Automated Hierarchical Density Shaving and Gene DIVER

Gunjan Gupta Alexander Liu Joydeep Ghosh

{ggupta/aliu/ghosh}@ece.utexas.edu

IDEAL-2006-TR05*

Intelligent Data Exploration & Analysis Laboratory (IDEAL)

(Web: http://www.ideal.ece.utexas.edu/)

Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, Texas 78712
U.S.A.

September 18, 2006

Abstract

A key application of clustering data obtained from sources such as microarray, protein mass spectroscopy, and phylogenetic profile is the detection of functionally related genes. Typically, only a small number of functionally related genes cluster into one or more groups, and the rest need to be pruned. For such situations, we present Automated Hierarchical Density Shaving (Auto-HDS), a framework that consists of a fast, hierarchical, density-based clustering algorithm and an unsupervised model selection strategy. Auto-HDS can automatically select between clusters of different densities, present them in a compact hierarchy and rank individual clusters using an innovative stability criteria. Our framework also provides a simple yet powerful 2-D visualization of the hierarchy of clusters that can be very useful for further exploring the dense clusters in high-dimensional datasets. We present results on Gasch and Lee microarray datasets to show the effectiveness of our methods.

1 Introduction

In many real-world clustering problems, only a subset of the data actually needs to be clustered. This could be due to the fact that only a subset of our data actually clusters well (e.g., market-basket data, where only a subset of the customers exhibit consistent behavior) while the rest can be treated as a "don't care" set. In particular, many types of large, high-dimensional bioinformatics datasets exhibit the above property. For example, consider gene-expression datasets that measure expression levels of genes compared to a control across a few thousand genes over several experiments. The experiments typically cover only a specific "theme" such as stress-response, and therefore only a few genes related to the conditions show good clustering. From this data, biologists are interested in recovering clusters formed from small subsets of genes that show strongly correlated expression patterns [1]. Other types of biological data that share similar properties include protein mass spectroscopy and phylogenetic profile data.

A wide variety of parametric approaches [BMDG05] have been applied to exhaustively cluster the data points based on the assumption that each cluster is a member of some parametric family (e.g., Gaussians). However, in problems where the subset of data that clusters well is proportionately small compared to the overall dataset, the "don't care" points can overwhelm an optimization method that optimizes over all the data points. An alternative to this approach consists of a class of non-parametric clustering algorithms (e.g., [EKSX96, ABKS99]) that use *kernel density estimation* [JNSRL94] at each data point to find dense clusters, where the choice of the kernel determines the notion of density at a data point. In addition to being non-parametric, these density based clustering algorithms are also capable of clustering only a subset of data.

The first paper to exploit kernel density estimation for non-parametric clustering was perhaps [Wis68], which proposed an algorithm called *Hierarchical Mode Analysis*(HMA), that could also find a compact hierarchy of dense clusters. However, HMA seems to have gotten lost in time (it was published in 1968) and is not known to most current researchers. One reason could be that HMA is slow ($O(n^3)$). In this paper, we present an improved framework called Automated Hierarchy Density Shaving(Auto-HDS) that builds upon the HMA algorithm and greatly broadens its scope and effectiveness. These improvements include: (1) creating a faster ($O(n^2)$) algorithm appropriate for larger datasets; (2) the ability to use a variety of distance metrics including Pearson Distance, a biologically relevant distance measure; (3) a robust *unsupervised* model selection that enables discovery of small clusters of varying densities and pruning of large amounts of irrelevant data, and (4) a novel, effective visualization of the resulting cluster hierarchy.

The collection of these abilities in a single framework brings together a set of requirements and features that are in great demand in bioinformatics and are yet to be fulfilled satisfactorily by existing methods. High throughput biological datasets have several properties that match the Auto-HDS framework, including: (1) "dense subsets" within the datasets can often have variable densities; for example a large number of somewhat weakly correlated genes could form a cluster that is as important as a small number of highly correlated genes; both could in turn be surrounded by a large number of irrelevant genes, (2) biological sub-processes can form sub-clusters within clusters, (3) a large number of irrelevant genes need to be pruned, (4) the need for a completely unsupervised setting since there is often little or no labeled data

[1] Often such clusters map to biological processes that are involved in the specific context (e.g., stress).

for selecting clustering model parameters, and consequently (5) the need for visualization of the clustering hierarchy; Auto-HDS provides an extremely compact hierarchy, and an equally compact visualization of the dense clusters that can further aid in cluster comprehension and selection. Empirical tests of our framework on gene expression data show that we do indeed obtain very good results.

Finally, we have also developed a Java based product called Gene Density Interactive Visual Explorer (Gene DIVER) that exploits an efficient heap data-structure and a serialization API to provide a memory-efficient, scalable and platform independent implementation of our framework that also includes a sophisticated SWING-based visualization and interactive user interface.

Notation: Bold faced variables, e.g. $\mathbf{x}$ represent vectors whose i^{th} element are accessed as either x_i or $x(i)$. Sets of vectors are represented by calligraphic upper-case alphabets such as $\mathcal{X}$ and are enumerated as either $\{\mathbf{x}_i\}_{i=1}^n$ or $\{\mathbf{x}(i)\}_{i=1}^n$, where $\mathbf{x}_i$ or $\mathbf{x}(i)$ are the individual elements. $|\mathcal{X}|$ represents the size of set $\mathcal{X}$. Bold faced capital letters such as $\mathbf{M}$ represent 2-d matrices. $\mathbb{R}$ and $\mathbb{R}^d$ represent the domain of real numbers and a d-dimensional vector space respectively.

2 Related Work

A variety of density-based methods have been developed that use different notions of density to cluster a part of the data and to prune the rest. One of the most widely cited density based algorithms is DBSCAN [EKSX96]. In DBSCAN, given a point that has at least *MinPts* points enclosed by a hypersphere of radius ϵ centered at the point, all points within the ϵ sphere are assigned to the same cluster. DBSCAN is particularly well suited for low dimensional data (e.g., image data) due to the availability of efficient indices that allow a fast implementation. However, one issue is the choice of ϵ and *MinPts*; different choices can give dramatically different clusterings. OPTICS [ABKS99] proposed a visualization to make it easier to select these parameters and also supports the discovery of a hierarchy on which additional, interactive exploration can be achieved.

As mentioned earlier, HMA [Wis68] is perhaps the first method for finding dense regions in data while allowing the pruning of the remaining non-dense regions. HMA is based on the idea that, given a set of i.i.d. data points from a multivariate distribution, the "modes" or dense regions of the data set need not satisfy any particular parametric shape. Available non-parametric techniques of the time such as single link agglomerative clustering were unable to deal with "chaining," a problem that occurs when two valid clusters are connected by a chain of spurious points, a problem that density based methods are able to overcome.

In addition, one of the salient features of HMA is its ability to find a highly compact hierarchy representing all possible clusters/modes in the data. By searching for the location, extent, and the hierarchical relationship between all of the modes/dense regions in the data, HMA addresses two fundamental questions in clustering, namely, *how many clusters are there in the data, and where are they located*. HMA is able to find a compact hierarchy by recovering the actual modes or the generative distributions generating the data, which stems from its ability to ignore the less dense or "don't care" points while building the hierarchy. These properties make HMA an extremely powerful unsupervised method, and in many ways it was ahead of its time; one of the cluster labeling and selection methods suggested by [Wis68] results

in an algorithm whose output is identical to that of DBSCAN. Furthermore, the method in [Wis68] also contains a solution for selecting ϵ, a parameter in DBSCAN that is difficult to choose for high-dimensional datasets. We describe HMA in more detail in Section 3.3.

DHC [JPZ03] proposes a hierarchical grouping of biological time-series data that can be used to visually browse similar genes. Although the general idea and motivation of [JPZ03] seems related to what we propose in this paper, the algorithms and the key issues that our method resolves are significantly different. The cluster hierarchy built by DHC uses the heuristic of *attraction* that assumes the data is uniformly distributed in the original d-dimensional space. However, points in many real-life high dimensional data tend to reside in much lower dimensional manifolds [TdSL00].

Specialized algorithms have been proposed that address other issues with clustering biological data [LO02, STG$^+$03, STG$^+$03, CDGS04]. For example, discovering overlapping gene clusters is popular since many genes participate in multiple biological processes. Gene Shaving [Has00] repeatedly applies Principal Component Analysis in a greedy fashion to find small subsets of genes that show strong expression change compared to the control sample, and allows them to belong to multiple clusters. For each discovered subset, Gene Shaving *shaves* a fraction of least relevant genes at each iteration, and we reuse the word "shaving" in Auto-HDS in the same context. However, Gene Shaving is different from our method in many ways. Gene Shaving has an implicit Squared Euclidean distance assumption which does not handle additive and multiplicative co-expression patterns, an important normalization needed for clustering genes. While Gene Shaving requires the number of clusters k as an input, Auto-HDS finds k automatically. Gene Shaving greedily finds overlapping clusters one at a time; the next cluster is found by using orthogonalized residue obtained after removing the previous cluster. Gene Shaving repeats the shaving sequence multiple times to obtain multiple clusters, while HDS finds all the clusters from a hierarchy built by one shaving sequence. HDS performs shaving by ordering points by density, whereas Gene Shaving orders and shaves genes with least correlation with the principal component.

Table 1 compares key features of some of these approaches with our framework. Note that only Auto-HDS is capable of automatic cluster selection. The ability to perform robust clustering and model selection is a key aspect of our framework. The set of all the selected clusters need not have the same density in Auto-HDS (last row in Table 1), while OPTICS and DBSCAN are shown as being suitable for indexed low-d spatial data (their current popular usage).

Table 1: Comparison of HDS with other related methods on some key features that make it ideally suitable for large, high-dimensional biological datasets. "TC" stands for time complexity.

Method	DBSCAN	OPTICS	HMA	Gene Shaving	DHC	Auto-HDS
TC high-d	n^2 [a] to $n^2 \log(n)$	> DBSCAN	n^3	n^3	Unavail.	n^2 [b] to $n^2 \log(n)$
TC low-d	n^2	>DBSCAN	n^3	n^3	Unavail.	n^2
TC Indexed low-d	$n \log n$	>DBSCAN	n^3	NA	NA	$n \log n$
Cluster Hierarchy	No	Yes	Yes	No	Yes	Yes
Compact Hierarchy	No	No	Yes	No	Yes	Yes
Overlapping Clusters	No	No	No	Yes	No	No
Data type	low-d spatial [c]	low-d spatial	Unavail. [d]	gene-exp.	gene-exp.	high-d bio. data [e]
Dist. Func.	Euclidean [f]	Euclidean [g]	high-d, Euclidean	Sq. Euclidean [h]	time series	Pearson Distance [i]
Visualization	No	Yes	No	No	Yes	Yes
Model Selection	No	No	No	No	Yes	Yes
Auto. Cluster Selection	No	No	No	No	No	Yes
Select mxd. dens. clust. [j]	No	No	No	Yes	No	Yes

[a] For $n_{\epsilon avg} < n/\log(n)$

[b] For $n_{\epsilon avg} < n/\log(n)$

[c] As implemented by [EKSX96] using indexing; the current popular usage. A modification that works well for high-d results in the DS algorithm (Section 4.3).

[d] We have not found any large scale applications.

[e] Pearson distance enables application to a variety of biological datasets.

[f] As tested and applied popularly.

[g] Same as DBSCAN.

