Dynamic Tree Cut: in-depth description, tests and applications

Peter Langfelder [a*], Bin Zhang [b*], Steve Horvath [a†]
[a]Dept. of Human Genetics, University of California at Los Angeles, CA 90095-7088
[b]Rosetta Inpharmatics-Merck Research Laboratories, Seattle, WA

February 5, 2009

Abstract

In hierarchical clustering, clusters are defined as branches of a cluster tree. The constant height branch cut, a commonly used method to identify branches of a cluster tree, is not ideal for cluster identification in complicated dendrograms. We describe a new dynamic branch cutting approach for detecting clusters in a cluster tree based on their shape. Compared to the constant height cutoff, our techniques offer the following advantages: (1) they are capable of identifying nested clusters; (2) they are flexible: cluster shape parameters can be tuned to suit the application at hand; (3) they are suitable for automation; (4) we find that they work well for finding modules in protein–protein interaction and gene co-expression networks. Additionally, our methods can optionally combine the advantages of hierarchical clustering and partitioning around medoids, giving better detection of outliers.

We describe the Dynamic Tree Cut algorithms in detail and give examples illustrating their use. The Dynamic Tree Cut package and example scripts, all implemented in R language, can be downloaded from http://www.genetics.ucla.edu/labs/horvath/CoexpressionNetwork/BranchCutting.

1 Why Dynamic Tree Cut?

Hierarchical clustering is a popular data mining method for detecting clusters of closely-related objects in data [7]; a major application in bioinformatics is clustering of gene expression profiles. Hierarchical clustering organizes objects into a hierarchical cluster tree (dendrogram) whose branches are the desired clusters. The process of identifying individual branches is variously referred to as branch or tree cutting or dendrogram pruning.

The most widely used tree cut method is the *fixed height branch cut*: the user chooses a fixed height on the dendrogram, and each contiguous branch of objects below that height is considered a separate cluster. When detecting gene clusters (also referred to as modules), one typically also requires each cluster to have size at least N_0, a chosen constant number.

The fixed height branch cut is a simple and elegant technique with many desirable properties, but it is not ideal in situations where one expects a complicated dendrogram structure with nested clusters. Examples of such situations are described in Section 4.2. It should be noted that this is not a fault of hierarchical clustering as such: often the dendrogram exhibits distinct branches corresponding to the desired modules, but no single fixed cut height can identify them correctly. To automatically detect the clusters, the tree cut method should identify branches based on their shape, not on absolute height.

For the benefit of the reader, we now briefly review hierarchical clustering and the structure of the resulting dendrograms; for more details we refer the reader to any of a large number of textbooks such as [4]. Hierarchical clustering is a class of agglomerative clustering techniques that iteratively merge two closest objects into a new composite object (cluster). Composite objects are further merged with other (original or composite) objects until all objects have been merged. Hierarchical clustering methods differ primarily on how the dissimilarity

*Joint First Authors.
†to whom correspondence should be addressed

between clusters is calculated given the dissimilarities of their constituents. In this work we concentrate on *average linkage* hierarchical clustering, in which the dissimilarity of two clusters A, B is the average of pairwise dissimilarities between all pairs of objects (a, b) such that $a \in A, b \in B$.

Hierarchical clustering methods produce a *dendrogram* which is a data structure containing information on which objects (singletons or clusters) were merged at each step and on their *merging height*, that is the dissimilarity of the two merged objects that were merged at the particular step. By construction, the sequence of merging heights is increasing. The number of merges (and merging heights) is always $n - 1$, where n is the number of input objects to be merged. Dendrograms are typically plotted by arranging the input objects along the x-axis and joining them by lines at the height (y-axis) corresponding to their merging heights; the result looks like an (up-side down) tree whose branches correspond to clusters and "leaves" correspond to the input objects. The input objects are permuted such that the lines in the resulting plot do not cross.

2 Algorithm and implementation

We developed a new cluster detection technique based on analyzing the shape of the branches on the dendrogram. To provide more flexibility, we present two variants of the method. The first variant, called "Dynamic Tree" cut, is a top-down algorithm relying only on the dendrogram and respecting the order of the clustered objects on it. This method has been utilized for identifying biologically meaningful gene modules in gene co-expression networks from several species such as yeast [2], [3], mouse [6] and human cell lines [5], but has not previously been systematically described nor made publicly available. The second variant, which we call the "Dynamic Hybrid" cut, builds the clusters from bottom up. In addition to information from the dendrogram, it utilizes dissimilarity information among the objects. This hybrid approach improves the detection of outlying members of each cluster. We describe each of the algorithms in turn below.

2.1 Dynamic Tree algorithm

The dynamic tree cut algorithm works through iteration of an adaptive process of cluster decomposition and combination until the number of clusters becomes stable.

