element j's cluster (*i.e.*: $a_j = 0$). The other extreme is -1. Heuristically, the silhouette measures how well matched an object is to the other objects in its own cluster versus how well matched it would be if it were moved to another cluster.

2.2 HOPACH-PAM algorithm

We first outline the proposed HOPACH-PAM algorithm (as we have implemented it in an Splus/R program) very generally. In subsequent subsections, we describe parts of the algorithm in more detail and discuss different options for implementation.

2.2.1 General outline

Consider a $p \times p$ dissimilarity matrix $\mathbf{D}$. Begin with all p of the elements in one cluster. Repeat the following steps in each level of the tree:

1. **Partition:** Apply PAM to the elements in each cluster;

2. **Order:** Order the new clusters;

3. **Collapse:** Possibly merge some clusters.

Stop when each cluster contains no more than 3 elements (3 is the minimum number of elements for clustering with PAM). The final level is an ordered list of the elements.

2.2.2 Partition step

The application of PAM to the elements in each parent cluster in any level to create child clusters for the next level is straightforward. The only option is how to select the number of child clusters k for each parent. In our implementation, we select each k by maximizing average silhouette over the range $k = 2, \ldots, K$, where K is a user-supplied maximum number of children per parent node. One could allow $k = 1$ (*i.e.* stop splitting a cluster) by using some criteria. For example, we have considered not splitting a cluster if the cluster size is below a cut-off value or if doing so results in a silhouette below a cut-off value. In the first level of the tree, it is not necessary to use PAM. For example, in the gene expression context, one might want to initially split the genes into those which are suppressed and those which are over expressed on average.

2.2.3 Order step

The key to producing a meaningful final ordered list of elements (and, similarly, ordered distance matrices for each level) is to order the clusters in each level in a sensible way using the same distance metric as in the partitioning step. Consider a set of k child clusters with medoids $M_1, \ldots, M_k$. Firstly, it is necessary to define a distance between clusters. We use the distance between the medoids. Since these are simply elements of $\mathbf{D}$, the ordering step is very quick. It would also be possible to use other pair-wise distances between clusters, such as the average or single linkage.

The ordering scheme we have implemented preserves the tree structure by keeping the children of each parent together and ordering them in the following way. Define their neighboring cluster as the cluster to the right of their parent (in the previous level) and denote its medoid M_N. Order the k clusters left to right from largest to smallest distance to the neighboring cluster. For the right-most set of child clusters, the neighboring cluster is to the left of their parent and we order from smallest to largest distance.

We have also implemented a function improveordering(), which finds the ordering of the elements of $\mathbf{D}$ that maximizes the correlation ordering, defined as the empirical correlation between distance $j - i$ in the list and the corresponding distance $d(\mathbf{x}_i, \mathbf{x}_j)$ across all pairs (i, j) with $i < j$. This function can be used to order the elements themselves or the clusters in any level (e.g: by using the distance between medoids). We do not use improveordering() as part of the usual HOPACH-PAM implementation because it can destroy the tree structure. It is of interest, however, to compare the ordering obtained by the above method with that of improveordering(). In our data

examples, the two methods usually produce the same ordering of the clusters, which indicates to us that our method is sensible.

The ordering of the $k1$ clusters in the first level of the tree requires a special method, because there is no parent cluster and hence no neighboring cluster as defined above. This initial ordering is important since subsequent orderings follow from it. If $k1 = 2$, then the ordering does not matter. If $k1 > 2$, then we have three proposals for ordering the $k1$ clusters. The first is to order the clusters according to mean expression (across the n samples) of the medoids. This method only makes sense when mean expression is a good summary measure of the cluster profile. The second is to apply improveordering() to the clusters' $k1 \times k1$ distance matrix D^{k1}. The third is to build a hierarchical tree from the $k1$ medoids (*i.e.*: to apply HOPACH-PAM to the medoids) with only 2 clusters in the first split. By running down the tree, we obtain a unique ordering of the $k1$ medoids. We use this third method in our usual implementation.

2.2.4 Visualization

For any level of the tree, it is interesting to visualize the data and distance matrices corresponding with an ordering of the elements in that level. The clusters are already ordered. The elements in each cluster can be further ordered or left as in the original data matrix (*i.e.* randomly). We order the elements within each of the clusters by either (i) their distance with respect to the medoid of that cluster so that the badly clustered elements end up at the edge of these clusters or (ii) their distance with respect to the medoid of the neighboring cluster. We can use this ordering to rearrange the rows and columns of the distance matrix. Then, we can visualize the ordered distance matrix by mapping the distances into a color scheme (*e.g.* as in Eisen *et al.* (1998)). The ordered data matrix can be visualized similarly (ordering only the dimension that is being clustered).

2.2.5 Collapse step

Due to the nature of hierarchical trees, two or more clusters in a level can be very similar. These may share a parent or not. Such clusters can often be identified by visual inspection of the ordered distance matrix. We also allow the decision to collapse to be based on a criteria which is compared before and after the collapsing step. In particular, we implement a function which collapses until there is no pair of clusters for which a collapse improves average silhouette for the whole level. Collapsing is performed by giving the labels of one cluster to the other, so that the tree structure is preserved. The choice of labels can be arbitrary (*e.g.*: the right-most cluster's) or based on similarity of the old medoids to the new neighboring cluster or some other criteria. A new medoid is chosen for the merged cluster. We have implemented several options for selecting the new medoid, such as the nearest neighbor of the (possibly weighted) average of the two old medoids and the maximizer of the cluster's average silhouette. Agglomerative algorithms, such as **Cluster** and AGNES, are equivalent to applying collapsing steps beginning at the bottom of the tree with each element initially in its own cluster.

2.2.6 Labels

A convenient way to keep track of the tree structure is to assign each element a label indicating it's path through the tree. At each level, the previous labels are extended with another digit representing the new cluster label. Thus, a typical label of an element at level 4 in the tree looks like 1321. If a cluster is not split (*i.e.* $k = 1$), then the label of each of the elements in that cluster is extended with the digit 0 so such a label will look like 13210. If the clusters in each split are numbered sequentially from left to right beginning with 1, then the numerical ordering of the labels in the final level corresponds with the ordering of the elements produced by the tree. The whole tree can be reconstructed from the final labels.

2.2.7 Automation

Most of the HOPACH-PAM algorithm is deterministic given a distance matrix $\mathbf{D}$, but there are several places where decisions can be made by the user. These steps of the algorithm can also be automated so that it runs without user input. As discussed above, the selection of the number of clusters in each PAM partitioning step can be done automatically by maximizing average silhouette over a range of possible values. The collapsing step can also be automated by selecting a criteria, such as average silhouette, to optimize and collapsing pairs of clusters until there is no more improvement in the criteria (greedy search) or collapsing any pair of clusters that improves the criteria (exhaustive search). It has been our experience that the collapsing step is most important at the initial levels of the tree, so that one can run down the tree automatically without further visual inspections after several initial iterations.

2.3 Applications of the algorithm

2.3.1 Final ordered list of elements

The final ordered list of elements has two important applications. The list itself is useful since elements close in the list will be close in distance to each other. For example, given any gene of biological interest, we can find similarly expressed genes nearby in the list. Visualization of the distance matrix corresponding with the final ordering of all elements is also of interest, since it shows clusters at all levels of detail.

2.3.2 Selecting the main clusters

With any clustering analysis, it is of interest to identify groups of related elements and to select the number of such groups. In the hierarchical tree context this corresponds with choosing the level l^* of the tree at which the clusters are still significant. We refer to the clusters in level l^* as the "main" clusters. There are many methods for selecting the main clusters. A method that we have found very useful is to inspect the ordered distance matrices $\mathbf{D}^l$ at various levels l and compare these to the final ordered distance matrix. It is often possible to identify the first level l^* which already closely resembles the important clustering structures in the final level. In other words,

the clusters are already homogeneous enough at level l^* that the changes in ordering produced in subsequent levels are not significant visually.

We have also implemented a completely automated HOPACH-PAM which has as its only goal to find the main clusters. This is done by moving down the tree and accepting the collapsed next level of the tree only if its average silhouette improves on the average silhouette of the previous level of the tree. Alternatively, a more exhaustive search can be made. We have noted, however, that average silhouette tends to be a global criteria in the sense that it is not necessarily maximized at the level of the tree which we would select visually (but rather usually higher up in the tree). Hence, we have proposed an alternative criteria, Mean Split Silhouette (MSS), for determining the number of significant clusters Pollard & van der Laan (2002b). MSS can be used in the same way as average silhouette to run down the tree until there is no more improvement or in a more exhaustive search.