[h] As a consequence of using PCA.

[i] Also applicable with cosine similarity and Euclidean distance.

[j] All the selected clusters need not have the same density.

3 Preliminaries

3.1 Distance Measure

Let $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^n \subseteq \mathbb{R}^d$ be a set of data points that need to be clustered. We assume that a relevant symmetric distance measure $d_S(\mathbf{x}_i, \mathbf{x}_j)$ is defined for all pairs of points $\mathbf{x}_i$ and $\mathbf{x}_j$ in $\mathcal{X}$. One such distance measure is the Euclidean Distance. Let $\mathbf{M}_S$ represent the corresponding $n \times n$ symmetric distance matrix such that $\mathbf{M}_S(i,j) = d_S(\mathbf{x}_i, \mathbf{x}_j)$. The algorithms described in this paper only require $\mathbf{M}_S$ as input. Therefore, an explicit embedding of the data points in $\mathbb{R}^d$ is not required.

3.2 Density Estimation

The following is a property of our notion of density that is required by our density-based algorithms. Given some $r_\epsilon \in \mathbb{R} : min(\mathbf{M}_S) \leq r_\epsilon \leq max(\mathbf{M}_S)$ as an input, the density $\rho_{r_\epsilon}(\mathbf{x})$ at any given point $\mathbf{x}$ is proportional to the number of points in $\mathcal{X}$ that are within r_ϵ of $\mathbf{x}$ [2]:

$$\rho_{r_\epsilon}(\mathbf{x}) \propto |\{\mathbf{y} \in \mathcal{X} : d_S(\mathbf{y}, \mathbf{x}) \leq r_\epsilon\}| \tag{1}$$

This notion of density corresponds to a uniform sphere as the kernel [3], since the points within the sphere r_ϵ contribute equally to the density irrespective of their distance from the center of the sphere.

3.3 Hierarchical Mode Analysis

Given a dataset $\mathcal{X}$ consisting of n points in $\mathbb{R}^d$, [Wis68] uses the notion of density defined by Equation 1 and describes the Hierarchical Mode Analysis(HMA) algorithm for discovering the distinct modes corresponding to the dense regions in $\mathcal{X}$. Let us now present the HMA algorithm in detail. Since [Wis68] is not easily available, what follows below is presented exactly as in [Wis68] except with the substitution of notation used in this paper.

1. Select the density threshold as integer $n_\epsilon < n$, compute the inter-point distance matrix $\mathbf{M}_S$ and the distances $\mathbf{d}^{n_\epsilon}$ from each point to its n_ϵ^{th} nearest point.

2. Order the distances $\mathbf{d}^{n_\epsilon}$ so that the smallest is first using the array $\mathbf{a}^{n_\epsilon}$ as an index. Thus $\mathbf{a}^{n_\epsilon}$ defines the order in which the data points become dense: point $\mathbf{a}^{n_\epsilon}(1)$ has the smallest n_ϵ^{th} distance $\mathbf{d}^{n_\epsilon}(1)$ and is first to become dense when $r_\epsilon = \mathbf{d}^{n_\epsilon}(1)$, point $\mathbf{a}^{n_\epsilon}(2)$ is second at $\mathbf{d}^{n_\epsilon}(2)$, and so on.

3. Select distance thresholds r_ϵ from successive $\mathbf{d}^{n_\epsilon}$ values, initializing a new dense point at each cycle. As the second and each subsequent dense point is introduced, the method tests the new point to determine one of three possible fusion phases: either (i) the new point does not lie within r_ϵ of another dense point, in which case it initializes a new cluster mode, (ii) the point lies within r_ϵ of dense points from one cluster only, and therefore the point is directly fused to that cluster, or (iii) the point falls in the saddle region, lying within r_ϵ of dense points from separate clusters, and the clusters concerned are fused.

[2] The set of points within r_ϵ distance of $\mathbf{x}$ includes $\mathbf{x}$.

[3] Other possible kernels could be a Gaussian or a *Bregmanian ball* [GG05]. Selecting a particular kernel for Equation 1 is similar to selecting a particular smoothing function for *Parzen window* based density estimation.

4. Finally, a note must be kept of the nearest-neighbor distance r_{min} between dense points of different clusters. When r_ϵ exceeds r_{min}, the direct fusion of the two clusters separated by r_{min} is indicated.

Using the above procedure, HMA is able to find a very compact hierarchy of clusters that correspond to the actual "modes" or generating distributions. An illustrative example is as follows; if the data was generated by two closely located low (but not necessarily spherical or equal) variance Gaussian distributions A and B, a third somewhat distant Gaussian C, and a uniform background distribution, then HMA finds a compact hierarchy that consists of only five nodes; a root node (1) consisting of all the data points, two child nodes with (2) consisting of most of the points from A and B while the other, (3) consists of mostly points from C, and finally (2) further contains two child nodes, (4) consisting mostly of points from A, while (5) consists mostly of points from B. We will later see how the ability of HMA to discover such a compact hierarchy stems from its ability to treat less dense points as "don't care" points or outliers.

3.4 Pearson Distance for biological datasets

An example of a symmetric distance measure is *Pearson Distance* (d_p) [GG05] computed as $1 - p$, where p is the *Pearson Correlation*, a popular similarity measure for clustering gene-expression and other biological data [SS00, MTAea04]. It can be shown that Pearson Distance is equal to the Squared Euclidean distance between z-scored [4] points normalized by $2(d-1)$:

$$d_p(\mathbf{x}, \mathbf{y}) = \frac{\parallel z(\mathbf{x}) - z(\mathbf{y}) \parallel^2}{2(d-1)} \tag{2}$$

where z represents the z-scoring function. Pearson Correlation is popular among biologists for clustering genes since it captures correlation that is invariant to linear scaling; it is useful for a variety of high throughput biological datasets such as gene-expression, protein-expression, phylogenetic profiles, and protein mass spectroscopy, among others. We use the corresponding distance measure, Pearson Distance, on biological data presented in this paper. In particular, it is reasonable to do so with HMA (and for our Auto-HDS framework derived from HMA) for the following reasons: (1) the triangle inequality property is exploited indirectly in the graph-traversal process that connects the clusters in HMA, (2) $\sqrt{d_p}$ gives a semi-metric that is the Euclidean distance between z-scored points, and (3) the clustering using $\sqrt{d_p}$ and d_p would be identical for the density kernel used in HMA (Equation 1) since the relative ordering between points is the same for $\sqrt{d_p}$ and d_p. For the same reasons, it can be shown that 1-cosine similarity [DM01] would also be an appropriate distance measure with HMA and Auto-HDS. Although any arbitrary symmetric distance measure could be used, it is not clear if HMA or Auto-HDS would be meaningful for all such distance measures.

4 Density Shaving (DS)

For the labeled points (i.e., the dense points) from the i^{th} iteration of HMA, it can be shown that two dense points $\mathbf{x}, \mathbf{y} \in \mathcal{G}$ (where $\mathcal{G}$ is the set of dense points), belong to the same dense

[4] Normally performed between points across a dimension. Here we perform it between dimensions for each data point.

Algorithm 1 DS

Input: Distance matrix $\mathbf{M}_S$, n_ϵ, f_{shave}.
Output: Cluster labels $\{lab_i\}_{i=1}^n$ corresponding to the n data points.
Initialize: $\{lab_i\}_{i=1}^n = 0$
$n_c = \lceil n(1 - f_{shave}) \rceil$
// Sort each row of the distance matrix
$[\mathbf{M}_{rad}^{nbr}, \mathbf{M}_{idx}^{nbr}] = sortrows(\mathbf{M}_S)$
//Sort $n_\epsilon{}^{th}$ column of matrix $\mathbf{M}_{rad}^{nbr}$
$[\mathbf{radx}^{n_\epsilon}, \mathbf{idx}^{n_\epsilon}] = sort(\mathbf{M}_{rad}^{nbr}(\cdot, n_\epsilon))$
// Recover the r_ϵ threshold
$r_\epsilon = radx^{n_\epsilon}(n_c)$
// Recover the n_c densest points
$\mathcal{G} = \{\mathbf{x}(idx^{n_\epsilon}(i))\}_{i=1}^{n_c}$
/* Lines 17-33: For each point in $\mathcal{G}$, find other dense points
within r_ϵ distance of it and make sure they have the same
labels, if not, relabel */
for $i = 1$ to n_c **do**
  /* Find the position of the last point within
  distance r_ϵ of dense point $\mathbf{x}(idx^{n_\epsilon}(i))$. */
  $idxb = binSearch(\{\mathbf{M}_{rad}^{nbr}(idx^{n_\epsilon}(i), j)\}_{j=n_\epsilon}^{n})$
  /* Neighbors of $\mathbf{x}(idx^{n_\epsilon}(i))$ are the $idxb$ closest points, all
  within r_ϵ distance. */
  $\mathcal{X}_{nbrs} = \mathbf{M}_{idx}^{nbr}(idx^{n_\epsilon}(i), l)_{l=1}^{idxb}$
  // save the neighbors
  // Identify neighbors that are dense points
  $\mathcal{X}_{dnbrs} = \mathcal{X}_{nbrs} \cap \mathcal{G}$
  // Recover their labels that are not 0
  $L_{dnbrs} = unique(lab(\mathcal{X}_{dnbrs}))/\{0\}$
  // Relabel all points that share this label to label i
  $\forall y \in \mathbf{lab}$ if $\exists y \in L_{dnbrs} : y = i$
  $\mathbf{lab}(indexOf(\mathcal{X}_{dnbrs})) = i$
end for
Count clusters: $k = |unique(\mathbf{lab})|/\{0\}$
Remap the non-zero labels in **lab** to the range 1 to k.

cluster represented as $\mathcal{C}$ if $d(\mathbf{x}, \mathbf{y}) < r_\epsilon$. That is,

$$\forall \mathbf{x}, \mathbf{y} \in \mathcal{G} : d(\mathbf{x}, \mathbf{y}) < r_\epsilon \Rightarrow \mathbf{x}, \mathbf{y} \in \mathcal{C} \tag{3}$$

As a consequence of Equation 3, for any two points $\mathbf{x}_1$ and $\mathbf{x}_m \in \mathcal{G}$, if there exists a chain of points $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{m-1}, \mathbf{x}_m \in \mathcal{G}$ such that $\{d_S(\mathbf{x}_i, \mathbf{x}_{i-1}) < r_\epsilon\}_{i=2}^m$, then $\mathbf{x}_1$ and $\mathbf{x}_m$ also belong to the same cluster in a given iteration of HMA.

This leads to an algorithm that can compute the cluster labels in the i^{th} iteration of HMA directly without the iterative process required in HMA. This algorithm is called *Density Shaving* (DS), and a pseudocode for DS is presented in (Algorithm 1). DS essentially takes two parameters as inputs: (1) f_{shave}, the fraction of least dense points to *shave* or exclude from consideration, and (2) n_ϵ, the number of points that must be within a distance r_ϵ of a given point x_i in order for x_i to be considered dense. Note that r_ϵ is not needed as a parameter. Instead, given f_{shave} and n_ϵ, the DS algorithm computes the corresponding r_ϵ using the same

approach as HMA using $r_\epsilon = \mathbf{d}^{n_\epsilon}(i)$, where $i = \lceil n(1 - f_{shave}) \rceil$. DS then applies a graph traversal process to discover the clusters composed of the dense points, where, as stated in Equation 3, two dense points are in the same cluster if the distance between them is less than r_ϵ. The output of the algorithm consists of k clusters labeled 1 to k formed by the set $\mathcal{G}$ of n_c densest points and a "don't care" set $\mathcal{O}$ containing the remaining points which are labeled 0.