We first define terminology. Consider a real-valued vector $S = (s_1, s_2, ..., s_n)$. We call a point i a *transition point* if $s_i > 0$ and $s_{i+1} < 0$. The *forward run length* of a point i in S, denoted $r(i)$, is the number of consecutive points immediately before i whose values in S have the same sign as s_i. The *breakpoint* corresponding to a transition point i, denoted as $b(i)$, is given by $i - r(i)$. Figure 1 illustrates the notions described above.

2.1.1 Procedure $\mathcal{T}reecutCore()$

An essential and basic procedure in the algorithm is procedure $\mathcal{T}reecutCore()$. Figure 2b) shows the flowchart of the procedure. We now describe the procedure in detail.

$\mathcal{T}reecutCore()$ takes as inputs a dendrogram height sequence $\mathcal{H} = (h_1, h_2, ..., h_n)$ and a reference height l. First the differences between the elements of $\mathcal{H}$ and l, i.e., $\hat{\mathcal{H}} = \mathcal{H} - l = (\hat{h}_1, \hat{h}_2, ..., \hat{h}_n)$ are calculated. Then, transition points in $\hat{\mathcal{H}}$ are found. For each transition point we compute its forward run length and locate the corresponding breakpoint. A break point is considered significant if its forward run length is greater than a threshold τ. Since a breakpoint separates two neighboring clusters, any two consecutive breakpoints define a cluster. Note that the actual value of l is not important as long as it is not too high (higher than the highest joining height in the cluster) or too low (below the lowest joining height in the cluster). If it is either too high or too low, the particular cluster will not be identified as a separate cluster and will instead be joined with neighboring clusters.

We allow the user to enforce a minimum cluster size requirement if tiny clusters are to be avoided. To avoid losing all tiny clusters, a post-processing step is performed in which the tiny clusters are attached to their neighboring major clusters if the mean tree height of the tiny cluster is smaller than the one of the major cluster. As output, the procedure returns significant clusters.

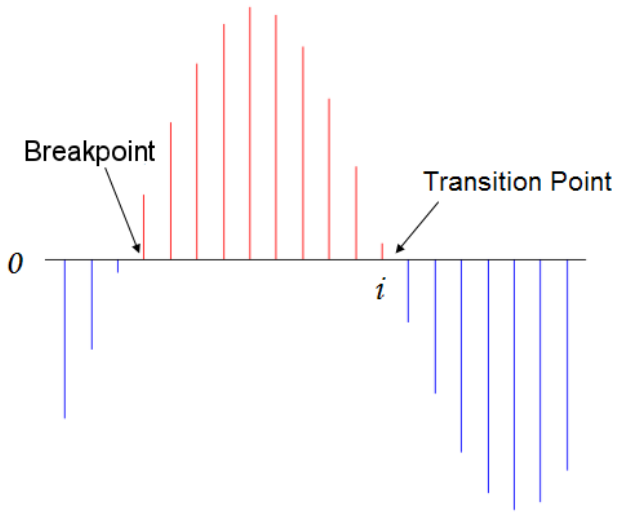


Figure 1: Illustration of the notions of transition point, run length and breakpoint. Bars correspond to elements of a vector S; red denotes positive and blue negative elements s_j. Here i is a transition point with a forward run length 9; elements within the run length are colored red. The breakpoint of i is $i - 9$.

2.1.2 Procedure *AdaptiveTreecutCore*()

As the procedure $\mathcal{T}reecutCore()$ takes into account only a single reference height, it may fail in the situations where one sub-cluster dendrogram sits well above or below a given reference height l. To solve this problem, our algorithm uses multiple heights to deal with such complex situations. Based on $\mathcal{H}$, we define three reference heights, l_m, l_u and l_d as follows:

$$l_m = \frac{1}{n}\sum_{i=1}^{n} h_i\,, \tag{1}$$

$$l_u = \frac{1}{2}(l_m + max\{h_1, h_2, ..., h_n\})\,, \tag{2}$$

$$l_d = \frac{1}{2}(l_m + min\{h_1, h_2, ..., h_n\})\,. \tag{3}$$

The procedure $\mathcal{A}daptiveTreecutCore(H)$ calls the basic cluster identification procedure, $\mathcal{T}reecutCore()$, adaptively, as shown in Figure 2b).