A final approach to selecting the main clusters is to use the bootstrap (as described below) to estimate the variability of the clusters at different levels of the tree and choose as main clusters the lowest level which is reproducible at a certain level of stringency. In this manner, one can use the bootstrap to establish the level of detail which cannot be distinguished anymore from the noise level.

2.4 Code and running time

We have implemented HOPACH-PAM as a collection of R/Splus functions, which are available by request. The running time of the algorithm depends upon the computer used, the size of the data set (on p much more than n), and how the tree unfolds. As an example, consider a data set with $p = 4500, n = 30$. We use a Dell Pentium workstation with 2.2GHz processor and 4 gigabytes of RAM. Computation of the $p \times p$ distance matrix takes 7.5 minutes, but need only be done once. The first few levels of the tree are the most time consuming to partition, each taking between 2 and 4 minutes. By the tenth level of the tree, the partitioning takes less than 30 seconds. Collapsing is time consuming, particularly if an exhaustive search of all possible collapses is done. In the lower levels of the tree (with possibly hundreds of clusters), collapsing can be quite slow. In practice, we have found that most of the clustering structure is revealed in the top levels, so that after identifying the main clusters with collapsing, it is usually sufficient to run the tree down from that level without collapsing. To run down the tree completely without collapsing takes 12 minutes (excluding computation of the distance matrix). In comparison, a data set with $p = 500, n = 60$ requires only 5 seconds for distance matrix computation, 3 seconds for an initial level, 1.5 minutes to find the main clusters with collapsing, and 28 seconds to rundown completely without collapsing.

3 The bootstrap

Let μ_n, Σ_n be the $p \times 1$ empirical mean and $p \times p$ covariance matrix of the data X. Since (for the previously mentioned dissimilarities) the distance matrix is a function of μ_n, Σ_n, we can view each particular HOPACH-PAM algorithm, as defined by the various choices

such as the level l^* of the tree, as a particular functional $(\mu, \Sigma) \to S(\mu, \Sigma)$ applied to (μ_n, Σ_n). Therefore we can consider the HOPACH-PAM clustering result $S(\mu_n, \Sigma_n)$ as an estimator of a true clustering parameter $S(\mu, \Sigma)$. The main cluster labels (at level l^* of the tree) and the final ordering are examples of clustering parameters which may be of interest.

In order to establish the variability and reproducibility of the clustering output $S(\mu_n, \Sigma_n)$, we propose to run the parametric or nonparametric bootstrap. This involves repeatedly sampling n observations $X_1^\#, \ldots, X_n^\#$ from a multivariate normal distribution $N(\mu_n, \Sigma_n)$ (van der Laan & Bryan (2001)) or from the empirical distribution which puts mass $1/n$ on each of the original observations $X_1, \ldots, X_n$. One estimates the distribution (and, in particular, the variance) of the clustering output $S(\mu_n, \Sigma_n)$, with the empirical distribution of $S(\mu_n^\#, \Sigma_n^\#)$. We have demonstrated the asymptotic validity of the bootstrap in this context (Pollard & van der Laan (2002a)).

One useful application of the bootstrap is to estimate the variability of cluster membership at different levels of the tree. This information can be used, for example, to select as main clusters the lowest level which is reproducible at a certain level of stringency or to measure the reliability of clusters identified in another way. In order to estimate the variability of specific clusters, it is necessary to have a correspondence between the original clusters and those in each bootstrap sample. This correspondence can be achieved by (1) fixing the medoids or (2) only fixing the number of clusters and then aligning the clusters (as described in the appendix). One can keep track of the proportion of times among the bootstrap samples that an element fell into each of the clusters. van der Laan & Bryan propose a cluster-probability plot to summarize these statistics which provides a visual way to inspect the cluster reproducibility, where the clusters are ordered as in $S(\mu_n, \Sigma_n)$. The bootstrap cluster-specific probabilities can also be used to post-screen badly clustered genes. In addition, van der Laan & Bryan propose a sensitivity and positive predictive value measure for each cluster measuring proportions of correct genes and proportions of false positives.

Alternatively, we might chose to estimate the variability of the number of clusters by not fixing the medoids or the number of clusters for the bootstrap samples. In this case, we can not infer a correspondence between clusters, so that it is no longer possible to compute cluster-specific measures. Summary measures, such as average silhouette and the ordered distance matrix, however, can still be computed. Furthermore, for every pair of elements, we can keep track of the proportion of bootstrap samples in which they appeared together in the same cluster, which is a useful measure of cluster reproducibility. These different bootstrap methods are explained in more detail in an appendix.

4 Simulations

In order to evaluate the clustering performance of HOPACH-PAM relative to existing algorithms, we have conducted two sets of investigations. In the first set, we look at the final ordering produced by HOPACH-PAM compared to two other hierarchical methods, DIANA (divisive) and AGNES (agglomerative, single linkage). We run

the HOPACH-PAM tree to the final level without collapsing. In the second set, we investigate the ability to identify the main clusters. We compare HOPACH-PAM to two partitioning methods, PAM and KMEANS, since identifying clusters is the goal of these methods (and not of most hierarchical methods). We use an automated version of HOPACH-PAM which collapses clusters in each level until there is no more improvement in average silhouette and stops at the first level where the percent improvement in average silhouette moving to the next level is less than 10%. We select the number of clusters for PAM and KMEANS by maximizing average silhouette. Throughout, we chose to use the Euclidean distance so that KMEANS (which allows only this distance metric in its usual implementation) can be compared to the other algorithms in the context where it performs best.

For each section, we chose a data generating distribution in which the parameter of interest, either an ordered list or a vector of cluster labels, is meaningful. Both of the data generating distributions are described as gene expression experiments, although we acknowledge that their structures are simplifications of real gene expression data. The analogy to a gene expression experiment is, however, useful for describing the data.

4.1 Investigation 1: Ordered list

4.1.1 Data generating distribution

Consider a simplified gene expression data set consisting of 100 over and 100 under expressed genes, where genes showing no significant difference in expression have been prescreened. Suppose that the means $\{\mu_j \; j = 1, \ldots, 200\}$ of the over and under expressed genes are fixed numbers drawn from $N(log(3), 0.1)$ and $N(-log(3), 0.1)$ distributions, respectively. Let the genes be uncorrelated. Then, the true Euclidean distance between a pair of genes is $d(\mathbf{x}_j, \mathbf{x}_{j'}) = \sum_l (\mu_{jl} - \mu_{j'l})^2$. Since the genes are uncorrelated and the means follow a gradient within each of the groups, we have a data generating distribution in which the final ordered list is meaningful; we expect that genes with similar means should appear close to each other in the list. To generate simulated data from this distribution, we specify a variance for each gene (possibly the same for all genes) and then draw samples from the corresponding multivariate normal distribution (with diagonal covariance). As we describe below, a comparison of the clustering parameters for the different hierarchical algorithms is interesting itself, so that we do not report the results for estimation of these parameters using simulated data sets.

4.1.2 The clustering parameter

In this investigation, the parameter of interest is a final ordered list produced by applying each hierarchical clustering algorithm to the true distance matrix. We apply DIANA, AGNES and HOPACH-PAM (to the final level without collapsing) with (i) binary splits only and (ii) a maximum of 9 clusters at any node. For each algorithm, we obtain a final ordering of the $p = 200$ genes. These orderings are not the same, so each algorithm defines a different clustering parameter. Furthermore, all four final orderings vary when the order of the genes in the true distance matrix is permuted, but

this random component is *much* less for HOPACH-PAM than for DIANA and AGNES. For HOPACH-PAM, the randomness enters only through the choice of medoids in PAM (see discussion in Section 6), because the ordering method is deterministic and independent of the original ordering. The ordering methods of DIANA and AGNES, however, depend more directly on the original ordering (see also Section 5).