The $indexof(\mathbf{x})$ is a special function used in Algorithm 1 (and elsewhere in this paper) that returns the original integer index of a given point $\mathbf{x} \in \mathcal{X}$ while $indexOf(\mathcal{Y})$, where $\mathcal{Y} \subseteq \mathcal{X}$, returns a sorted vector of indexes corresponding to points in $\mathcal{Y}$. We use this function mainly to make the pseudocode for the algorithms compact. We have provided C++ style inline comments in Algorithm 1 that explains the algorithm in detail. For clarity, note that at Line 6 of Algorithm 1, we sort each row of the distance matrix $\mathbf{M}_S$ by increasing distance, to get an ordered list of neighbors of each point as matrix $\mathbf{M}^{nbr}_{idx}$, and their corresponding distances as matrix $\mathbf{M}^{nbr}_{rad}$. For each point in $\mathcal{G}$, the binary search (between n_ϵ and n) at Line 19 allows efficient recovery of the set of all the dense neighbors $\mathcal{X}_{nbrs}$ that are within a distance of r_ϵ of the point.

4.1 Properties of DS

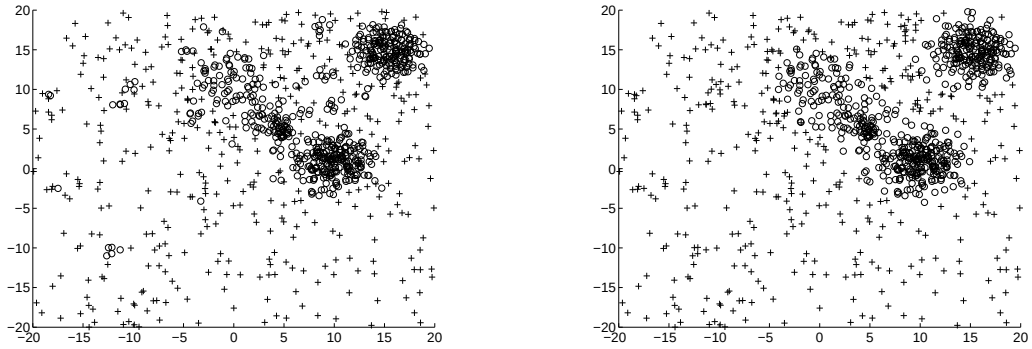


Figure 1: The effect on clustering when n_ϵ is changed from 5 (left) to 50 (right), showing the robustness of DS with respect to parameter n_ϵ. "o" represent dense points and "+" are outliers. Increasing n_ϵ by small amounts results in a smoothing effect on the density estimation that prunes the smallest dense regions and grows the larger ones, thus preserving the large-scale structures in the clustering. Data: Sim-2 (see Section 9.1).

DS requires only n_ϵ and f_{shave} as inputs, and, just like HMA, estimates an r_ϵ corresponding to f_{shave} automatically. This could be important for high-dimensional biological data sets when using Pearson Distance (which scales between 0 and 2), since there would be no intuitive value of r_ϵ that can be computed. For example, in our experiments on very high-dimensional data, we found that the difference in the value of r_ϵ when clustering 10 % less points is usually small (e.g., $\sim$0.01 for clustering 500 vs. 550 points for Gasch genes, Table 2), and does not follow volumetric intuitions of a uniform distribution. This makes the parameters n_ϵ and f_{shave} preferable over n_ϵ and r_ϵ. In addition, for a constant f_{shave}, n_ϵ acts as a *smoothing parameter* in DS (Figure 1). While DS can be applied as a clustering algorithm, we mainly use DS to construct the HDS algorithm described in the next section.

4.2 Time Complexity of DS

It is possible to implement each iteration of the graph traversal in only $O(n)$ steps [5]. This results in a graph traversal with time complexity $O(n^2)$ and a time complexity of $O(n^2 \log n)$ for Algorithm 1. Furthermore, by using heap sort to find the neighbors within r_ϵ of dense points, a faster implementation of DS with a time complexity of $O(max(n^2, nn_{avg\epsilon} \log(n)))$ is possible [6], where $n_{avg\epsilon}$ is the average neighborhood size within distance r_ϵ of the dense points. Note that $n_{avg\epsilon} \geq n_\epsilon$, and is usually small for small n_ϵ.

4.3 Connection between DBSCAN, DS and HMA

The i^{th} of the n iterations of HMA corresponds to an $r_\epsilon = \mathbf{d}^{n_\epsilon}(i)$, and results in three types of points: (a) a set of dense points that have cluster labels, (b) a set of non-dense points that are within the current r_ϵ of at least one dense point, and (c) a set of non-dense points that are neither of the two. It can be shown that for the particular values of $r_\epsilon = \mathbf{d}^{n_\epsilon}(i)$ and n_ϵ at the i^{th} level, if the set of non-dense points within r_ϵ distance of a dense point are clustered with their nearest dense point (in a specific order to break ties), you get a clustering that is identical to that of the DBSCAN algorithm, using the corresponding r_ϵ and n_ϵ used in HMA as inputs. As mentioned earlier in Section 2, this alternative form of clustering non-dense points was noted in [Wis68]. Since DS computes the i^{th} iteration of HMA directly, one could consider DS to be the HMA equivalent of DBSCAN. That is, DS obtains the same clustering as DBSCAN for points of type (a) but does not cluster points of type (b), whereas DBSCAN clusters points of both types (a) and (b).

5 Hierarchical DS (HDS)

In this Section we describe a framework called *Hierarchical Density Shaving* (HDS) that finds a good approximation of the HMA hierarchy in two steps. We first use the DS algorithm to compute a subset of the HMA iterations, and then we perform a relabeling of the DS clusterings to create a noise-tolerant version of the HMA hierarchy. We start by observing the following property of HMA:

Proposition 5.1. *The cluster labels in each of the n iterations of the HMA hierarchy can be computed independently of one another.*

This proposition follows as a consequence of the DS algorithm that can compute the i^{th} iteration of HMA directly without using the iterative procedure originally proposed by [Wis68]. Computing all the n levels of HMA using either the original algorithm (Section 3.3) or DS results in a time complexity of $O(n^3)$. However, since the HMA iteration cluster labels are nested (which follows from HMA directly), and because of Proposition 5.1, any subset of DS executions corresponding to some subset of the n HMA iteration clusterings also forms a hierarchical clustering.

Therefore, an obvious way to produce such a hierarchy faster would be to compute DS clusterings only for $n_c = n, n - r, n - 2r$ and so on, where r is some small integer greater than

[5] The details of such an implementation and the corresponding time complexity proof are being omitted here for brevity.

[6] Section 8 describes an implementation based on this approach.

1, thus skipping some of the HMA iterations and reducing the total runtime by a factor of r. However, we propose an alternate scheme that works as follows: at each iteration, a constant fraction of dense points are removed ("shaved") from the set of dense points from the previous iteration. This process has the ability to perform finer shavings as clusters get smaller, thus preserving the ability to discover small-scale but highly dense structures. This process can also be thought of as an *exponential shaving*, where we specify a "shaving rate" $0 < r_{shave} \leq 1$. Then, the value $n_{cr} \in \mathbb{R}$ obtained by applying the exponential shaving t times on n is given by:

$$n_{cr} = n \times (1 - r_{shave})^t \tag{4}$$

HDS level: The *level* of an iteration in HDS that clusters n_c points is computed using a function $f_{level}(n_c)$ defined as follows:

$$f_{level}(n_c) = \frac{\log(n_c) - \log(n)}{\log(1 - r_{shave})} \tag{5}$$

f_{level} measures the number of shavings of fraction r_{shave} required to obtain n_c, starting from n, and can be derived from Equation 4 by substituting $t = f_{level}$ and $n_c = n_{cr}$. Note that unlike t, f_{level} is a real number and takes into account the fact that n_c is an integer.

HDS iterations: By substituting $n_c = 1$ in Equation 5, we can show that, for a given r_{shave} and n, at most $j_{max} = \lceil -\frac{\log(n)}{\log(1-r_{shave})} \rceil$ iterations are needed. In practice, for small r_{shave}, since many such iterations result in the same n_c, the number of distinct HDS iterations is upper-bounded by j_{max}. These distinct values of n_c are used for the iterations of HDS, and can be obtained as a sorted list of positive integers (from Equation 4) as follows:

$$\mathbf{n}_{nclist} = sortd(unique(\{\lceil n \times (1 - r_{shave})^t \rceil\}_{t=0}^{j_{max}})) \tag{6}$$

where *sortd* represents a sort by decreasing value. Let $n_{iter} = |\mathbf{n}_{nclist}|$ represent the number of iterations of HDS, with the j^{th} entry of $\mathbf{n}_{nclist}$ now corresponds to the j^{th} iteration of HDS. The level of the j^{th} iteration of HDS can be obtained using Equation 6 and 5:

$$level(j) = \frac{\log(n_{nclist}(j)) - \log(n)}{\log(1 - r_{shave})} \tag{7}$$

Since a smaller n_c corresponds to a larger level (Equation 5), the iterations represented by n_{nclist} are also ordered by increasing HDS level. For the first iteration, $\mathbf{n}_{nclist}(1) = n$ and the corresponding level is 0, and in general, the level in the j^{th} iteration roughly corresponds to $(j - 1)$.

HDS vs. HMA iterations: Although each distinct iteration of HDS could be computed in any order because of Proposition 5.1, they are organized as a shaving order listed by Equation 6. This ordering is required for the faster algorithm known as Recursive HDS introduced later in Section 5.3. Note that the first iteration of HDS maps to the last iteration of HMA; HMA iterations are "top-down" whereas HDS iterations are defined bottom up. Also note that the HDS iterations have a level defined on them given by Equation 7 that corresponds to the exponential shaving and not to the iterations of HMA; this difference is critical for the automatic cluster selection described in Section 6.

5.1 A Simple HDS Algorithm

We showed earlier that HDS consists of less than $\lceil -\frac{\log(n)}{\log(1-r_{shave})} \rceil$ iterations, which is $O(\log(n))$. Therefore, repeated calls to Algorithm 1 to compute each iteration of HDS results in a time complexity of $O(n^2(\log(n))^2)$. A faster alternative is to modify Algorithm 1 as follows:

Algorithm C: (1) Sort the distance matrix only once, and (2) Recompute the r_ϵ for each n_c in $\mathbf{n}_{nclist}$, the corresponding neighbor memberships, and the graph traversal. It can be shown that the graph traversal in each iteration of DS has a time complexity of $O(nn_c)$. In HDS, since n_c decays exponentially with the shaving iterations, it can be shown that the total time complexity of the graph traversal for all values of n_c (Equation 6) is only $O(n^2)$, resulting in a time complexity that is the same as that of DS.

5.2 Extracting a smoothed HMA hierarchy

The hierarchy produced by HMA involves top-down "growing" clusters; the algorithm starts with the densest point and then repeatedly merges an additional point in each iteration, either (1) starting a new cluster, (2) merging with an existing cluster, or (3) merging with two or more existing clusters. Although Algorithm C produces a subset of the full HMA iterations (represented by matrix $\mathbf{L}$), the labels are not based on labels in previous iterations. Hence, there is no correspondence between the different iterations of Algorithm C. In order to produce labels that also correspond to the HMA labels on a hierarchical basis, a re-labeling of $\mathbf{L}$ needs to be performed. Such a relabeling can also be viewed as a "compaction" of the hierarchy generated by Algorithm C.