Compute l_m, l_u *and* l_d*;*
Call $\mathcal{T}reecutCore(H, l_m)$*;*
if (no new cluster detected)
then
  Call $\mathcal{T}reecutCore(H, l_d)$*;*
  if (no new cluster detected)
  then
    Call $\mathcal{T}reecutCore(H, l_u)$*;*

2.1.3 The Dynamic Tree Cut Algorithm

The flowchart of the Dynamic Tree algorithm is shown in Figure 2a). Given a dendrogram from hierarchical clustering, the algorithm first calls the fixed height cut procedure [1] with a given, typically very high, cut

height (for example, if the dissimilarity range is 0—1, an appropriate cut height is 0.99). This returns a starting set of large clusters that will be decomposed into smaller ones by subsequent processing. For each starting cluster we isolate its ordered dendrogram. Thereby, we obtain a set of cluster-based dendrograms, denoted by $\Omega = \mathcal{H}_1, \mathcal{H}_2, ..., \mathcal{H}_m$. Next, each cluster is processed by $\mathit{AdaptiveTreecutCore}()$. If new clusters are generated during the process, Ω is updated with newly generated clusters and $\mathit{AdaptiveTreecutCore}()$ is called for each cluster. This process is iterated until no new clusters are produced.

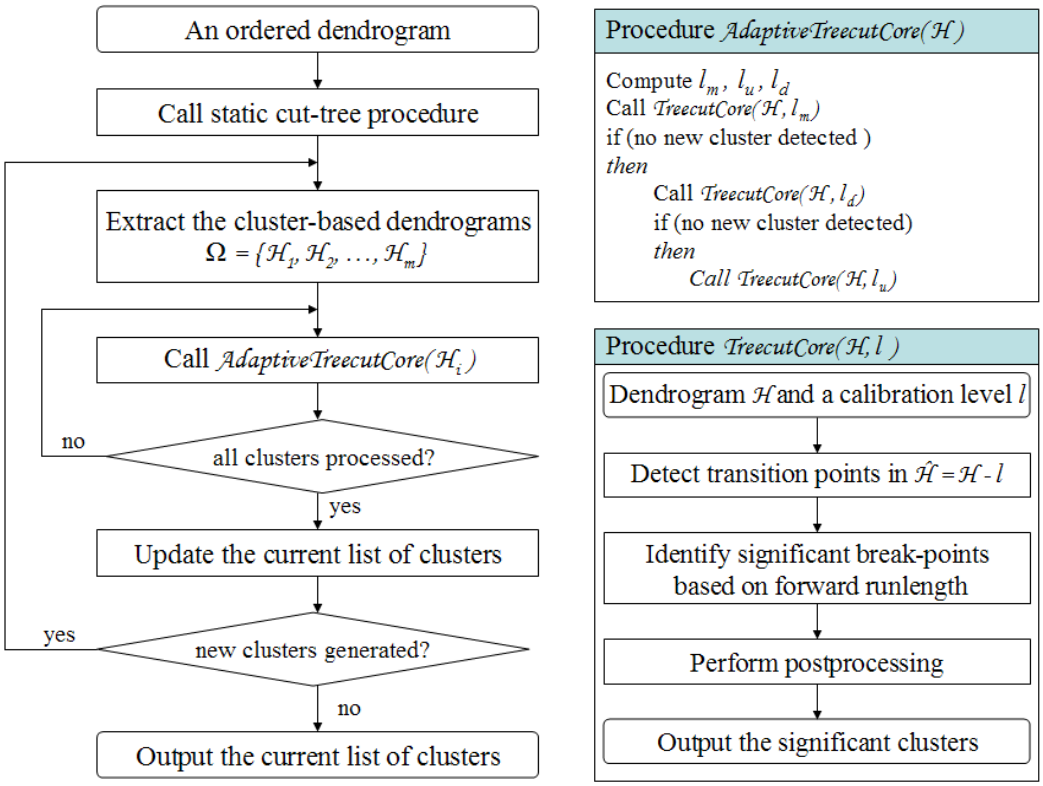


Figure 2: The dynamic cut-tree algorithm: (a) the overall flowchart; (b) the flowcharts of the procedures $\mathit{AdaptiveTreecutCore}$ (the top block) and $\mathit{TreecutCore}$ (the bottom block).

2.2 Dynamic Hybrid algorithm

In a sense, this variant is a complement to the top-down Dynamic Tree method: it builds clusters in a bottom-top manner in two steps. In step 1, branches that satisfy specific criteria (detailed below) for being clusters are detected. This "branch pruning" step is based on the merging information of the dendrogram (but not on the order of the clustered objects). In step 2, all previously unassigned objects are tested for sufficient proximity to clusters detected in the first step; if the closest cluster is close enough in a precise sense that we define below, the object is assigned to that cluster. In this step the dendrogram is ignored and only dissimilarity information is used. Step 2 can be considered a modified k-medoid partitioning (also known as Partitioning Around Medoids, PAM [8]); hence the method is best described as a hybrid of hierarchical clustering and modified k-medoid partitioning.