Since the four algorithms do not share the same clustering parameter, we do not report their performance in terms of estimation (as we do with cluster labels below). It is of interest, nonetheless, to compare the four parameters. We assess these orderings in two ways. First, we compute the correlation ordering (defined in Section 2). In order to account for the randomness in the final orderings, we permuted the original ordering of the true distance matrix randomly $B = 100$ times and report the mean and standard error of correlation ordering over these permutations for each algorithm. Second, we look at the corresponding ordered distance matrices. Because the true distance between genes is a function of the means, we expect to see a gradient in the ordered distance matrices from smallest to largest mean (or visa versa).

4.1.3 Correlation ordering

None of the algorithms is designed to maximize correlation ordering per se. We have seen, however, that correlation ordering is a good objective measure of an ordering. Table 1 shows the mean and standard error of correlation ordering across the $B = 100$ permutations for each algorithm. HOPACH-PAM has the best clustering parameter in terms of both the magnitude and the variance of correlation ordering. Binary HOPACH-PAM performs essentially the same as HOPACH-PAM, because the first split of the genes is binary (into over and under expressed groups) and the rest of the ordering method follows from the initial level. If there were more than two groups, we would not expect binary HOPACH-PAM to be so similar to the general HOPACH-PAM. DIANA and AGNES have clustering parameters with smaller correlation ordering, and these parameters are much more variable.

Algorithm	Mean (SE) correlation ordering
DIANA	0.71 (0.058)
AGNES	0.70 (0.043)
binary HOPACH-PAM	0.85 ($2.1 * 10^{-5}$)
HOPACH-PAM	0.85 ($2.1 * 10^{-5}$)

Table 1: Correlation ordering of the final ordered list for each clustering algorithm. The means and standard errors across $B = 100$ permutations of the true distance matrix are reported.

4.1.4 Visualization

Figure 1 shows the ordered distance matrices corresponding with a typical final ordering of the true distance matrix from each algorithm. The orderings of elements within the

over and under expressed groups are different with each algorithm. The two HOPACH-PAM orderings are nearly identical and produce nice gradients of the genes. These gradients are similar to that produced if the genes are simply ordered according to their means, as we would hope to see. DIANA and AGNES, however, do not produce such a gradient. The over and under expressed genes are separated, but the ordering of the genes within these groups does reveal the structure in the data set.

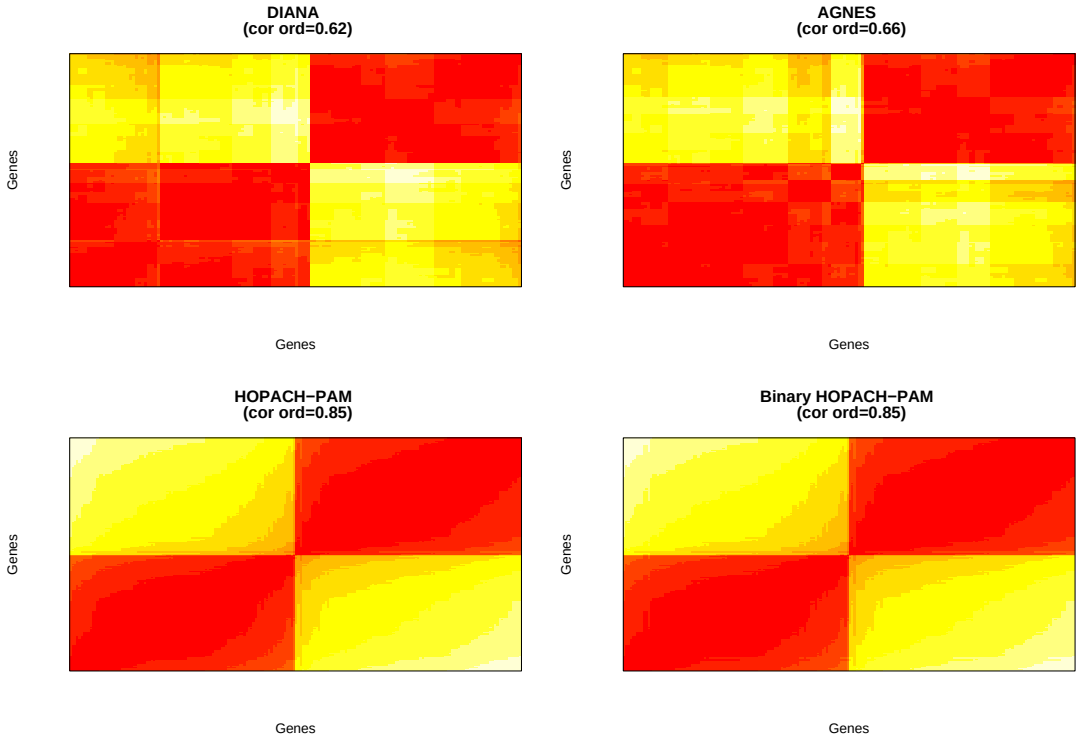

Figure 1: Plots of the ordered true Euclidean distance matrices of genes run to the final level with four different hierarchical algorithms. Red corresponds with smallest and green with largest distance. Each matrix is accompanied by its correlation ordering.

4.2 Investigation 2: Main clusters

4.2.1 Data generating distribution

We have designed a data generating distribution such that a clustering routine is needed to identify the underlying structure. Consider a sample of $n = 60$ relative gene expression profiles corresponding to cancer patients. Suppose that the data have been pre-processed and subsetted so that $p = 500$ genes of interest remain, a third of which have 3-fold differential expression in some patients. Let there be three groups of 20 patients corresponding with three distinct types of cancer. To generate such data, we sample three groups of 20 patients from three multivariate normal distributions with diagonal covariance matrices, which differ only in their mean vector. All genes have common standard deviation $\log(1.6)$, which corresponds with a 0.75-quantile of

all standard deviations in an actual data set. For the first subpopulation, the first 25 genes have $\mu_j = \log(3)$, genes 25-50 have $\mu_j = -\log(3)$, and the other 350 genes have mean zero. Then for the second subpopulation, genes 51-75 have $\mu_j = \log(3)$, genes 76-100 have $\mu_j = -\log(3)$ and the other 350 genes have mean zero. For the third subpopulation, genes 101-125 have $\mu_j = \log(3)$, genes 126-150 have $\mu_j = -\log(3)$ and the other 350 genes have mean zero. In other words, the cause of each of the three types of cancer is related to 50 genes of which 25 are suppressed (tumor-suppressor genes) and 25 are over-expressed (the onco-genes). All logs are base 10.

4.2.2 The clustering parameter

The parameters of interest are the cluster labels identified by applying HOPACH-PAM, PAM, and KMEANS to determine the main clusters. We use the automated HOPACH-PAM described above with (i) binary splits only and (ii) a maximum of 9 clusters at any node. We choose the number of clusters for PAM and KMEANS by maximizing average silhouette over $k = 2, \ldots, 9$. Given the data generating distribution, each of the algorithms defines a (possibly different) clustering parameter. These parameters can be identified by running the algorithms on the true Euclidean distance matrix, which is a function of the mean and covariance of the data generating distribution. All four algorithms have the same clustering parameter, which is six clusters of 25 genes and one large cluster of 350 genes.

The true cluster labels and distance matrix correspond with a true average silhouette of 1, since the elements in each group have dissimilarity zero with each other. KMEANS has difficulty converging for $k < 7$ using the true distance matrix. PAM, however, does converge and corresponds with average silhouettes of 0.64, 0.70, 0.77, 0.84, 0.92, 1, 0.30, and 0.30 for $k = 2, \ldots, 9$, respectively. The maximum is clearly at $k = 7$. Since the algorithms are estimating the same clustering parameter, the distance of average silhouette from this shared true average silhouette can be used as a measure of clustering performance. For example, if average silhouette for an algorithm is far from the true average silhouette then the cluster labels can not be correct. The converse is not necessarily true, so that when average silhouettes are similar and/or close to the true average silhouette, it is useful to also visualize the clustering result.

4.2.3 Simulations

We generate $B = 100$ data sets and permute the order of the genes in each data matrix randomly to avoid biasing the algorithms by having similar genes appear consecutively. From each data matrix, we compute the Euclidean distance matrix. To illustrate the need for clustering algorithms to identify the structure in this type of data, we simply order the genes from each data set by mean expression and make a picture of the corresponding Euclidean distance matrix. We see no obvious patterns within the over-expressed genes and suppressed genes. For each of the $B = 100$ data sets, we apply HOPACH-PAM, PAM, and KMEANS to determine the main clusters. We keep track of the number of clusters, the average silhouette, and the number of incorrectly clustered elements.