Additionally, we introduce the notion of a *particle*, a cluster that emerges from a parent cluster but is considered spurious due to its small size. Particles are ignored when compacting $\mathbf{L}$. We define a *particle* as any dense cluster of size less than n_{part} points (note that we can include particles simply by letting $n_{part} = 0$). Starting from the left most column, relabeling of $\mathbf{L}$ proceeds as follows: (1) find the unique cluster IDs at iteration $j-1$. (2) Repeat the following for each of the clusters found in Step 1: (2.1) If all points belonging to a cluster in iteration $j-1$ are either: (a) clustered in the same cluster in iteration j, (b) are assigned to the don't care set $\mathcal{O}$, or (c) are assigned to a cluster that is a particle, then we assign the child cluster at iteration j the label of the parent cluster at iteration $j-1$. That is, one can view the cluster on iteration j as a continuation of the corresponding cluster on iteration $j-1$, barring those points which are now part of $\mathcal{O}$ or a particle. If the condition in (2.1) is not satisfied, then (2.2) a cluster has split into two or more child clusters. Each of these child clusters is assigned a new cluster ID. The relabeled label matrix $\mathbf{L}_{HDS}$ output by this procedure represents a smoothed HMA hierarchy, which we refer to as the *HDS hierarchy*.

5.3 A faster "Recursive-Iterative" Algorithm

A property of the HDS hierarchy is that two distinct clusters from an earlier shaving iteration stay distinct as we keep shaving, and eventually disappear (i.e., all their points become "don't care"). This makes it possible to apply Algorithm 1 recursively to each of the clusters found in $\mathbf{L}(\cdot, j)$ to obtain the cluster labels in $\mathbf{L}(\cdot, j+1)$. However, using recursive function calls to implement such an algorithm would require that relevant data be saved on a stack for each such call. This operation is extremely memory intensive and can cause stack overflow. Fortunately, a more efficient *iterative* implementation of this conceptually recursive algorithm

Algorithm 2 Recursive HDS

Input: Distance matrix $\mathbf{M}_S$, n_ϵ, r_{shave}
Output: $n \times n_{iter}$ Cluster hierarchy matrix $\mathbf{L}$.
Initialize all values in $\mathbf{L}$ to 0.
$[\mathbf{M}_{rad}^{nbr}, \mathbf{M}_{idx}^{nbr}] = sortrows(\mathbf{M}_S)$
$[\mathbf{idx}^{n_\epsilon}, \mathbf{radx}^{n_\epsilon}] = sort(\mathbf{M}_{rad}^{nbr}(\cdot, n_\epsilon))$
Compute $\mathbf{n}_{nclist}$ using Equation 6.
$\mathbf{r}_{\epsilon list} = \mathbf{radx}^{n_\epsilon}(\mathbf{n}_{nclist})$
$n_c = \mathbf{n}_{nclist}(1)$
$r_\epsilon = \mathbf{r}_{\epsilon list}(1)$.
Initialize: $\mathbf{L}(\cdot, 1) = \mathbf{lab}$, $\mathcal{G}_{all} = \mathcal{G}$, and $\mathcal{N}$ using Algo. 1.
for $j = 2$ to n_{iter} **do**
  $\mathcal{G}_{last} = \mathcal{G}_{all}$
  $\mathcal{N}_{last}^o = \mathcal{N}/\mathcal{G}_{all}$ // non-dense nbrs.
  Compute $\mathcal{L}_{last}^{row}$ as unique rows of matrix $\{\mathbf{L}(;i)\}_{l=1}^{j-1}$
  $k_{last} = |\mathcal{L}_{last}^{row}|$ // no. of dense clust. found in iteration $j-1$
  $\mathcal{N} = \varnothing$, $\mathcal{G}_{all} = \varnothing$
  for $m = 1$ to k_{last} **do**
    $\mathcal{C}_m =$ // members of the dense cluster
    $\{\forall \mathbf{x} \in \mathcal{G}_{last} : \{\mathbf{L}(indexOf(\mathbf{x}), i)\}_{i=1}^{j-1} = \mathcal{L}_{last}^{row}(m)\}$
    // combine cluster pts with non-dense nbrs. from
    // iteration j-1 to create a working set of points
    $\mathcal{X}_{work} = \mathcal{C}_m \cup \mathcal{N}_{last}^o$
    $\mathcal{G}_{all} = \mathcal{G}_{all} \cup \mathcal{C}_m$ // keep track of dense points found
    //recover the distance matrix subset for working set
    $M_s^{work} = M_s(indexOf(\mathcal{X}_{work}), indexOf(\mathcal{X}_{work}))$
    // Recover sorted rows and corresponding indices
    $S_{rad}^{nbr} = M_{rad}^{nbr}(indexOf(\mathcal{X}_{work}), indexOf(\mathcal{X}_{work}))$
    $S_{idx}^{nbr} = M_{idx}^{nbr}(indexOf(\mathcal{X}_{work}), indexOf(\mathcal{X}_{work}))$
    $[\mathbf{idx}^{n_\epsilon}, \mathbf{radx}^{n_\epsilon}] = sort(\mathbf{S}_{rad}^{nbr}(\cdot, n_\epsilon))$
    $r_\epsilon = \mathbf{r}_{\epsilon list}(j)$
    Compute n_c as index pos. in $\mathbf{radx}^{n_\epsilon}$ with value r_ϵ
    Find cluster labels $\mathbf{L}(indexOf(\mathcal{X}_{work}), j)$, nbr. set
    $\mathcal{N}_{work}$ and cluster set $\mathcal{G}_{work}$ using Ln. 12-20, Algo. 1.
    $\mathcal{N} = \mathcal{N} \cup \mathcal{N}_{work}$
  end for
end for

is possible and is described in Algorithm 2. Recursive HDS works by applying DS at iteration $j+1$ to each of the clusters found at level j. An extra step required is the recomputation of the n_c used for each of the clusters. Although Recursive HDS has the same time complexity as Algorithm C (Section 5.1), it usually runs much faster in practice. An HDS iteration involving dense, well-separated clusters can run up to k times faster using Recursive HDS as compared to Algorithm C, resulting in overall speed gains that are significant for practical applications.

5.4 Properties of HDS

Besides a faster time complexity, the exponential shaving used in HDS has several advantages over a linear shaving. For one, smaller clusters of higher density get shaved more finely leading

to a good approximation of the HMA hierarchy. The exponential shaving also has certain other properties that leads to an unsupervised cluster selection that we describe in Section 6.

For $n_{part} = 0$, the relabeled HDS hierarchy is identical to a subset of the HMA hierarchy. A larger n_{part} acts as a smoothing parameter, somewhat similar to the effect produced by a larger n_ϵ. However, there is a subtle but important difference between the two; while n_ϵ "smooths" the notion of density, resulting in less significant dense regions getting ignored (Figure 1), n_{part} has a smoothing effect on the HDS hierarchy generation, preventing insignificant child clusters from forming from parent clusters. The use of n_{part} in HDS is also similar to *runt pruning* used in [Stu03].

Since the hierarchy compaction process is extremely fast ($O(n \log n)$), n_{part} can be selected interactively [7] by the user to smooth clustering. In Section 7, we describe an illustrative example of how this can be achieved in conjunction with the Visualization framework. These properties make HDS a good but much faster approximation of HMA, and makes interactive HMA-type analysis for finding small dense regions practical on much larger datasets.

6 Model Selection

We now describe a method for ranking the clusters in the HDS hierarchy using a novel stability criteria, and a simple algorithm for selecting the clusters based on the ranking.

6.1 Ranking Clusters

The compact hierarchy generated using the process described in 5.2 makes it easier to select clusters. For many applications, such a hierarchy by itself may be the end goal. However, in the absence of any supervision, a notion of cluster quality that we call *stability* can be used to rank clusters, where a higher stability gives a higher rank to a cluster. We define stability as the number of levels a cluster exists in the HDS hierarchy. For a given cluster $\mathcal{C}$ [8], the stability can be calculated as:

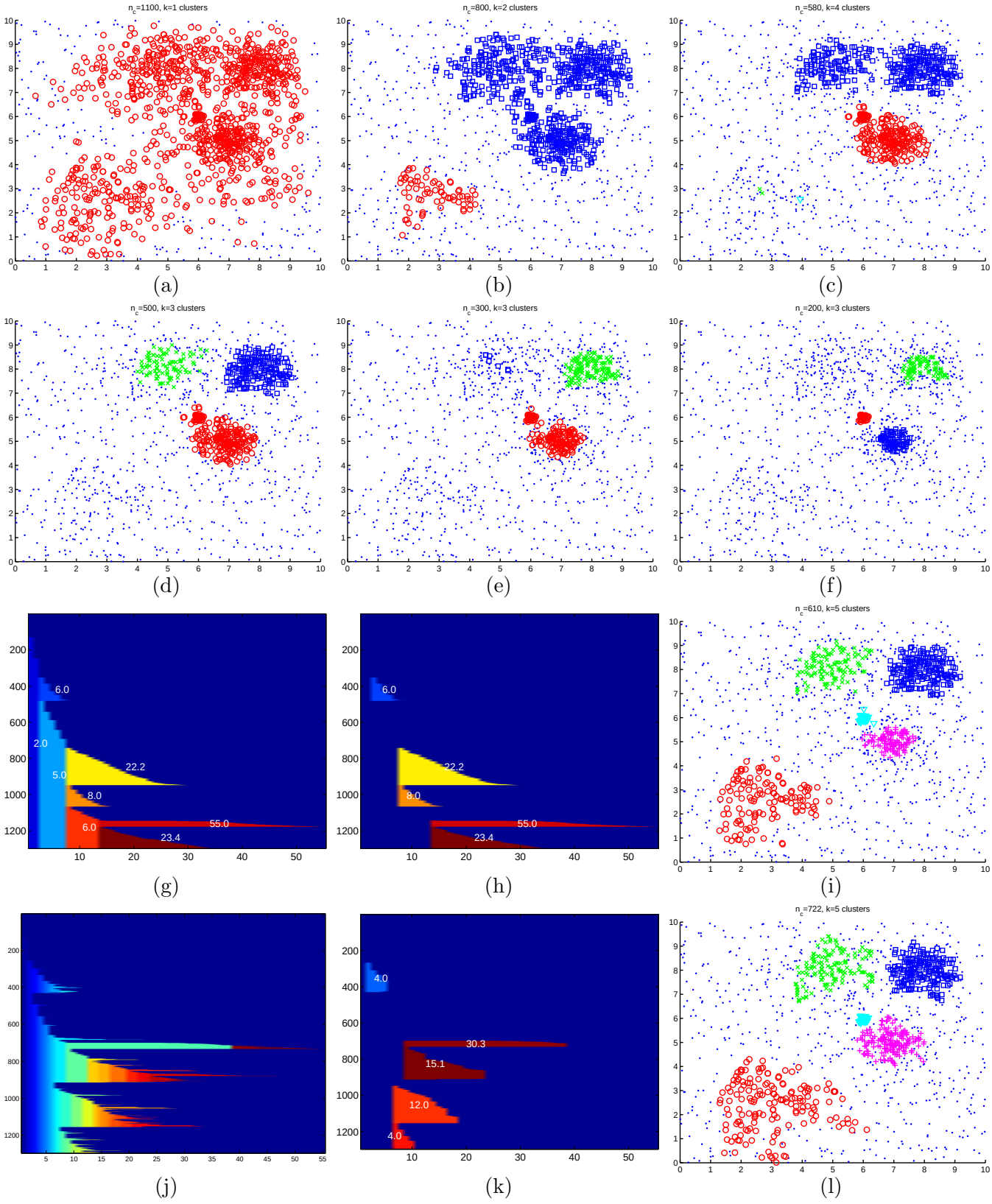
$$Stab(\mathcal{C}) = \frac{\log(n_{clist}(j_e)) - \log(n_{clist}(j_s - 1))}{\log(1 - r_{shave})} \tag{8}$$

where j_s is the iteration of HDS where $\mathcal{C}$ first appears and j_e is the last iteration where $\mathcal{C}$ survives (after which it either splits into child clusters or disappears completely). This notion of cluster stability has the following properties that makes ranking of clusters of various sizes and densities using stability robust and meaningful: 1) When ordering clusters by cluster stability, the ordering is independent of the shaving rate, and 2) **Scale Invariance of Stability**: Ordering clusters by stability results in the same ordering as when ordering clusters by the fraction of data shaved from the entire dataset between the first iteration where $\mathcal{C}$ appears and the last iteration before $\mathcal{C}$ disappears.