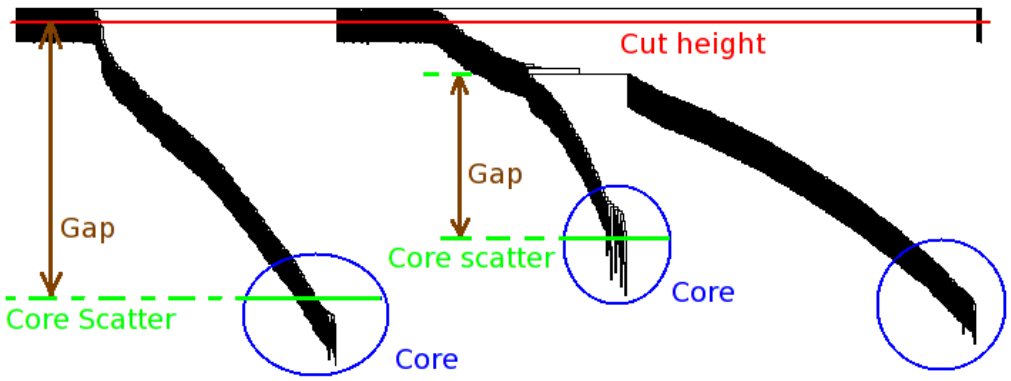


Figure 3: Illustration of the notions used to define clusters in step 1 of the Dynamic Hybrid algorithm. Shown is a simple simulated dendrogram with 3 branches whose joining heights differ. Cut height corresponds to h_{max}.

2.2.1 Step 1: Detection of clusters

Clearly, criteria defining a cluster vary from application to application. To provide flexibility in cluster definition, we have chosen the following four shape criteria: (1) a cluster is required to have a certain minimum number of member objects; (2) objects too far from a cluster are excluded from this cluster even if they belong to the same branch of the dendrogram; (3) each cluster should be separated from its surroundings by a gap; and (4) the *core* of each cluster should be tightly connected, where by core we mean the lowest-merged objects in the cluster.

To give a more precise formulation of the four criteria above, we define the following concepts. Given a cluster core, we denote by $\bar{d}$ its scatter, that is the average of all pairwise dissimilarities between objects belonging to the core of the cluster. We define the *cluster gap* g as the difference between $\bar{d}$ and the joining height where the cluster attaches to the rest of the dendrogram. Figure 3 illustrates the definitions on a simple dendrogram. In our implementation, we chose the following formula for the number n_c of objects in the cluster core: $n_c = \min\left\{\text{int}\left(N_0/2 + \sqrt{N - N_0/2}\right), N\right\}$, with N_0 the minimum cluster size and N the total number of objects in the branch or cluster. The rationale behind this choice is to make the cores of small clusters almost as large as the whole clusters, while for large clusters the cores are a small fraction of lowest-joined objects (*e.g.*, choosing $N_0 = 8$, if there are $N = 8$ objects in the cluster, the core size is $n_c = 6$, while for $N = 400$ the core size is $n_c = 23$).

Formalized, a branch is considered a cluster if it satisfies all of the following conditions: (1) it has at least N_0 singletons (original objects) on it, (2) all joining heights are at most h_{max}, (3) the cluster gap g is more than a minimum gap g_{min}; and (4) the core scatter $\bar{d}$ is at most d_{max}. The parameters N_0, h_{max}, g_{min} and d_{max} are given (user-supplied) constants.

The algorithm goes through the dendrogram from the bottom-most (first) to the top-most (last) merge. Each merge can be one of three types: a merge of two objects (singletons), a merge of an object and a branch, and a merge of two branches. For each merge the following is performed: if it is a merge of two singletons, a new branch is started (containing the two singletons). If it is a merge of a singleton and a branch, the singleton is added to the branch. If it is a merge of two branches, the algorithm tests both branches for satisfying the cluster criteria detailed above. If at most one of the branches satisfies the cluster criteria detailed above, the two branches are merged into a single one. If both branches satisfy the cluster criteria, both branches are declared "closed", that is no further objects are added to them in this step. Objects joining the dendrogram directly above a merge of two clusters are left unlabeled, because the dendrogram itself provides no information as to which of the two clusters a particular object is closest to. This is a major distinction with respect to the

(non-hybrid) Dynamic Tree variant which assigns objects above a merge to the cluster that is nearest in the dendrogram ordering.

2.2.2 Step 2: Assignment of unlabeled objects to nearest clusters

This optional, Partitioning Around Medoids-like step is based on average dissimilarities between unlabeled objects and the basic clusters detected in step 1. In general, unlabeled objects are treated one by one; the exception are branches that, of the four criteria in step 1, failed the size criterion only. We call such branches *tiny clusters*. To avoid spurious divisions, tiny clusters are not split, that is a tiny cluster is either assigned to a basic cluster as a whole, or all objects on it remain unlabeled. The user has the option of turning the recognition of tiny clusters off if desired.