4.2.4 Number of clusters

Table 2 shows the number of data sets out of $B = 100$ in which each algorithm identifies a given number of clusters. HOPACH-PAM with a data-adaptive number of clusters at each node is the only algorithm that selects $k = 7$ clusters in the majority of the data sets. The binary HOPACH-PAM always selects $k = 2$ clusters, indicating the importance of not enforcing binary splits. PAM favors $k = 8$ clusters and KMEANS usually identifies $k = 6$ or 7 clusters. Thus, we see that choosing the number of clusters by maximizing average silhouette, as suggested by Kaufman & Rousseeuw, would usually not result in the correct number of clusters with KMEANS or PAM.

Algorithm	$k =$ 2	5	6	7	8	9	10	11	12	13	14
KMEANS	2	9	35	37	14	3					
PAM	1	1	7	13	78						
binary HOPACH-PAM	100										
HOPACH-PAM	1		6	62		7	11	3	2	3	5

Table 2: Distribution of the number of clusters for four algorithms in $B = 100$ data sets. Blank cells correspond with zeros. The true number of clusters is $k = 7$.

4.2.5 Number of incorrectly clustered genes

When seven clusters are identified in a simulated data set, we can compare the cluster labels to the true labels. Table 3 shows the mean and standard error of the number of incorrectly clustered genes across all data sets with $k = 7$ clusters for each algorithm. Even when they identify the correct number of clusters, PAM and KMEANS both label a number of genes incorrectly about half of the time. HOPACH-PAM, in contrast, usually labels all genes correctly when $k = 7$. Binary HOPACH-PAM never selects $k = 7$ clusters, so the mean is omitted.

Algorithm	Data sets with $k = 7$	Data sets with no errors	Mean (SE) number incorrect
KMEANS	37	16	52.8 (67.7)
PAM	13	7	46.5 (61.5)
binary HOPACH-PAM	0	0	-
HOPACH-PAM	62	56	0.2 (1.0)

Table 3: Number of incorrectly clustered genes. The number of data sets with $k = 7$ and the number with no errors are reported for each clustering algorithm, along with the mean and standard error across all data sets with $k = 7$ clusters.

4.2.6 Average silhouette

Table 4 shows the mean and standard error of average silhouette across all $B = 100$ data sets and across the data sets with $k = 7$ clusters for each algorithm. According to this criteria, HOPACH-PAM performs better than the existing algorithms since it is less variable and produces higher average silhouettes. Both KMEANS and PAM are quite variable, sometimes producing average silhouettes as high as HOPACH-PAM. PAM also produces many average silhouettes below 0.1 and is therefore more variable overall and when $k = 7$. On average, KMEANS performs better than PAM in terms of maximizing average silhouette. Binary HOPACH-PAM produces surprisingly high average silhouettes even though it always identifies only $k = 2$ clusters. Mean average silhouettes are higher for the data sets with $k = 7$, particularly for PAM. KMEANS, PAM and HOPACH-PAM all have a mean average silhouette of 0.24 (with very small standard error) for the data sets with no errors.

Algorithm	Mean (SE) Avg Sil all data sets	Data sets with $k = 7$	Mean (SE) Avg Sil for $k = 7$ only
KMEANS	0.23 (0.012)	37	0.24 (0.0074)
PAM	0.11 (0.053)	13	0.19 (0.075)
binary HOPACH-PAM	0.20 (0.0073)	0	-
HOPACH-PAM	0.24 (0.0073)	62	0.24 (0.003)

Table 4: Average silhouettes for each clustering algorithm. The means and standard errors across all $B = 100$ simulated data sets and across only those data sets with $k = 7$ clusters are reported. The "true value" is computed using PAM on the true Euclidean distance matrix.

4.2.7 Visualization

In order to investigate how well each algorithm is able to identify the seven clusters and the degree to which average silhouette measures this success, we plotted the reordered distance matrix for each data set according to the cluster labels from each algorithm. The performance of the algorithms varied greatly, although all of the algorithms identified the clusters perfectly at least once. As noted above, the algorithms do not always choose the correct number ($k = 7$) of clusters, but even when the number of clusters is correct some errors are made.

Figure 2 illustrates the ordered Euclidean distance matrices from a typical data set. KMEANS has six clusters, splitting the mean zero genes and combining two pairs of smaller clusters. This result may be a consequence of the lack of robustness of cluster means. Such erroneous clustering results for KMEANS have relatively high average silhouettes (> 0.15), because the errors only affect the average silhouettes of a small group of genes. PAM also splits the mean zero genes into two clusters, forming eight clusters. In data sets with $k = 7$ clusters, PAM usually still splits the mean zero genes and combines one half with a smaller cluster. This behavior makes sense since PAM minimizes the sum of the distances to the closest medoid and splitting the mean

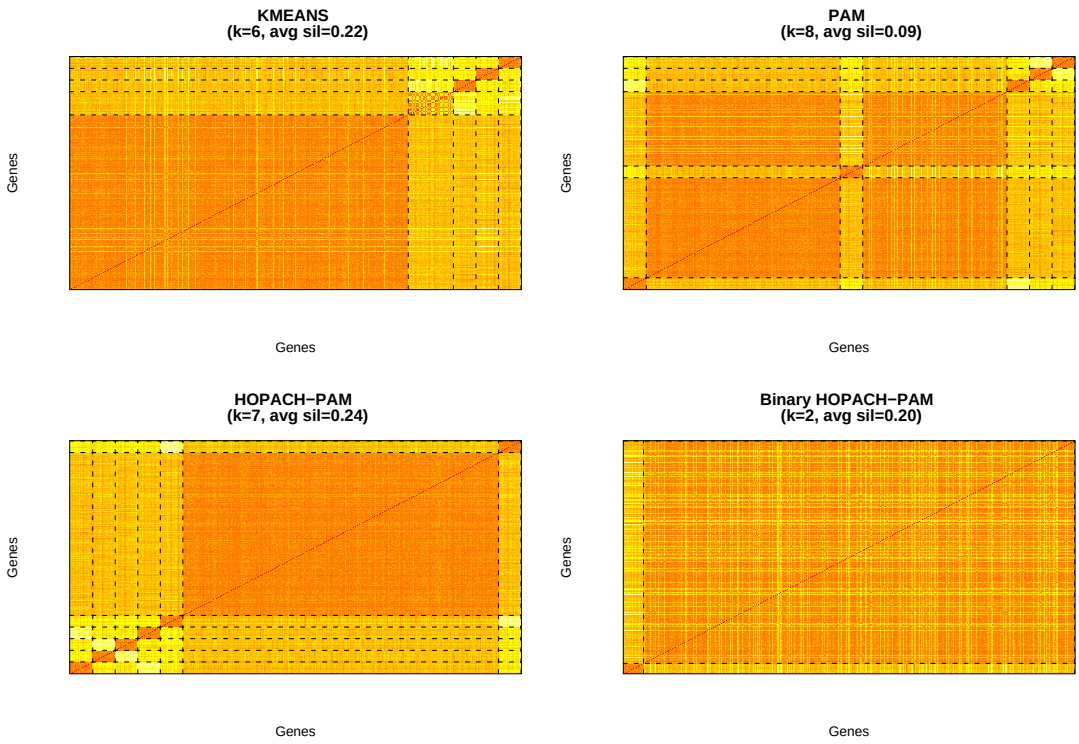

Figure 2: Plots of the ordered Euclidean distance matrices of genes from a simulated data set clustered with four different algorithms. Genes clustered together appear consecutively, but the clusters are not ordered. Lines mark the boundaries between clusters. Red corresponds with smallest and white with largest distance. Each clustering result is accompanied by the number of clusters and average silhouette.

zero genes reduces the distance to the closest medoid for many genes. Such erroneous clustering results for PAM always result in an average silhouette less than 0.10, since splitting of the ≈ 350 mean zero genes creates two clusters whose neighbors are very close, making the silhouettes of the genes in these two clusters very small. HOPACH-PAM finds the correct clusters and has a high average silhouette. In the occasional data set where HOPACH-PAM chooses the wrong number of clusters, it usually splits one of the small clusters. Binary HOPACH-PAM has only two clusters, but a relatively high silhouette.