The first property follows from the fact that the denominator in equation 8 is a constant for all clusters $\mathcal{C}$. The second property is due to the fact that since the fraction of data shaved between two levels is constant, the fraction of points shaved between $Stab(\mathcal{C})$ levels is also

[7] In contrast, modifying n_ϵ for smoothing is too slow to be interactive for large problems.

[8] Clusters can only be found at iteration 2 or greater.



(j)

Figure 2: (a) to (f): Effect of DS applied with varying n_c (f_{shave}) for $n_\epsilon = 20$, resulting in a hierarchy of clusters. For $n_\epsilon = 20, n_{part} = 5$, HDS visualization after cluster identification (g), and cluster selection (h). Unlike DS, Auto-HDS can discover clusters of different densities (i). The fourth row shows Auto-HDS results for $n_\epsilon = 7$: (j) shows the degradation of hierarchy compaction with $n_{part} = 0$, (k) shows hierarchy compaction and cluster selection using $n_{part} = 30$ that result in clusters (l) very similar to those in (i). An r_{shave} of 0.1 was used for HDS. Dataset: Sim-2 (see Section 9.1).

constant and is given by the equation $f^{\mathcal{C}}_{shave} = (1 - r_{shave})^{Stab(\mathcal{C})}$, which can be derived from equation 8.

Note that if one were to use a linear shaving rate, i.e. shaving off the same number of points, the second property would no longer hold true. That is, stability is meaningful only because of the exponential shaving rate, i.e., shaving off the same fraction of points in each iteration.

To summarize, we can now discover all the significant clusters in the relabeled hierarchy and can compare *all* clusters with each other (e.g., clusters with different number of member points and densities, and parent and children clusters).

6.2 Selecting Clusters

Picking the most stable clusters proceeds iteratively as follows. First, make a list of all clusters eligible for selection (i.e., all clusters that are not particles). Second, pick the eligible cluster with the largest stability. Third, mark all parent and child clusters of the selected cluster as ineligible. Repeat steps two and three until there are no more eligible clusters. Note that the cluster members of the selected clusters are all the points assigned to that cluster on the first level the cluster appears in the HDS hierarchy. Using such an assignment, we are able to select clusters that satisfy different notions of density, which is not possible using DS. This also impacts the points that are assigned to the "don't care" set as well; they now correspond to all the points in $\mathcal{X}$ not belonging to any of the selected clusters. An illustrative example highlighting the difference between HDS selected clusters and DS is shown in Figure 2, where (a) through (f) show Density Shaving applied to the Sim-2 data when clustering from 1,100 to only 200 points. Notice the difference in clustering between Figure 2, (c) and (i) where approximately the same number of points were clustered: in 2(c), DS is used to cluster 580 points, while in 2(i), HDS with model selection is used to cluster 610 points.

7 Visualization with HDS

x_i	*Cluster IDs*
$\mathbf{x_1}$	1 1 2 2 0 0 0 0
$\mathbf{x_2}$	1 0 0 0 0 0 0 0
$\mathbf{x_3}$	1 1 3 4 0 0 0 0
$\mathbf{x_4}$	1 0 0 0 0 0 0 0
$\mathbf{x_5}$	1 1 0 0 0 0 0 0
$\mathbf{x_6}$	1 1 3 5 5 0 0 0
$\mathbf{x_7}$	1 1 3 0 0 0 0 0
$\mathbf{x_8}$	1 1 3 4 4 4 0 0
$\mathbf{x_9}$	1 1 3 4 4 4 4 0
$\mathbf{x_{10}}$	1 1 0 0 0 0 0 0

(a)

x_i	*Cluster IDs*
$\mathbf{x_2}$	1 0 0 0 0 0 0 0
$\mathbf{x_4}$	1 0 0 0 0 0 0 0
$\mathbf{x_5}$	1 1 0 0 0 0 0 0
$\mathbf{x_{10}}$	1 1 0 0 0 0 0 0
$\mathbf{x_1}$	1 1 2 2 0 0 0 0
$\mathbf{x_7}$	1 1 3 0 0 0 0 0
$\mathbf{x_3}$	1 1 3 4 0 0 0 0
$\mathbf{x_8}$	1 1 3 4 4 4 0 0
$\mathbf{x_9}$	1 1 3 4 4 4 4 0
$\mathbf{x_6}$	1 1 3 5 5 0 0 0

(b)

Figure 3: Example of the dictionary sort on $\mathbf{L}_{HDS}$ for $n = 10$ and $n_{iter} = 8$, $n_{part} = 0$. (a) shows the unsorted label matrix $\mathbf{L}_{HDS}$, while (b) shows the result of the sorting.

Each row of the $n \times n_{iter}$ matrix $\mathbf{L}_{HDS}$ representing the HDS hierarchy contains the cluster label of each point in all the HDS iterations. We now sort the rows of $\mathbf{L}_{HDS}$ using a dictionary sort, such that we give higher precedence to labels with smaller column indices. An example of the effect of this sorting is shown on a toy example in Figure 3.

A simple yet powerful visualization of the HDS hierarchy can be achieved by plotting this sorted matrix, assigning separate colors to each distinct positive value in the matrix, and a background color to the values that are 0. Figure 2(g) shows such a visualization for the 2-D Sim-2 data, while Figure 6(a) shows the same for the 6,151 dimensional Gasch data. The corresponding plots where the clusters have been selected using the model selection described in Section 6 is shown in Figure 2(h) and Figure 6(d) respectively, wherein, the visualization also labels the selected clusters with their actual stability.

We call the combination of HDS, model selection, cluster selection, and visualization the *Auto-HDS* framework.

Note that just as in HMA, for a range of iterations of HDS (for example Figure 2, (d) vs. (e)), the number of clusters often does not change, and each of the clusters at iteration $j-1$ simply loses some points to the "don't care" cluster at iteration j. However, since HDS uses exponential shaving, the iterations of HDS are on a log-scale (x-axis) as compared to HMA and therefore show the smaller, denser clusters well. Furthermore, the length of the clusters approximately corresponds to the stability of the clusters, while the relative separation of two points or clusters along the y-axis is proportional to their distance in the original space, since dense points in the same clusters are closer in the dictionary sort; the process of labeling used by HDS results in a novel and amazingly simple projection of high-d data density and clusters onto a 2-d space that is at the same time tightly integrated with the clustering methodology.

The Auto-HDS visualization also enables easy visual verification of the cluster selection process. The cluster selection process selects clusters from the first level that they occur, and this can be seen in the toy example in Figure 3(b), where the first level of cluster 4 occurs at the fourth column from left, and thus the member points of cluster 4 are $\mathbf{x}_3$, $\mathbf{x}_8$ and $\mathbf{x}_9$. Another example of the cluster selection process with visualization is shown in Figure 2, (g) and (h). Figure 2(g) shows the relabeled and sorted HDS hierarchy on the Sim-2 data, while 2(h) shows the corresponding clusters selected automatically, along with their stability values. Figure 2(i) shows the clusters corresponding to 2(h) in the original 2-d Euclidean space. It can be seen that Auto-HDS finds five clusters and a compact hierarchy with only eight nodes, and that the results match remarkably well with the actual dense regions and their relative positions in the Sim-2 data.

It is important to note why the hierarchy produced by Auto-HDS (e.g. in the Sim-2 example above) is extremely compact, a key property that distinguishes it from traditional hierarchical clustering methods. The Auto-HDS hierarchy at any level corresponds to only the dense subset of the data; that is at any given level, the least dense points are assigned to the "don't care" set. Therefore, the number of clusters does not grow rapidly as one moves up or down the Auto-HDS levels; as we move up the levels, new clusters only appear when a cluster splits, and old, less dense clusters disappear. In contrast, traditional hierarchical methods, such as Agglomerative clustering, cluster all of the data at all levels, and in the presence of noise (a common characteristic of real-life data) they end up discovering numerous spurious clusters.

Auto-HDS visualization can also aid in choosing the two HDS smoothing parameters n_ϵ and n_{part}. It is possible to prevent small, insignificant clusters from being found by selecting either

a larger value of n_ϵ or n_{part}. If the user changes n_{part}, Auto-HDS clustering can be updated fairly quickly (usually interactively, within seconds), since the hierarchy compaction process is very fast. In contrast, changing n_ϵ requires the regeneration of the HDS clustering. It is therefore possible to chose a very small n_ϵ to start with, obtain a noisy clustering, and then slowly increase n_{part} until clusters obtained stop changing substantially. Once the clustering appears stable and satisfactory, the user can then either use the resultant clustering as the final output, or use the optimum value of n_{part} as a rough estimate to select a larger n_ϵ and run the HDS clustering from scratch. An example of using this approach for finding good clustering on the Sim-2 data is shown in Figure 2, (g) through (l)), where a small n_ϵ was first used to generate a "noisy" hierarchy (Figure 2(j)), and was subsequently smoothed using a larger n_{part} (Figure 2(k)). A larger n_ϵ was later found to be sufficient for obtaining a good hierarchy (Figure 2(h)). The hierarchy of clusters found using the two alternatives, i.e. a larger n_ϵ vs. a larger n_{part}, shows very similar clustering results (Figure 2, (i) vs. (l)), organized in a very similar topology (Figure 2, (h) vs. (k)).

To summarize, the visualization provides a powerful, compact, informative hierarchy and a spatially relevant 2-D projection of a high-dimensional density distribution. While many visualization tools are built on top of clustering results, the Auto-HDS visualization directly corresponds to the clustering methodology. The visual feedback provided also allows the user to go back, if desired, and adjust the smoothing parameters n_{part} and n_ϵ. Typically, however, the results are quite stable over a range of parameter values.