As a default setting, for each unassigned object or tiny branch can only be assigned to a basic cluster that lies on the same (large) branch of the dendrogram. We refer to this restriction as respecting the dendrogram. The restriction can optionally be removed, in which case an unassigned object will be assigned to the nearest basic cluster irrespective of where the nearest cluster lies on the dendrogram.

Two methods of calculating the object–cluster dissimilarities for this step are available. The recommended method is to use average dissimilarities of unlabeled objects to clusters. The alternative is to calculate the medoid for each cluster and use object–medoid distances (medoid–medoid in the case of tiny branches). The medoid of a cluster is defined as the object with the smallest average dissimilarity to all other objects in the cluster. Whichever object–cluster dissimilarity is used, the object or tiny cluster is assigned to the closest cluster if the corresponding dissimilarity is either (1) smaller than a given maximum allowable dissimilarity given by the user, or (2) it is smaller than the cluster "radius". When using average dissimilarities, the "radius" is defined as the maximum (over objects within the cluster) of the average dissimilarity of each object with the rest of the cluster. If medoids are used, the "radius" is the maximum of the dissimilarities of the cluster's medoid to all other objects in the cluster.

3 Choice of detection parameters

Our methods provide no built-in mechanism to select an "optimal" partition or, equivalently, optimal cutting parameters. The selection is entirely up to the user, although we do provide default settings that worked well in several past applications.

The Dynamic Tree cut is somewhat less sensitive to parameter choice but also less flexible of the two methods presented here. The maximum height `heightCut`, or h_{max}, can be set quite high, although under certain circumstances a high cut may lead to some detected clusters being larger than optimal. The minimum cluster size and the `deepSplit` parameter should be set according to the desired output; their meaning is quite straightforward and easily interpreted from the dendrogram.

The Dynamic Hybrid method uses more parameters, making it more flexible but also placing the burden of choosing good settings on the user. For users who do not wish to get involved in the details, a simple control of the relative sensitivity to cluster splitting is provided in the parameter `deepSplit`, which can take values 0,1,2,3. The value 0 is analogous to `deepSplit = FALSE` for the Dynamic Tree method and will produce relatively few, large and well-defined clusters. Values 1,2,3 will progressively produce a larger number of clusters that are allowed to exhibit larger core scatter and may be separated by smaller gaps, akin to setting `deepSplit = TRUE` in the Dynamic Tree variant. We recommend decreasing the minimum cluster size when using higher settings of `deepSplit`.

The decision whether to enable the PAM-like stage of assigment should largely be determined by whether sensitivity is to be favored over specificity or vice-versa. Users favoring specificity over sensitivity (that is, tight clusters with few mis-classifications) should disable stage 2, or at least specify the maximum allowable object–cluster dissimilarity to be zero (in which case objects are assigned to clusters only if the object–cluster dissimilarity is smaller than the radius of the cluster). Conversely, in applications favoring sensitivity over specificity it is advantageous to perform step 2 with a relatively high allowed object–cluster dissimilarity. The default, which sets the maximum allowed object–cluster dissimilarity equal to the maximum joining height h_{min}, should perform well in such situations.

The overall default settings are to use the Hybrid method with minimum cluster size 20, `deepSplit` 1, the cut height above the highest merge, and to perform the (PAM-like) step 2 with maximum dissimilarity given by the height cut. If the Tree method is chosen, the defaults are to use the same minimum cluster size 20, `deepSplit = FALSE` and cut height of $0.99h_{max}$, where h_{max} is the maximum joining height on the dendrogram.

The user should be aware that the interpretation of the minimum cluster size represented by the parameter N_0 is different in the two variants: while in the Dynamic Tree variant N_0 is the minimum size of the final clusters, in the Dynamic Hybrid method it is the minimum size of the "basic" clusters in step 1. In certain situations it may be advantageous to use a lower minimum cluster size for the Hybrid method because clusters may grow to the desired size in step 2. In both cases, the resulting cluster size will never be smaller than N_0.

The final judgment of the quality of the partition will, in most applications, be obtained by external measures, *e.g.,* functional enrichment analysis, gene ontology, module–trait significance etc. for gene expression data. Such external measures are mostly unrelated to the dendrogram; hence it is likely that no parameter setting produces an "ideal" partition in which all modules are significant, clearly distinct from one another and at the same time there is no additional splitting that would improve the (externally measured) quality of the partition. In such circumstances, we recommend using finer split settings that will produce "too many" modules, then merging modules that, based on external measures, are not clearly separated. For example, one may merge modules exhibiting functional enrichment in the same categories, or modules whose eigengenes (first principal components of the expression matrix) have very high correlation.