4.2.8 Variations in the simulation

Number of genes. We repeated the simulations with many more genes, but cluster sizes of the same proportions. The trends are the same as those for 500 genes.
Number of samples. Since the clustering algorithms estimate the same clustering parameter, their differing performance is the consequence of their having different efficiencies for a fixed, relatively small number of samples ($n = 60$). In order to investigate the asymptotic consistency, we repeat the simulation for $n = 1200$ samples. PAM always has a maximum average silhouette at $k = 7$, but KMEANS has a maximum

average silhouette at either $k = 6$ or 7. KMEANS is very sensitive to its starting values, only some times finding the true clusters (even when we fix $k = 7$). In contrast, PAM always finds the true clusters. HOPACH-PAM also always identifies the true seven clusters. Hence, we see that for n large enough the outputs of PAM and HOPACH-PAM consistently estimate the same clustering parameter. For a fixed size data set with $n << p$, however, HOPACH-PAM is more efficient at estimating this parameter.

Using visualization. Rather than applying the automated HOPACH-PAM, it is also possible to cluster the simulated data sets manually by visualizing the ordered distance matrix at each level of the tree. When this method is used, it is clear which clusters need to be collapsed since the ordered distance matrix contains off-diagonal blocks of solid color corresponding with the correlated genes which have been put in different clusters in the first level of the tree. By collapsing such pairs of clusters, the true seven clusters are easily identified in the second level of the tree.

Distance metric. We repeated the simulations multiplying each gene vector by a random number $a \in [1, 10]$. The genes in the six small clusters are no longer close to each other with respect to Euclidean distance, but they are still close to each other with respect to cosine-angle distance since multiplication by a constant changes the magnitude, but not the angle, of the vectors. The mean zero genes are no longer close to each other with respect to either metric. We apply each clustering algorithm to $B = 100$ such data sets, using cosine-angle distance in HOPACH-PAM and PAM and Euclidean distance in KMEANS (since the usual implementation of KMEANS does not allow cosine-angle distance). As expected, KMEANS can not identify the six small clusters. PAM and HOPACH-PAM do identify the six small clusters, but again HOPACH-PAM is more efficient for small n. With PAM and HOPACH-PAM (using cosine-angle distance), the mean zero genes do not form a cluster and get assigned in each data analysis to one of the six small clusters. We often find "noisy" genes like these in real data analyses and have noted that many clustering algorithms have trouble identifying the true underlying cluster structure in the presence of much noise. The PAM family of algorithms has the nice property that the medoids are a good representation of the true clustering patterns even in the presence of such noise.

5 Data Analysis

We have extracted a publicly available data set from the data base accompanying Ross *et al.* (2000). The authors performed microarray experiments on 60 human cancer cell lines (the NCI60) derived from tumors from a variety of tissues and organs by researchers from the National Cancer Institute's Developmental Therapeutics Program. The data set includes gene expression measurements for 9,996 cDNAs representing approximately 8,000 unique transcripts. Each tumor sample was cohybridized with a reference sample consisting of an equal mixture of twelve of the cell lines chosen to maximize diversity. We use the normalized tumor:reference ratios, as in Ross *et al.* (2000). These are transformed to a log base 10 scale and truncated above and below, so that any ratio representing greater than 20-fold over or under expression is set

to $\log_{10}(20)$. We apply HOPACH-PAM, always using the cosine-angle distance and ordering relative to the neighboring cluster medoid.

5.1 Gene clustering

In order to create a real example that might represent a similar problem to that explored in the simulation, we select three very different types of cancer from those included in the NCI60: leukemia, colon and melanoma. We create a data set with all samples from these three types of cancer, which includes six leukemia, seven colon, and eight melanoma cell lines ($n = 6 + 7 + 8 = 21$). Next, we apply a subset rule in order to select all genes differentially expressed in a significant proportion of samples. We retain those genes where at least 25% of cell lines have a ratio corresponding with greater than 2-fold over or under expression. The 25% cut-off is chosen so that if a gene is differentially expressed in one type of cancer and not the other two, it will still be included. HOPACH-PAM is used to cluster the resulting data set of 3445 genes.

In the first level of the hierarchical tree, average silhouette suggests strongly a split into $k = 2$ clusters (Figure 3, panel 1). These clusters corresponded closely (but not exactly) with a split into the genes with negative and positive mean log ratios across the cell lines (*i.e.:* genes over and under expressed on average).

In the second level of the tree, each of the two clusters from level 1 is split into two clusters. When these are reordered, some structure begins to appear in plots of the data and distance matrices, but the clusters still seem heterogeneous and no collapsing appears to be necessary (Figure 3, panel 2). So, the third level of the tree is examined. In this level, two of the clusters from the previous level are split into four clusters each and two into two clusters each. The resulting twelve clusters are reordered and the distance matrix is examined (Figure 3, panel 3). Correlation between several clusters which had been united in a large cluster in the previous level indicates that some collapsing might be necessary. The clusters requiring collapsing are over expressed genes, which we have found tend to have more homogeneous expression profiles. After two collapsing steps, the reordered distance matrix has the desired block diagonal structure (Figure 3, panel 4). The resulting ten medoids and corresponding clusters of genes are identified as the main clustering result. The tree is then run down completely from this level.

As discussed above, there are several ways to use the bootstrap to assess the variability of HOPACH-PAM clustering results. As an illustration, we apply PAM with $k = 10$, fixing the ten medoids identified in level three after two collapsing steps in each bootstrap sample. Figure 4 shows the cluster probabilities for each gene. As a final step, we could perform post-screening and remove genes with lowest probability of belonging to their cluster or greatest distance from their cluster medoid.

Gene clustering is also performed with the automated HOPACH-PAM. As we have seen in other data sets, if allowed to do so, the automated HOPACH-PAM will converge to the first level of the tree with two clusters containing essentially the over and under expressed genes. If we want a more detailed clustering result, we can require that collapsing not occur until a lower level of the tree. In this case, the automated HOPACH-PAM clustering result is similar to the one identified manually. There are 8

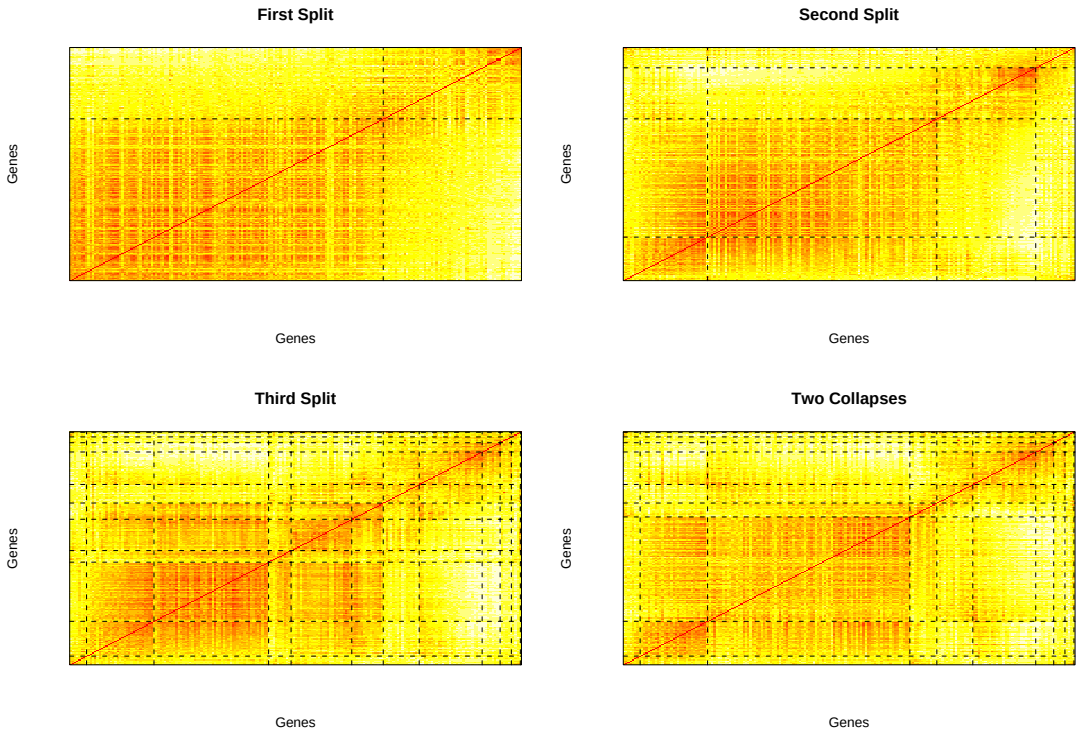

Figure 3: Plot of the ordered distance matrices of genes from the NCI60 leukemia, colon, and melanoma data set for different levels of the HOPACH-PAM tree. Lines mark the boundaries between clusters. The genes are ordered with respect to the neighboring medoid. Red corresponds with smallest and white with largest distance. Panels 1 to 3 show the first three levels without collapsing. Panel 4 shows level 3 after two collapsing steps.

clusters identified in the third level of the tree, and collapsing is mostly applied to the over expressed genes.