8 Gene DIVER: A Scalable Java implementation

The Gene DIVER product website is at:
http://www.ideal.ece.utexas.edu/∼gunjan/genediver

The results presented in this paper were mostly produced using a Recursive HDS implementation in Matlab, which is ideal for rapid prototyping and testing. By exploiting some of the advantages provided by a heap based implementation (briefly mentioned in Section 4), we also implemented a highly scalable and sophisticated version of Auto-HDS in Java called *Gene DIVER* (Gene Density Interactive Visual Explorer). This product has several key features that make Auto-HDS scalable on modest computers for reasonably large clustering problems. Gene DIVER provides a sophisticated SWING-based user-interface that not only makes it usable by non-programmers, but also provides the ability to perform interactive clustering. Furthermore, the Java implementation is an open-source implementation makes the application immediately runnable on most platforms, and does not bind the user to a specific commercial platform such as Matlab or SAS. Some of the salient features of the Gene DIVER 1.0 product are as follows:

1. Computes and stores distance matrices on secondary storage instead of loading the $O(n^2)$ distance matrix in memory. By loading and sorting only one data row at a time in memory, the memory usage by Gene DIVER is only $O(nn_{avg\epsilon})$, where $n_{avg\epsilon}$ is the number of neighbors within the r_ϵ neighborhood, averaged over all the clustered data points. This is usually much smaller than $O(n^2)$. For example, when clustering all the genes in the Lee dataset (Table 2) across all the experiments, Gene DIVER requires only 40 MB of memory. In contrast, a single copy of the distance matrix of the Lee dataset would correspond to over 120 MB in Matlab. A corresponding implementation of HDS that

computes and performs traditional sorting on such a distance matrix also requires many intermediate copies of such a matrix, along with Matlab itself to reside in memory; this ends up requiring a machine with approximately 1 GB of RAM. The improvements in both memory and speed efficiency usage are so large that the Java implementation of Auto-HDS is now suitable for much larger clustering problems.

2. Heap Sort is not available in JDK 1.5 SDK, the current version of Java from Sun Microsystems (java.sun.com). We implemented a custom Heap Sort API that uses sorting of referenced integer indices, rather than of Java objects made of an {index,value} pair. This results in a much faster implementation of heap sort.

3. A custom serialization of the distance matrices and the row heaps with large buffering for reading/writing to secondary storage makes the $O(nn_{avg\epsilon})$ memory implementation practical, by keeping the disk read/write overhead very small.

4. Partially sorted heaps are also saved to secondary storage for future clustering. When the user tries to cluster a dataset which she/he had clustered before, Gene DIVER automatically reuses the heaps created and stored earlier to estimate the neighbors within the specified radius r_ϵ, and only incrementally sorts the heaps if needed. Gene DIVER performs checks for all possible combinations of changes in inputs by the user and maximizes the degree of reuse of the older heaps, thus greatly reducing the amount of additional processing needed. This allows for much faster and almost interactive clustering for reasonably large problems.

5. Ability to skip clustering of some of the least dense data. Gene DIVER allows the user to specify a fraction of the least dense data to skip clustering upon, and only to build the HDS hierarchy beyond that. This allows the clustering in practice to be considerably faster.

6. Allows user to choose between three important distance measures: (1) Squared Euclidean, for spatial data- the classical domain of density based clustering algorithms, (2) Pearson Distance (Section 3.4), a useful measure for clustering many types of biological data, and (3) 1-Cosine Similarity, which is useful for text clustering- an application area that we have not yet tested, but which could be a useful domain to explore with Gene DIVER.

7. Ability to specify either a precomputed distance matrix, or a vector space as input data. The ability to specify a precomputed distance matrix in a file makes it possible for users to incorporate their own custom distance measure into Gene DIVER. This could be important for applications in bioinformatics such as clustering of phylogenetic data where the ideal distance measure may not be Pearson Distance, or in situations where the distance between genes is computed using an ensemble of information over multiple types of datasets [LDAM04]. In such cases, the distance between two genes could be precomputed using the external model; the resulting distance matrix could then be passed as an input to Gene DIVER.

8. A Java Swing based user interface allows users to select data and other parameters and options for clustering. Furthermore, the novel visualization and model selection of the HDS cluster hierarchy is also a part of the Java Swing user-interface, and allows the

user to select and browse the resulting clusters. For some of the clustering parameters such as the n_{part} (runt) parameter, Gene DIVER can re-cluster the results in seconds; the SWING interface thus enables the user to perform interactive clustering, a desirable property for biologists. The user interface currently also allows one to specify the class label data as input, and then shows the users the class label distribution statistics for the generated clusters.

9. For more Java savvy users who want to incorporate Auto-HDS in their own applications/programs, Gene DIVER also provides an equally efficient and scalable command-line version of Auto-HDS that does not use the user interface.

We are currently working on Gene DIVER version 2.0 which will have several new features, such as the ability to automatically label selected gene clusters using known functional linkages between genes, and the ability to modify the automatically selected clusters and cluster boundaries in order to discover overlapping gene clusters. These modifications will make Gene DIVER even more powerful, especially for discovering new functional relationships between genes using a variety of biological datasets. Another enhancement will be the ability to zoom in and out of the clusters for better visualization of small, dense clusters on large data sets. This could be particularly useful for large gene-expression datasets, where often the most significant gene clusters are usually very small.

9 Experimental Evaluation

Additional results are available at:
http://www.ideal.ece.utexas.edu/∼gunjan/hds/readme.html.

9.1 Datasets

Table 2: A summary of the datasets used. Mic. stands for gene-expression data from microarray experiments, Euc. stands for Euclidean distance, and D is the distance function used for clustering.

Dataset	Source	n	d	D	true k	k_A
Gasch	Mic.	173	6,151	d_p	12	11
Lee	Mic.	5,612	591	d_p	NA	9
Sim-2	Sim.	1,298	2	Euc.	5	5

We tested our framework on two real and one artificial datasets. In Table 2, which summarizes the properties of these datasets, *true* k corresponds to the number of known classes in the dataset, while k_A refers to the number of clusters automatically found by Auto-HDS. The Sim-2 dataset was generated using five 2-D Gaussians of different variances (which roughly correspond to the clusters in figure 2 (i)) and a uniform distribution. Two of the Gaussians had relatively small mass, and one of them had very low variance. This dataset is useful for verifying algorithms since the true clusters and their spatial distributions are known exactly. The Gasch

dataset [Gas00], a widely used benchmark for testing clustering algorithms on microarray data, consists of 6,151 genes of yeast *Saccharomyces cervisiae* responding to diverse environmental conditions over 173 microarray experiments. These experiments were designed to measure the response of the yeast strain over various forms of stress such as temperature shock, osmotic shock, starvation and exposure to various toxins. Each experiment was categorized into one of 12 different categories based on the experiment label. Many of the categories are closely related such as "temperature", "cold shock" and "heat shock". Furthermore, each of the 173 experiments have a description associated with them. The Lee dataset [LDAM04] consists of 591 gene-expression experiments on yeast obtained from the Stanford Microarray database [Gol03] (http://genome-www5.stanford.edu/) and also contains a *Gold* standard based on Gene Ontology(GO) annotations (http://www.geneontology.org). The Gold standard contains 121,406 pairwise links (out of a total of 15,744,466 gene pairs) between 5,612 genes in the Lee data that are known to be functionally related. The Gold standard was generated using levels[9] 6 through 10 of the Gene Ontology biological process.

9.2 Evaluation Criteria

We use the following three criteria for performing evaluations against labeled data. Note that the labels were only used for evaluations. Auto-HDS, DS, and all benchmarks were executed in a completely unsupervised setting.

1. *Adjusted Rand Index*: Adjusted Rand Index was proposed by [HA85] as a normalized version of Rand Index, and returns 1 for a perfect agreement between clusters and class labels and 0 when the clustering is as bad as random assignments. ARI can be used on the Gasch Array and the Sim-2 datasets since the true class-labels are available.

2. *p-value*: We use p-value to evaluate individual clusters of Yeast genes found on the Lee dataset. *Funspec* [10] is a popular Yeast database query interface on the Web that computes cluster p-values for individual clusters using the hypergeometric distribution, representing the probability that the intersection of a given list of genes with any given functional category occurs by random chance.

3. *Overlap Lift*: The Go annotation Gold standard provides labels for the Lee dataset in the form of a set of pairwise links between functionally related genes; one could also view these labels as an undirected graph. It is not possible to use ARI with such a set of labels. Furthermore, the p-value described above is only relevant for individual clusters. For evaluating the overall clustering quality on the Lee dataset using the Go annotation Gold standard, we can compute the statistical significance of all the clusters simultaneously using *Overlap Lift*, which we define as follows: A cluster containing w genes in one cluster creates $w(w-1)/2$ links between genes, since every point within the cluster is linked to every other point. Therefore, k clusters of size $\{w_j\}_{j=1}^{k}$ would result in a total of $l_c = \sum_{j=1}^{k} w_j(w_j-1)/2$ links. The fraction of pairs in the Gold standard that are linked f_{linked} is known (e.g., for Lee dataset $f_{linked} = 121,406/15,744,466 = 0.007711$). If we construct a null hypothesis as randomly picking l_c pairs out of $n(n-1)/2$ possible

[9] Note that the term "level" is used here only in the context of the GO annotation and should not be confused with levels of HDS.

[10] http://funspec.med.utoronto.ca/

pairs, we can expect $l_{null} = f_{linked} l_c$ pairs to be correctly linked. A good clustering should result in more correctly linked pairs than l_{null}. If l_{true} is the number of correct links observed (which will always be $\leq l_c$) in our clustering, then the Overlap Lift is computed as the ratio l_{true}/l_{null}, which represents how many more times correct links are observed as compared to random chance. A larger ratio implies better clustering.

Note that the points in the background or the "don't care" clusters were excluded from the evaluation.

9.3 Benchmark Algorithms

Most labeled evaluation measures for clustering are sensitive to the number of clusters discovered and the percentage of data clustered. To get around this problem we ensure that the benchmark algorithms use the same n_c and k as our methods by applying the following procedure that we call *MaxBall*:

1. For a given benchmark clustering algorithm, find k clusters $\{\mathcal{C}_j\}_{j=1}^{k}$, where k corresponds to the number of clusters found by DS for a particular n_c.
2. Compute cluster center $\mathbf{c}_j$ for cluster $\mathcal{C}_j$ as the mean of the cluster's member points.
3. Assign each of the n points to the cluster center among $\{\mathbf{c}_j\}_{j=1}^{k}$ that is closest [11] to it.
4. Select n_c points closest to their assigned cluster centers as the final clustering. Reassign remaining $n - n_c$ points to the "don't care" set.

Using *MaxBall*, we modified K-Means and Agglomerative clustering as follows:

K-Means: Since the centers output by K-Means are means of the k clusters generated, we can directly apply MaxBall step 3 to obtain a clustering of n_c points. We refer to the resultant algorithm as *MaxBall K-Means*. Since K-Means uses Squared Euclidean distance, it is not suitable for clustering gene-expression data. However, it is easy to modify K-Means to work with Pearson Distance, a biologically relevant measure (equation 2). This modification is similar to that of spherical K-Means [DM01], except that the recomputed centers are required to be z-scored (just as the points are z-scored in equation 2) after each iteration. This modified version of K-Means is used to run experiments on Lee and Gasch datasets.