4 Examples

4.1 Human brain gene expression data

The newly developed tree cut methods were applied to the human brain gene expression data of [9]. In Figure 4 the Dynamic Hybrid method is compared to the constant-height ("static") cut and the Dynamic Tree cut.

4.2 Simulated examples

4.2.1 Toy example

We begin with a toy example. The "data" consist of 16 simple numbers, of which 15 fall in 3 clusters and one is outside of the clusters. The dissimilarity of numbers χ_i, χ_j equals their (absolute) difference $|\chi_i - \chi_j|$. Figure 5 shows the 16 numbers arranged on their average linkage dendrogram as well as the three clusters detected by our methods. This is an example of a dendrogram on which the constant height cutoff would fail to recover all three clusters since the numbers 19 and 38 are joined into the brown cluster higher than the turquoise and blue clusters are joined together.

4.2.2 Simulated gene expression data

To demonstrate the differences between the two variants presented here, and to compare them to a few of the currently available cluster detection methods, we applied the cluster detection methods to simulated gene expression data. Details of the simulations are given in the Appendix. The obvious advantage of simulations is that one can compare the partitions found by various algorithms to the simulated ("true") partition and thus get a sense of their relative performance.

The first simulated example consists of 3 clusters and a smaller number of unassigned objects, Figure 6 (a). In this simple example we illustrate the chief difference between the Dynamic Tree and Hybrid variants, namely the labeling of objects that lie above the merge of two clusters in the dendrogram. The second simulation consists of 10 modules of varying co-regulation, Figure 6 (b). While both Dynamic methods peform very well, the complicated structure of the dendrogram proves too difficult for the static cut which cannot find all of the clusters with one cut height. PAM depends crucially on the number of clusters in its input being correct, and even in the optimal case tends to favor large clusters at the expense of smaller ones. Quantitatively, values of the Rand index were 0.97 for Dynamic Hybrid; 0.96 for Dynamic Tree; 0.91 for Static with cut 0.92; 0.88 for Static with cut 0.995; 0.87 for PAM 9; 0.91 for PAM 10; and 0.91 for PAM 11. These values illustrate that

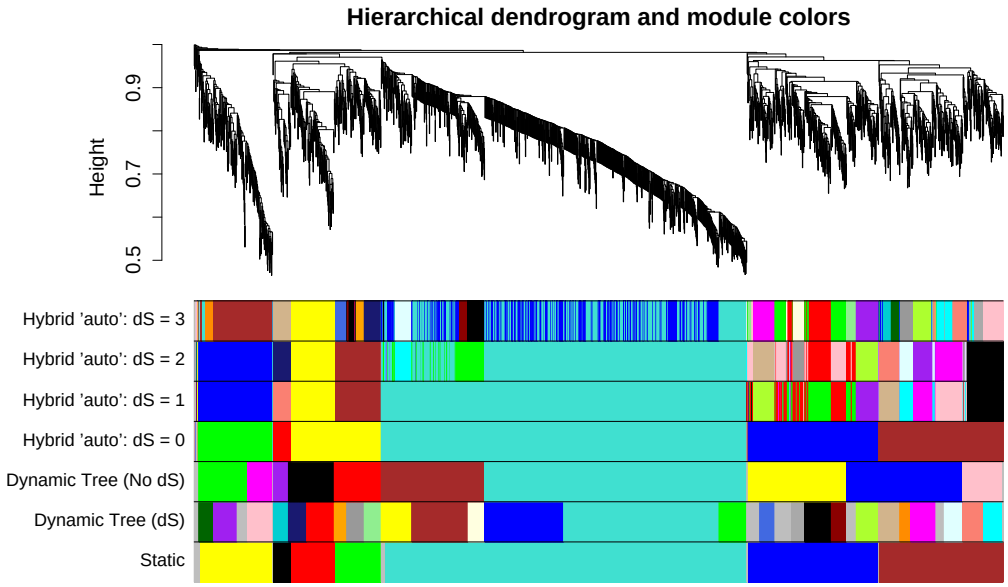


Figure 4: Example use of the newly developed tree cut methods. Human brain expression data were processed as in the analysis of [9] (see the supplement of that work for a detailed explanation of the analysis). The hierarchical dendrogram was obtained by clustering the gene expression profiles. Color bands below the dendrogram give module membership detected by the following methods: Dynamic Hybrid method with simplified parameter setting via the `deepSplit` parameter as shown, the Dynamic Tree method with `deepSplit` unset and set (DS and No DS, respectively), and using the static cut. The full R script executing this analysis can be found at http://www.genetics.ucla.edu/labs/horvath/CoexpressionNetwork/BranchCutting.