5.2 Cell line clustering

HOPACH-PAM can be applied to a transposed data matrix to obtain a final ordering of the samples. This ordering can be used to make an improved version of the reordered data matrix, where both genes and samples are ordered. The order obtained in this way for the cell lines in the leukemia, colon and melanoma data set is perfect in the sense that the three types of cancer are totally separated from each other (See Table 5, column 1). This result is explored further in the comparison section.

It is also of interest to cluster the cell lines using only the genes from each of the 10 gene clusters. Figure 5 shows the data matrices for each gene cluster with the cell lines reordered uniquely within each gene cluster. The cell line clustering results are of several types. For most gene clusters, the cell line clustering labels correspond extremely well with distinctions between the different types of cancer. In many cases, there are two clusters with one type of cancer clustering by itself (*e.g.*: gene cluster

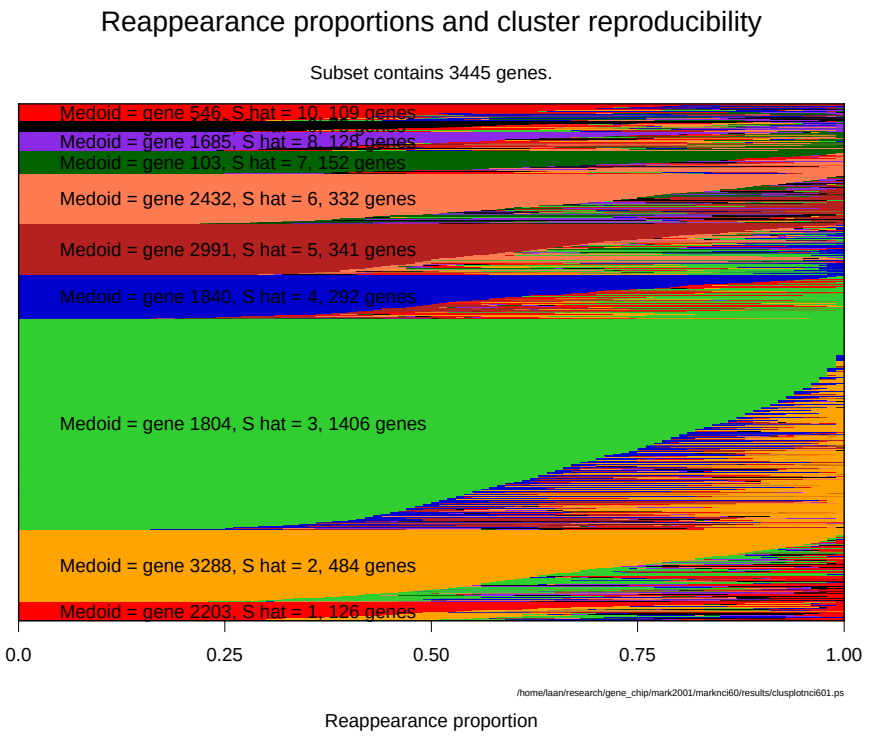

Figure 4: Cluster probability plot for NCI60 leukemia, colon, and melanoma data set using the 10 cluster medoids identified in level 3 of the HOPACH-PAM tree after two collapsing steps. Genes are ordered within cluster by their reappearance probabilities We can see which clusters are most stable (wide bars) and which pairs of clusters are most likely to exchange genes with one another (*e.g.*: clusters 3 and 4).

4). In several cases, there are three clusters containing each of the types of cancer (*e.g.*: gene cluster 6). We could use these gene cluster specific cell line cluster labels to more accurately characterize the three types of cancer. For example, a gene cluster where two of the types cluster together might represent a biological pathway which is activated (or not activated) in two of these cancers, but not the other. A few gene clusters produce a cluster result which does not correspond with the distinction of the three types of cancer (*e.g.*: gene cluster 8). These gene clusters are also of interest as they may include genes which differentiate cell lines on some other biological basis, such as tumor severity. In experiments with patient samples, this sort of clustering result can be correlated with clinical outcomes, such as survival or treatment history.

5.3 Comparison with other hierarchical clustering methods

We compare the final ordering from HOPACH-PAM to orderings produced by other hierarchical clustering routines and our proposed improveordering() function, which maximizes correlation ordering. We use HOPACH-PAM without collapsing because we are interested in a final ordering of the cell lines rather than the main cluster results. Our knowledge of the cell line labels is not used to supervise the clustering

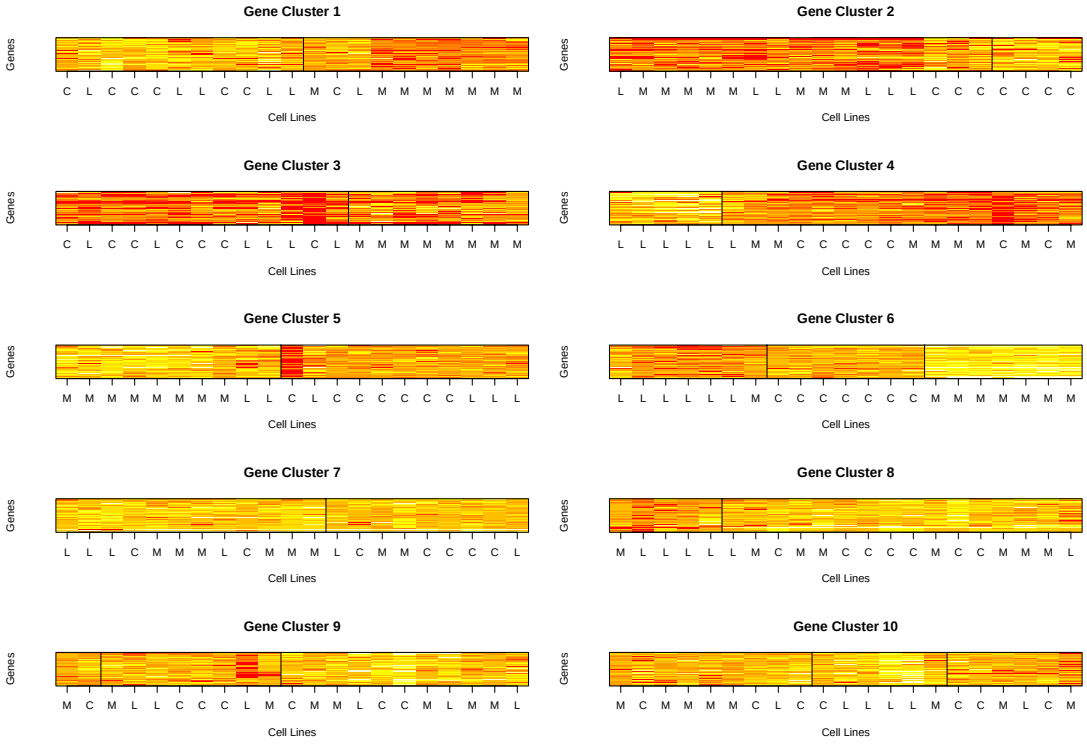

Figure 5: Data matrices for each of the 10 gene clusters found in the NCI60 leukemia, colon, and melanoma data set. The color scheme is on a scale from red to white (no green), with red representing the lowest expression in *that* cluster of genes. The genes are reordered (as in Figure 3, panel 4) according to distance from the neighboring medoid. The cell lines are reordered in each gene cluster according to the final ordering after applying HOPACH-PAM without collapsing. Lines mark the boundaries between the cell line clusters from the first level of the tree (*i.e.*: after simply applying PAM once to the cell lines with data-adaptive numbers of clusters). The first level PAM clusters do not always correspond with the visible clusters, while the orderings seem sensible. These plots and the corresponding distance matrix plots indicate that a few clustering results (*e.g.*: gene cluster 2) could be improved if the usual HOPACH-PAM method were employed and collapsing were applied at a lower level of the tree to determine the medoids. We found one melanoma cell line (LOXIMVICL10001) with unique expression that often clustered away from the remaining melanomas in these plots (*e.g.*: gene cluster 6). One colon cell line (HT-29CL4001) had very low under-expression of the genes in several of these plots (*e.g.*: gene cluster 5).