Agglomerative: One way to obtain k clusters from Agglomerative clustering is to split the cluster dendrogram at a level that gives exactly k clusters. Applying the MaxBall procedure to clusters found using Agglomerative Single Link results in *MaxBall-SL*, while other variants such as Agglomerative Complete Link and Average Link result in *MaxBall-CL* and *MaxBall-AL* respectively. The performances of average and complete link derivatives were comparable to that of the single link derivate. Therefore, for brevity, we present results on Auto-HDS for Single Link and Complete Link, and for DS against Single Link.

The MaxBall procedure was also applied on DS in order to compare it with Auto-HDS.

DBSCAN: For the sake of discussion, we define coverage as the fraction of points clustered (i.e., n_c/n). As discussed earlier, the clustering obtained using DBSCAN is similar to that of DS, and identical to a special case of HMA described by [Wis68]. However in contrast with

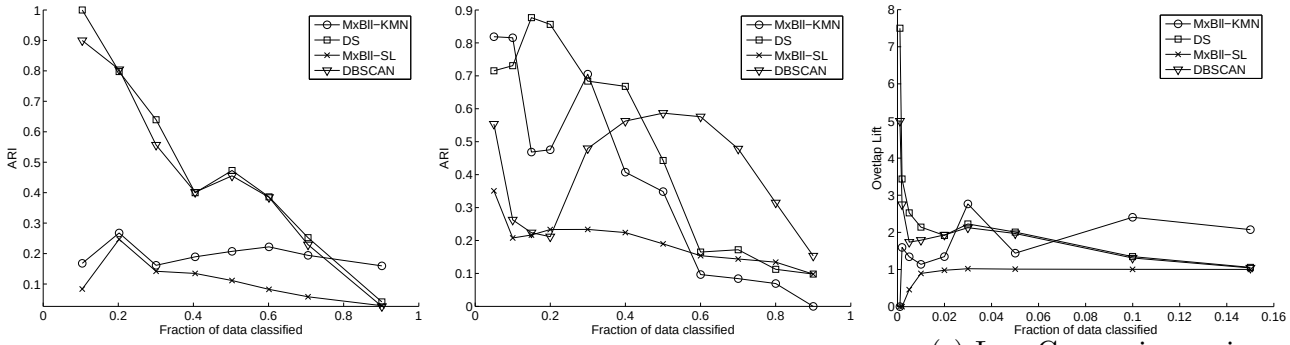
[11] using the same distance measure as that used by DS.

DS, it is not possible to control the coverage directly in DBSCAN, which is essential for a fair comparison against other methods. Therefore, for the two DBSCAN parameters, we set $MinPts$ to 4 as recommended by [EKSX96], and then perform a search for an Eps that results in the desired fraction of data in clusters.

Comparing DS and Auto-HDS with Benchmarks: For DS, comparisons with other benchmarks were performed across a range of coverages. Since varying the coverage for DS results in varying k, the corresponding k is used as an input to the benchmark algorithms except for DBSCAN.

For Auto-HDS, which is a deterministic algorithm, it is not possible to vary n_c; the corresponding k and n_c output by Auto-HDS are used along with the MaxBall procedure described earlier to obtain results summarized in Table 3, except for DBSCAN for which we used the same procedure to control the coverage as that for DBSCAN comparisons with DS. Note that the number of clusters k cannot be controlled for DBSCAN.

9.4 Results



(a) Gasch, Comparison using ARI (b) Sim-2, Comparison using ARI (c) Lee, Comparison using Overlap Lift

Figure 4: ARI comparisons of DS with other methods on Gasch (a) and on the Sim-2 (b) datasets. (c) compares DS with other benchmarks on the Lee dataset using Overlap Lift. Results for MaxBall K-Means were averaged over 100, 10 and 3 trials for the Sim-2, Gasch and Lee datasets.

Table 3: Comparisons of the benchmarks with Auto HDS using ARI on Gasch and Sim-2 data. For Gasch data, $k = 11$ and coverage was 62.4%. For Sim-2 data, $k = 5$ and the coverage was 48.4%.

Dataset:	Gasch	Sim-2
Auto-HDS	**0.3509**	**0.6985**
MxBll-DS	0.1533	0.5414
MxBll-KMN	0.2301	0.6433
MxBll-SL	0.1304	0.1948
MxBll-CL	0.2511	0.5114
DBSCAN	0.0234	0.5711

General results: Figure 4(a) and (b) compare DS with the other three benchmark algorithms on the Gasch and Sim-2 dataset using ARI over a range of fraction of data clustered (x-axis) respectively, while 4(c) shows performance comparison on the Lee dataset using Overlap Lift. In general, for lower coverages that correspond to dense regions, DS tended to perform very well. Auto-HDS, which selects clusters automatically, performed better than the other methods for detecting the most significant dense regions in the data, and the discovered k matched well with the number of classes (Table 2). Furthermore, Auto-HDS clusters also matched well with the true labels in the target classes. This can be seen in the highly correlated expression patterns across gene experiments (e.g. Figure 6 (b), (c), (e), (f), (i), and Figure 5), when evaluating against known functional categories (Table 4) or class labels (Figure 6 (g) and (h), Figure 2 (i) and (l), and Table 3). It should be stressed that since k is discovered by our framework and is given as an input to the benchmarks (except for DBSCAN where it is not possible to directly control k), they are not a viable alternative to our framework for finding dense regions automatically. Also, DBSCAN tended to over-split the clusters for the Sim-2 dataset, resulting in a much larger number of clusters (between 17 and 48) than the number of classes, which was 5.

Robust model parameters: Auto-HDS is also very robust to the choice of the two major model parameters n_ϵ and n_{part} (Figure 2, (i) vs. (l)). For high-dimensional gene-expression data, usually small values for both works well. Furthermore, r_{shave} is only a speed parameter and does not effect the shape of the HMA hierarchy discovered; smaller values of the shaving rate r_{shave} give slightly higher resolution hierarchies and more precise cluster boundaries. For all experiments we used r_{shave} in the range of 0.01 and 0.05.

Clustering Gasch experiments: The hierarchy found by Auto-HDS on the extremely high-dimensional Gasch dataset is quite compact and easy to interpret (Figure 6(a)). Many of the 11 clusters discovered by Auto-HDS (Figure 6(d)) contain highly correlated experimental descriptions, while others that form siblings have closely related descriptions. For example, a particularly interesting pair of sibling clusters $\mathcal{A}$ and $\mathcal{B}$ are shown in Figure 6, (g) and (h). Both clusters contain heat shock experiments. However, the heat shock experiments in cluster $\mathcal{A}$ involve a constant heat (37 degrees) and variable time, while the heat shock experiments in cluster $\mathcal{B}$ involve variable heat and constant time. Additional such examples can be found at our website.

Table 4: Example high purity clusters from Auto-HDS, Lee dataset. Note that Funspec returns multiple categories (column 3) with low p-values for a given cluster; (x/y) stands for the coverage, where we found x out of y known members of a category.

C. Id.	$\|\mathcal{C}\|$	Cat(Cov.)	p-val
2	8	Nucleosomal protein complex (8/8)	<1e-14
3	7	PF00674-DUP (6/7)	1.132e-14
5	11	glycolysis (7/16)	3.175e-14
5	11	cytoplasm (11/554)	3.175e-14
6	120	Cytoplasmic ribosomes (111/138)	<1e-14
7	7	PF00660-SRP1_TIP1 (6/30)	<1e-14
7	7	stress (5/175)	<1e-14

Clustering Genes, Lee: Automating clustering for biological data sets in general, and gene-expression datasets in particular, is a hard problem that motivated the design of Auto-HDS. One of the critical issues facing biologists analyzing the vast stream of biological data is that obtaining pure gene clusters often requires significant manual pruning of the clustering results. With $n_\epsilon = 4$ and $n_{part} = 2$, Auto-HDS found 9 clusters in the Lee dataset formed by 182 out of 5,612 genes. After pruning such vast numbers of genes, most of the clusters were very pure when evaluated using FunSpec and show very small p-values. Some of the high purity clusters with extremely low p-values are summarized in Table 4. More details on these clusters are available online on our website. The most surprising among these clusters is Cluster 6 where 111 out of the 120 genes in the cluster belong to a known biological process category - *Cytoplasmic ribosomes* that has only 138 known members. Given that there are 5,612 genes to pick from, this accuracy is remarkable.

Another popular approach for quickly verifying the quality of gene-expression clustering is by visualizing the clustered genes in the original feature space. For five of the clusters from Table 4, Figures 6(b), (c), (e), (f) and (i) show the gene-expression level of a sample of genes from the corresponding cluster across 591 experiments in the Lee dataset. Clearly the genes are highly correlated. A high degree of correlation is also visible in the middle plot in Figure 5, where the (182) genes belonging to all the clusters discovered from the Lee data were sorted based on the discovered HDS hierarchy.

10 Concluding Remarks

In this paper, we have introduced Auto-HDS, a framework that is well-suited for unsupervised learning on large, high-dimensional datasets such as those in bioinformatics, where the labeled datasets are complex and incomplete and where the solution set is often hidden in a small subset of the data. A key property of our framework is the ability to find a compact hierarchy. This stems from the ability to ignore less dense points while generating the hierarchy. Auto-HDS is also able to automatically discover the size, number and location of dense clusters. In particular, the remarkably pure gene clusters discovered by Auto-HDS on Lee data (Table 4) lend support to the hypothesis that Auto-HDS is suitable for the problem of clustering genes for pathway discovery.

The Gene DIVER product, a highly scalable and memory efficient implementation of Auto-HDS, includes several key innovations that make interactive clustering of large biological datasets practical on modest computers. We hope that Gene DIVER will eventually become a popular clustering tool for many clustering problems in general, and for bioinformatics in particular.

Acknowledgments: This research was supported by NSF grants IIS-0325116 and IIS-0307792.

References

[ABKS99] M. Ankerst, M. Breunig, H. P. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. In *ACM SIGMOD Conference*, 1999.

[BMDG05] A. Banerjee, S. Merugu, I. Dhillon, and J. Ghosh. Clustering with Bregman divergences. *JMLR*, 6:1705–1749, 2005.