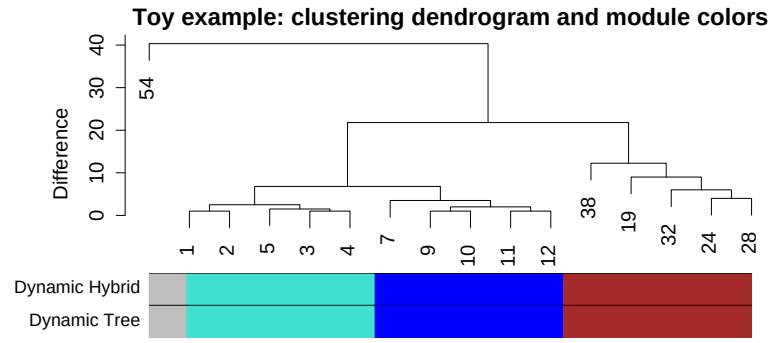
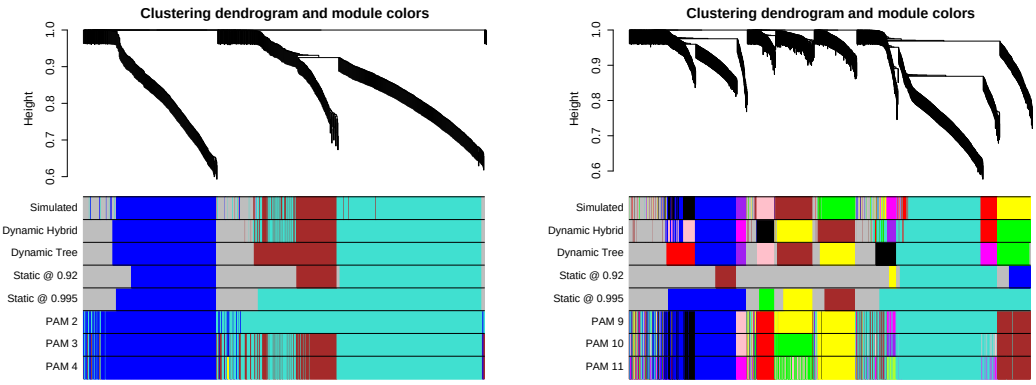


Figure 5: A toy example of Dynamic Tree Cut algorithms. The upper panel shows a dendrogram of the numbers shown below the branches whose dissimilarity (joining height) is given simply by their difference. The numbers were chosen such that 15 numbers fall into 3 clusters and one (54) is outside of the clusters. Clusters detected by our algorithms are shown by the rows of colors below the dendrogram; each cluster is assigned a color and unassigned objects are colored grey.



(a) (b)

Figure 6: Cluster detection on simulated data. Clusters of different sizes and a smaller number of unlabeled objects were simulated and the simulated data was used as input to several cluster detection methods. The upper panel displays the hierarchical clustering dendrogram of input data. On the lower panel, each row of colors shows cluster membership as simulated or determined by the corresponding cluster detection method. The methods we used are the Dynamic Hybrid and Dynamic Tree presented here, the fixed height (static) tree cut with two different cutting heights (0.92 and 0.995), and Partitioning Around Medoids (PAM). The requested number of clusters used as input for PAM is indicated by the number following the PAM label. (a) A simple example of 3 clusters. Note the difference is labeling of objects above the merge of the brown and turquoise clusters produced by the two Dynamic methods. The fixed height cut cannot detect the outliers correctly; either they are cut off from the clusters when the cut is low, or the brown and turquoise clusters will be distinguished. (b) A more complex example involving 10 clusters. Note the excellent correspondence between the simulated module membership and the ones detected by our two methods, particularly the Dynamic Hybrid one. The dendrogram structure in this example proves too complex for the fixed height cut which is unable to detect all clusters. PAM performs relatively well if the user inputs the correct number of requested clusters, but even then it favors large clusters at the expense of small ones.

the Dynamic Hybrid cut performs best, but more comprehensive simulation studies should be carried out in the future. As an aside, in the computation of the Rand index we treated each unassigned ("grey") object as a separate cluster.

A Meaning of the `deepSplit` parameter

The parameter `deepSplit` controls the sensitivity of both variants of the Dynamic Tree Cut algorithm to cluster splits. While the overall effect is the same in both variants, the details differ to some degree.

When using the Dynamic Tree variant, `deepSplit` can only take values `FALSE` and `TRUE` and controls whether, after recursively processing all clusters, the algorithm should stop or whether it should re-process all clusters until there are no new clusters detected. Re-processing leads to smaller clusters, increasing the sensitivity of the method.