of cell lines, but rather to check (after clustering) if the various orderings are able to separate different types of cancer. We consider:

1. improveordering() function
2. HOPACH-PAM (without collapsing) with data adaptive numbers of clusters between 2 and 6 at each split (based on silhouettes)
3. HOPACH-PAM (without collapsing) with binary splits only

4. DIANA algorithm for divisive hierarchical clustering as implemented in Splus Kaufman & Rousseeuw (1990)

5. AGNES algorithm for average linkage agglomerative clustering as implemented in Splus Kaufman & Rousseeuw (1990)

6. AGNES algorithm for single linkage agglomerative clustering as implemented in Splus Kaufman & Rousseeuw (1990)

7. Average linkage agglomerative clustering as implemented in the publicly available packages **Cluster** and **TreeView** Eisen *et al.* (1998)

8. Single linkage agglomerative clustering as implemented in the publicly available packages **Cluster** and **TreeView** Eisen *et al.* (1998)

The improveordering() function is not a clustering routine, but simply a proposed method for ordering elements using any distance metric. The other methods are all hierarchical clustering routines that produce an ordering of the clustered elements. HOPACH-PAM without collapsing and DIANA are divisive hierarchical methods, and the other algorithms are agglomerative methods. We use the cosine-angle distance in all of the algorithms. The data set was selected *a priori*, not based on the performance of the algorithms. We repeat the analysis with randomly permuted data matrices in order to account for any effect of the original ordering of the cell lines on the final orderings produced.

These algorithms are applied to the leukemia, colon and melanoma data set (See Table 5). The improveordering() algorithm separates the three types of cancer and places colon between leukemia and melanoma. The two versions of HOPACH-PAM produce the same ordering as each other, which also places the three types of cancer totally separately with colon lying between leukemia and melanoma. The DIANA algorithm places one melanoma cell line (also noted in Figure 5) with the leukemias. The AGNES algorithms places this same melanoma cell line with the colon cell lines. The **Cluster** algorithm (single and average linkage) keeps the colon cell lines together and distinct from melanoma, but spreads the leukemia and melanoma cell lines throughout the ordering. Next, we permute the order of the samples in the data set to test the dependence of the algorithms on the original order. HOPACH-PAM produces the same ordering regardless of the original ordering of the data set. The other algorithms, in contrast, place the groups of different types of cancer in different orders depending on the original ordering of the data set (*i.e.* they have a random component and are quite variable). This result confirms our observation in Section 4 that HOPACH-PAM produces a less variable ordering than DIANA or AGNES.

This analysis suggests that HOPACH-PAM can improve upon the orderings produced by currently employed hierarchical clustering algorithms. Although the binary splits and data-adaptive number of splits versions of HOPACH-PAM perform similarly here, we have seen evidence in simulations and data analyses that it is beneficial to allow greater than two clusters at each node of a HOPACH-PAM tree. The improveordering() function also produces an ordering which separates the cell lines by cancer

improveordering()	HOPACH-PAM $k \in [2,6]$	HOPACH-PAM $k = 2$	DIANA	AGNES average	AGNES single	**Cluster** average	**Cluster** single
MEL10008	MEL10001	MEL10001	MEL10001	MEL10001	MEL10001	MEL10001	MEL10001
MEL10005	MEL10020	MEL10005	LEUK7010	COL4003	COL4003	LEUK7010	LEUK7010
MEL10002	MEL10014	MEL10008	LEUK7003	COL4010	COL4010	LEUK7019	MEL10002
MEL10014	MEL10021	MEL10007	LEUK7006	COL4015	COL4015	LEUK7006	MEL10021
MEL10020	MEL10007	MEL10021	LEUK7008	COL4002	COL4002	LEUK7003	MEL10007
MEL10021	MEL10008	MEL10020	LEUK7005	COL4017	COL4017	MEL10005	MEL10020
MEL10007	MEL10002	MEL10014	LEUK7019	COL4009	COL4009	MEL10002	MEL10008
MEL10001	MEL10005	MEL10002	COL4003	COL4001	COL4001	MEL10007	MEL10014
COL4003	COL4009	COL4001	COL4010	LEUK7019	LEUK7019	MEL10021	MEL10005
COL4002	COL4015	COL4003	COL4015	LEUK7010	LEUK7003	MEL10014	LEUK7006
COL4015	COL4010	COL4017	COL4002	LEUK7003	LEUK7006	MEL10020	LEUK7003
COL4017	COL4017	COL4010	COL4017	LEUK7006	LEUK7008	MEL10008	LEUK7008
COL4009	COL4002	COL4002	COL4009	LEUK7008	LEUK7005	COL4001	LEUK7005
COL4010	COL4003	COL4015	COL4001	LEUK7005	MEL10005	COL4003	COL4001
COL4001	COL4001	COL4009	MEL10005	MEL10005	MEL10008	COL4009	COL4003
LEUK7019	LEUK7006	LEUK7019	MEL10008	MEL10008	MEL10014	COL4017	COL4009
LEUK7006	LEUK7003	LEUK7010	MEL10021	MEL10014	MEL10020	COL4002	COL4017
LEUK7003	LEUK7008	LEUK7003	MEL10020	MEL10020	MEL10021	COL4015	COL4002
LEUK7008	LEUK7005	LEUK7006	MEL10007	MEL10021	MEL10002	COL4010	COL4015
LEUK7005	LEUK7019	LEUK7008	MEL10014	MEL10007	MEL10007	LEUK7008	COL4010
LEUK7010	LEUK7010	LEUK7005	MEL10002	MEL10002	LEUK7010	LEUK7005	LEUK7019

Table 5: Orderings of the cell lines in the NCI60 leukemia, colon, and melanoma data set produced by different hierarchical clustering methods.

type. In fact, in data sets where clustering is not so clear, we have seen that this function performs better at creating an ordered list than any of the hierarchical clustering routines. Since improveordering() allows any distance metric, it is a flexible tool for creating an ordered list when the number of elements to order is not too large, such as the medoids of gene clustering (but not all genes) or the samples themselves, as illustrated here with cell lines. In addition to being slower than hierarchical clustering, improveordering() also has the disadvantage that it produces only an ordering and not a tree.

6 Discussion

HOPACH algorithms combine partitioning and collapsing steps to produce an ordered hierarchical tree and corresponding list of elements. This family of proposed algorithms has some interesting properties.

Employed with collapsing, HOPACH-PAM is a "hybrid" clustering method which uses both partitioning and agglomerative steps. Collapsing serves two different purposes. First, collapsing can correct the number of clusters k in a split, to allow $k = 1$ (no split) or to correct for the number of clusters in a split. Secondly, collapsing can combine clusters across branches in the tree. For example, a split of a heterogeneous parent cluster may produce a cluster which is more similar to clusters from another part of the tree than to the clusters with which it shares a parent. This group represents an error made at a higher level of the tree, which has only become apparent as the clusters got smaller and more homogeneous. Collapsing can unite this cluster with the most similar other cluster elsewhere in the level. By simply changing the cluster's labels to those of the other cluster (or visa versa), the structure of the tree is preserved. Thus, we do not just use collapsing to select the number of clusters in PAM at each node of the tree, but rather we use PAM to separate the elements into groups and then collapse to decide on the final clusters at that level. This recursive partitioning

and collapsing of clusters to form a tree can be likened to forward/backward step-wise searches in cross validation methods and the pruning of classification and regression trees. Contrary to the usual approach, however, we use a step-wise search to build the initial tree.

An intuitively appealing property of HOPACH is that the same distance metric is used for the partitioning, the collapsing, the ordering of clusters, and the ordering of the elements within clusters. DIANA and AGNES also use the same distance metric for clustering and ordering. **Cluster**, however, clusters with a user-specified distance metric and chooses amongst possible orderings using mean expression level or additional covariates.