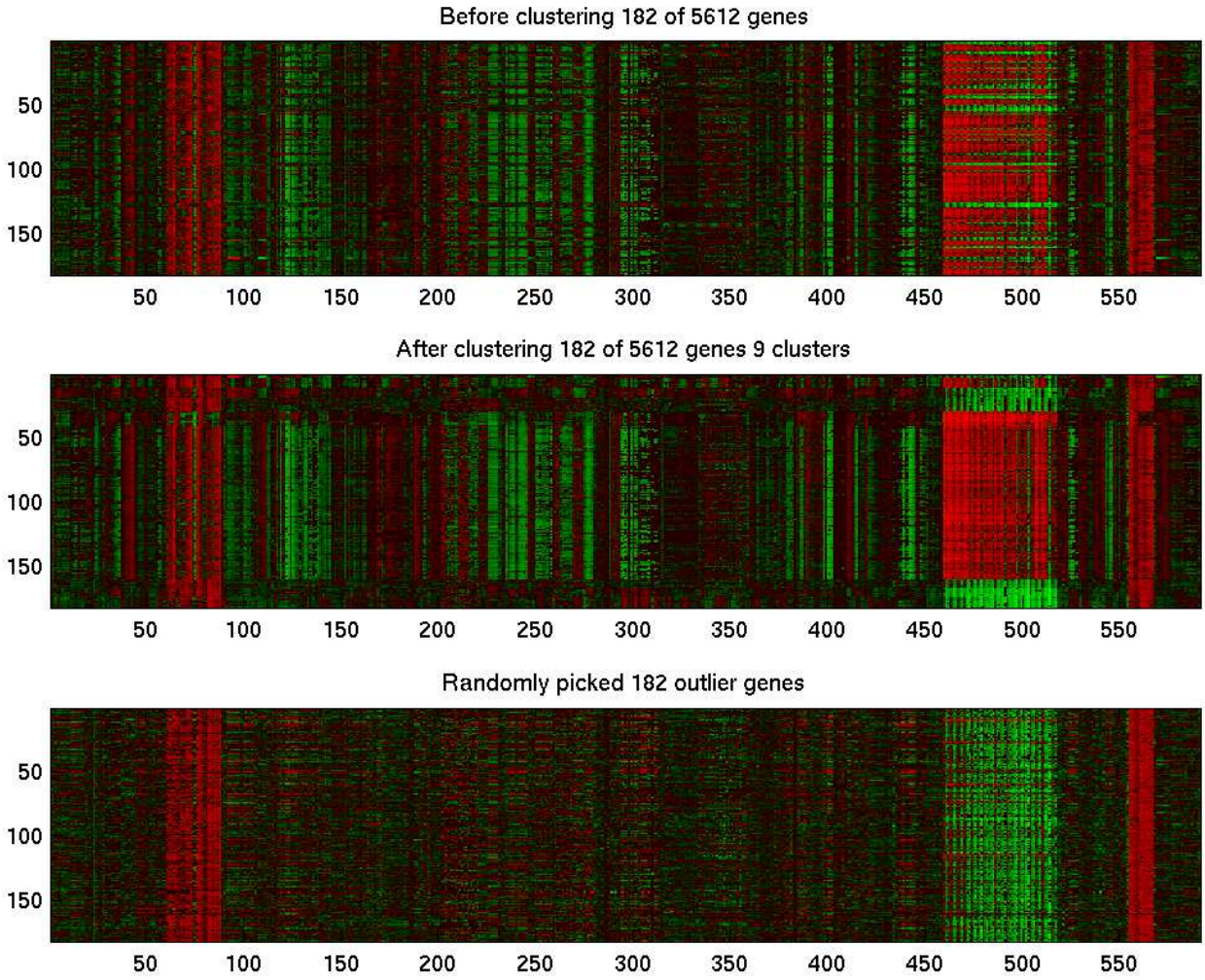
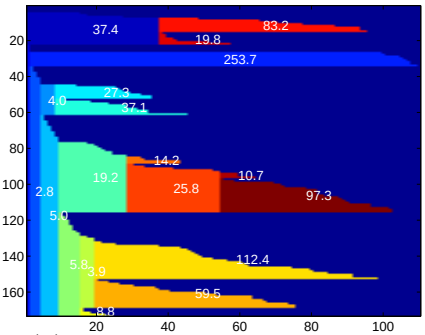
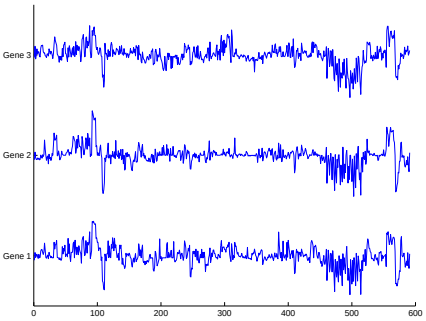


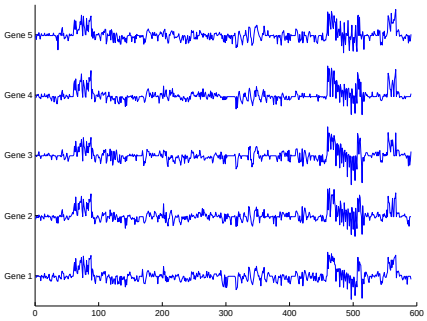
Figure 5: The 182 genes belonging to the 9 clusters found on the Lee data appear highly correlated when sorted using the discovered Auto-HDS hierarchy. The plot is shown in a popular red-green format used for visualizing gene-expression data; red represents under-expressed genes, while green implies over-expressed genes. The x-axis represents the 591 experiments while the y-axis represents the 182 genes. Within the plot are 3 subplots; the top and middle plots respectively are shown before and after sorting genes using rows of HDS label matrix $\mathbf{L}$, while the bottom row consists of 182 random genes for comparison.



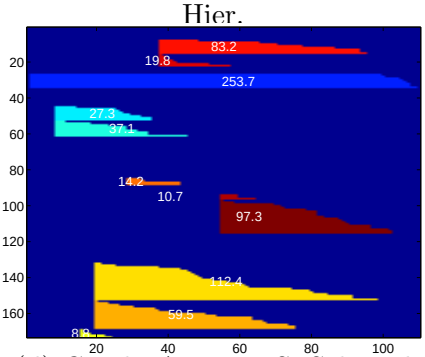
(a) Gasch, Auto-HDS, Clust. Hier.



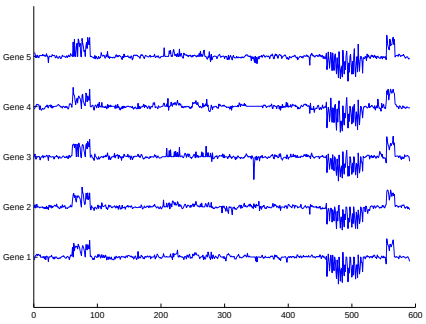
(b) Lee, Cluster 1, $|\mathcal{C}| = 3$



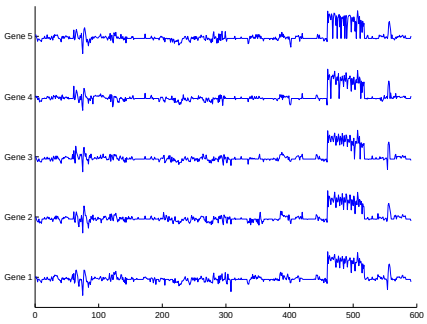
(c) Lee, Cluster 2, $|\mathcal{C}| = 8$



(d) Gasch, Auto-HDS, Selected Clusters



(e) Lee, Cluster 4, $|\mathcal{C}| = 11$



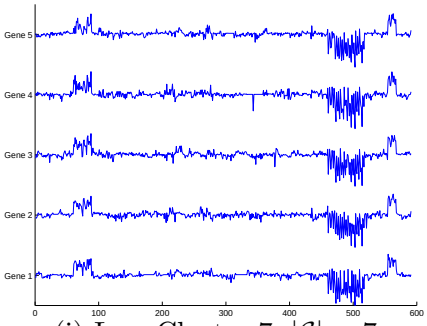
(f) Lee, Cluster 5, $|\mathcal{C}| = 11$

Heat Shock 05 minutes hs-1 1
Heat Shock 10 minutes hs-1 1
Heat Shock 15 minutes hs-1 1
Heat Shock 20 minutes hs-1 1
Heat Shock 30 minutes hs-1 1
Heat Shock 40 minutes hs-1 1
Heat Shock 60 minutes hs-1 1
Heat Shock 80 minutes hs-1 1

(g) Gasch, Cluster $\mathcal{A}$, $|\mathcal{C}| = 8$

Heat Shock 17 to 37,20 minutes 1
Heat Shock 21 to 37,20 minutes 1
Heat Shock 25 to 37,20 minutes 1
Heat Shock 29 to 37,20 minutes 1
Heat Shock 33 to 37,20 minutes 1
DBY7286 37deg.heat-20min12
DBYyap1-37deg.heat-20min(r)9
DBY7286+.3mMH2O2(20min)9
DBYyap1-+.3mMH2O2(20min)9

(h) Gasch, Cluster $\mathcal{B}$, $|\mathcal{C}| = 9$



(i) Lee, Cluster 7, $|\mathcal{C}| = 7$

Figure 6: (a) and (d): Demonstration of Auto-HDS clustering and visualization of Gasch experiments showing the effectiveness of the 2-D projection of the 6,151 dimensional Gasch data; related clusters form "siblings" that are located close to each other; their size, density and stability differences are easy to compare. (g) and (h): Sibling clusters discovered by Auto-HDS on the Gasch dataset. (b),(c), (e), (f) and (i): Expression patterns across 591 experiments (x-axis) for a sampling of genes within clusters discovered by Auto-HDS on the Lee dataset are highly correlated.

[CDGS04] H. Cho, I. S. Dhillon, Y. Guan, and S. Sra. Minimum sum-squared residue co-clustering of gene expression data. In *Fourth SIAM International Conference on Data Mining*, pages 114–125, April 2004.

[DM01] Inderjit S. Dhillon and Dharmendra S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1-2):143–175, January-February 2001.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *In Proc. KDD-96*, 1996.

[Gas00] Gasch A. P. et al. Genomic expression programs in the response of yeast cells to environmental changes. *Mol. Bio. of the Cell*, 11(3):4241–4257, December 2000.

[GG05] Gunjan Gupta and Joydeep Ghosh. Robust one-class clustering using hybrid global and local search. In *Proc. ICML 2005*, pages 273–280, Bonn, Germany, August 2005.

[Gol03] Gollub J. et al. The stanford microarray database: data access and quality assessment tools. *Nucleic Acids Res*, 31:94–96, 2003.

[HA85] L. Hubert and P. Arabie. Comparing partitions. *Journal of classification*, pages 193–218, 1985.

[Has00] Hastie T. et al. Gene shaving as a method for identifying distinct sets of genes with similar expression patterns. *Genome Biology*, 1:1–21, 2000.

[JNSRL94] Hwang Jenq-Neng, Lay Shyh-Rong, and A. Lippman. Nonparametric multivariate density estimation: a comparative study. *Science*, 42(10):2795–2810, October 1994.

[JPZ03] Daxin Jiang, Jian Pei, and Aidong Zhang. DHC: A density-based hierarchical clustering method for time series gene expression data. In *BIBE '03*, page 393, Washington, DC, USA, 2003. IEEE Comp. Soc.

[LDAM04] I. Lee, S. V. Date, A. T. Adai, and E. M. Marcotte. A probabilistic functional network of yeast genes. *Science*, 306:1555–1558, 2004.

[LO02] L. Lazzeroni and A. B. Owen. Plaid models for gene expression data. *Statistica Sinica*, 12(1):61–86, January 2002.

[MTea04] Robert Mansson, Panagiotis Tsapogas, and Mikael Akerlund et al. Pearson correlation analysis of microarray data allows for the identification of genetic targets for early b-cell factor. *J. Biol. Chem.*, 279(17):17905–17913, April 2004.

[SS00] R. Sharan and R. Shamir. Click: A clustering algorithm with applications to gene expression analysis. *Proc. 8th ISMB*, pages 307–316, 2000.

[STG+03] Eran Segal, B. Taskar, A. Gasch, N. Friedman, and D. Koller. Rich probabilistic models for gene expression. *Bioinformatics*, 17(1):243–252, 2003.

[Stu03] Werner Stuetzle. Estimating the cluster tree of a density by analyzing the minimal spanning tree of a sample. *Journal of Classification*, 20:25–47, 2003.

[TdSL00] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 2000.

[Wis68] D. Wishart. Mode analysis: A generalization of nearest neighbour which reduces chaining effects. In *Proceedings of the Colloquium in Numerical Taxonomy*, pages 282–308, University of St. Andrews, Fife, Scotland, September 1968. Academic Press.