Our implementation of HOPACH uses PAM as the partitioning algorithm. The PAM family of algorithms has some nice characteristics. Our simulations show, for example, that PAM and HOPACH-PAM are both consistent estimators of clustering parameters such as cluster labels. HOPACH-PAM is more efficient for small, fixed sample sizes. We specifically like employing the PAM algorithm in HOPACH-PAM for two reasons. First, we have found that the medoids of clusters, which are themselves elements in the cluster, are much more robust profiles of a cluster than the cluster means. Secondly, the clustering in HOPACH-PAM can be based on any choice of distance metric, such as absolute correlation, which we have found is of interest to biologists since it clusters together anti-correlated over and under expressed genes that may be part of the same pathway.

Our application of the HOPACH methodology to simulated and a publicly available data illustrates some of its strengths. One advantage of HOPACH over other hierarchical methods is that the nodes of the HOPACH tree need not be binary. Lifting the binary split restriction can aid in the identification of clusters and can also improve the final ordering produced, even when the algorithm is otherwise identical. The binary HOPACH-PAM has trouble identifying clusters and produces unusual final orderings in simulations with more than two clusters. In data analyses we have noted similar problems with binary algorithms.

Another advantage of HOPACH is that the ordering method is deterministic and does not depend on the order of elements in the data set. HOPACH provides a data-adaptive, sensible ordering of the clusters at each level of the tree. For **Cluster**, DIANA and AGNES, the ordering of the clusters at every level of the resulting tree is determined by the original ordering of the data. Because **Cluster** and AGNES start at the bottom of tree, there is no data-adaptive way to order the clusters at every level of the tree. On the other hand, HOPACH begins as a partitioning method so that it produces an ordering of the clusters (based on the distance metric) from the first level of the tree. To obtain such an ordering is particularly important for producing a final ordered list of genes which is unique and not dependent on the ordering in the original data set. This ordering is maintained after collapsing. As a divisive algorithm, DIANA could have a unique, data-adaptive ordering function, but does not. Many other ordering methods, such as multi-dimensional scaling and principal components, have been proposed in the literature. Most of these provide an ordered list (or higher dimensional set of points) only, and are not useful in the context of a hierarchical tree. We have restricted our analysis to orderings produced by trees plus

our improveordering() function; a comparison of the HOPACH ordering with a larger set of orderings remains for future work.

One note needs to be made, however, regarding the uniqueness of the ordering produced by HOPACH-PAM. Like many clustering routines optimizing a criteria, PAM solves a minimization problem with many local minima. In this sense, HOPACH-PAM shares (through the imperfect convergence of PAM) the dependence on the order of the original data matrix that we have noted in other algorithms. In particular, the choice of medoids in the PAM partitioning in the first level of the tree affects the subsequent ordering method and has a small effect on the final ordering. We could regain full uniqueness by rerunning PAM on a large number of permuted data sets (*i.e.*: starting values) and selecting the medoids which give the smallest sum of distances. We have noted, however, that this is not usually necessary since the cluster labels and even the final ordering are often very stable so that the impact of the original ordering of the data on the final ordering is *much* less for HOPACH-PAM than for DIANA, AGNES or **Cluster**.

In addition to describing a clustering algorithm, we have also suggested two important methods for interpreting the output of this (and other) algorithms. First, visualization of the ordered distance matrix is an innovative approach to identifying and illustrating the clustering structure. This tool is particularly useful in high dimensional settings, such as gene expression data. Second, HOPACH-PAM fits into the general statistical framework of van der Laan & Bryan (2001), so that we can treat the clustering output as an estimate of a true underlying parameter and perform inference with the bootstrap. Most commonly used clustering algorithms are deterministic functions of the data generating distribution, so HOPACH with many different choices of partitioning and collapsing methods will fit into the statistical framework. Viewing clustering output as a parameter also aids in the comparison of different methods, as in the simulations section of this paper.

The HOPACH methodology as we have presented it is intentionally general enough to allow for adaptations in specific parts of the algorithm. For example, a range of different partitioning algorithms and distance metrics can be used. We have also suggested multiple ways to perform the ordering step and to make the decision to collapse. We see this flexibility, along with the hybridizing of partitioning and agglomerative algorithms to produce a sensible ordering, as HOPACH's greatest strengths.

References

Chen, C. H. (2002). *Statistica Sinica,* **12**, 7–29.

DeRisi, J., Penland, L., Brown, P., Bittner, M., Meltzer, P., Ray, M., Chen, Y., Su, Y., & Trent, J. (1996). *Nature Genetics,* **14**, 457–460.

Eisen, M., Spellman, P., Brown, P., & Botstein, D. (1998). *Proc. Natl. Acad. Sci.* **95**, 14863–14868.

Kaufman, L. & Rousseeuw, P. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis.* New York: John Wiley & Sons.

Pollard, K. & van der Laan, M. (2002a). *Mathematical Biosciences,* **176** (1), 99–121.

Pollard, K. & van der Laan, M. (2002b). Technical Report 107 Group in Biostatistics, University of California. To appear in SCI2002 Proceedings.

Ross, D., Scherf, U., Eisen, M., Perou, C., Rees, C., Spellman, P., Iyer, V., Jeffrey, S., Van de Rijn, M., Waltham, M., Pergamenschikov, A., Lee, J., Lashkari, D., Shalon, D., Myers, T., Weinstein, J., Botstein, D., & Brown, P. (2000). *Nature Genetics,* **24**, 227–235.

Sokal, R. R. (1966). *Scientific American,* **Dec.**, 106–116.

Törönen, P., Kolehainen, M., Wong, G., & Castren, E. (1999). *FEBS Letters,* **451**, 142–146.

van der Laan, M. & Bryan, J. (2001). *Biostatistics,* **2**, 1–17.

APPENDIX

The bootstrap can be applied to HOPACH-PAM output in several different ways. The fixed medoids and fixed number of medoids at each node approaches establish a correspondence between the original clusters and those in each bootstrap sample so that the variability of specific clusters can be estimated. The fixed procedure only approach allows for estimation of the variability in the number of clusters, but does not permit calculation of cluster-specific bootstrap probabilities.

A Fixed medoids

Let the medoids from $S(\mu_n, \Sigma_n)$ be fixed as the medoids for every bootstrap sample. Assign the elements in each bootstrap sample to the closest medoid's cluster. Note that this bootstrap avoids estimating the variability in the selection of the medoids. Nonetheless, it is a sensible approach to understanding the variability of the specific clusters obtained in the data analysis.

B Fixed number of clusters at each node

Let the number of clusters at each node from $S(\mu_n, \Sigma_n)$ be fixed for every bootstrap sample. If the actual clustering was based on an interactive process with the user, then the steps defining $S(\mu_n, \Sigma_n)$ need to be treated as fixed in the bootstrap. Although the variability in number of clusters is not addressed, this bootstrap does measure the variability in the selection of medoids. This approach views the data adaptive (possibly interactive) selection of the number of clusters as a way to decide on a particular clustering parameter $S(\mu, \Sigma)$, and we use the bootstrap to estimate the variability of this particular parameter (but not variability in choosing the parameter). In order to infer a correspondence between each bootstrap cluster and one of the original clusters,

we propose to align the clusters in each split of the tree by examining the matrix of pair-wise distances between all bootstrap and original clusters in that split and consecutively matching the closest pairs. The distance between clusters could be based on the distance between medoids or a measure of the overlap in membership, such as $(A \cap B)/(A \cup B)$ or $P(A|B)/2 + P(B|A)/2$.

C Fixed procedure only

Repeat the clustering procedure (including selection of the number of clusters) independently on each bootstrap sample. Since the number of clusters is now also variable it does not necessarily make sense to align the bootstrap clusters with the original clusters and define cluster-specific variability measures. One can, however, now estimate the variability of the number of clusters. It helps to at least enforce an ordering of the bootstrapped clusters corresponding as close as possible to the ordering in $S(\mu_n, \Sigma_n)$ by comparing their medoids. In this case, visualizing a number of bootstrapped ordered distance matrices provides a very good sense of the variability of $S(\mu_n, \Sigma_n)$. In addition, for every pair of elements, we can compute the proportion of bootstrap samples in which they appeared together in the same cluster. The $p \times p$ matrix of these estimated pairwise reappearance probabilities can be ordered in the same way as $S(\mu_n, \Sigma_n)$ to give a visual representation of cluster reproducibility. Furthermore, the reappearance probability matrix can itself be clustered and ordered to find groups of elements that appear together in the original data set with high probability.