When using the Dynamic Hybrid variant, `deepSplit` can take values 0,1,2,3. In this variant the `deepSplit` parameter simply provides a simplified way of setting the minimum gap g_{min} and the maximum core scatter d_{max} as follows. First, the minimum[1] and maximum joining heights on the dendrogram, h_{min} and h_{max}, are determined. The maximum core scatter is then determined as $d_{max} = h_{min} + x(h_{max} - h_{min})$, where the fraction x takes values 0.64, 0.73, 0.82, 0.91 for `deepSplit` = 0, 1, 2, 3, respectively. Analogously, the minimum gap g_{min} is determined as $g_{min} = (1 - x)(h_{max} - h_{min})$. In summary, the maximum core scatter and minimum gap

[1]More precisely, to guard against outliers, we use the 5th percentile of the joining heights instead of the minimum. In practice, the difference is minimal.

are determined as fixed fractions (controlled by `deepSplit`) of the range of joining heights on the dendrogram. If the user feels that the range of values of `deepSplit` is insufficient, the shape parameters can be set directly.

B Simulation of clustered data

In this appendix we briefly sketch our method for simulating data sets that exhibit a cluster structure. The simulation code is available on the web site mentioned in the Abstract. We simulate a set of vectors x_i, $i = 1, .., n$, where each vector has m components. For example, in clustering of microarray data each vector would correspond to one gene expression profile, and each component to the expression of the gene in one sample. The cluster structure is specified by choosing the simulated number of clusters N, cluster sizes $n_I, I = 1, .., N$, a set of N cluster seeds (one for each cluster) and maximum and minimum correlations $r_{I,max}, r_{I,min}$ of vectors within each cluster. The cluster sizes are chosen such that $n > \sum_I n_I$, that is a fraction of the vectors is simulated to be outside of the clusters (we refer to such objects as "grey" since our convention is to label unassigned objects by grey color). The cluster seeds can be chosen in many ways, for example as independent (uncorrelated) vectors or with some correlations among them. Given that in real gene expression data clusters tend to be correlated, the cluster seeds in our simulations were chosen to have a particular correlation matrix.

B.1 Simulating a single cluster

Given a cluster seed $seed_I$ and desired size n_I, cluster objects (vectors) are simulated as follows. Generate vectors such that the correlation of the k-th vector with the cluster seed is

$$\begin{aligned} \mathrm{cor}(x_{k,I}, seed_I) &= r_{I,max} - (k/n_I)(r_{I,max} - r_{I,min}) \,, & (4) \\ &\equiv r_{k,I} & (5) \end{aligned}$$

that is the first vector has correlation $r_{1,I} \approx r_{I,max}$ with the seed while the last (n_I-th) vector has correlation $r_{n_I,I} = r_{I,min}$. The required correlation (4) is achieved by calculating the k-th vector as the sum of the seed vector $seed_I$ and a suitable multiple of a random vector ϵ_k,

$$x_{k,I} = seed_I + a_k \epsilon_k \,. \quad (6)$$

Components of ϵ_k are generated independently from $N(0, 1)$. It is easy to show that choosing

$$a_k = \sqrt{\frac{\mathrm{var}(seed_I)}{\mathrm{var}(\epsilon_k)} \left(\frac{1}{r_{k,I}^2} - 1 \right)} \quad (7)$$

will indeed result in $\mathrm{cor}(x_{k,I}, seed_I)$ being given by Eq. (4). This technique results in clusters consisting of vectors distributed symmetrically around the cluster seed; in this sense, the simulated clusters are spherical clusters whose centers coincide (on average) with the cluster seed.

Grey objects, *i.e.,* objects that are not simulated to belong to any of the clusters, have components of their vectors chosen randomly and independently from $N(0, 1)$.

B.2 Simulating data sets

In Section 4.2 we used two different simulations. The first one consisted of 800 vectors with 80 components in 3 clusters of sizes 344, 120 and 208. The rest (128) were "grey" objects. The seeds of the first two clusters were created with a relatively high correlation of 0.8, while the correlation of the first and third cluster seeds was low, 0.04. Thus, the joining height of the first two clusters on the dendrogram is lower than that of the third cluster with the combined first and second cluster.

In the second simulated example we simulated 2000 vectors with 100 components in 10 clusters of sizes 400, 120, 200, 60, 100, 260, 60, 100, 220, and 190 vectors. The leftover 290 objects were simulated to be outside of the clusters. Module seeds were generated in three groups with significant correlations within each group

but no significant correlation between seeds in different groups. Maximum and minimum correlations in each cluster were chosen by hand such that the simulated clusters exhibit varying levels. We refer the reader to the R code posted on our website for more details.

The simulated vectors were processed in the same way as the real gene expression data in the example in Section 4.1. The processing is described in detail the Supplement of [9]. Briefly, the gene correlation matrix $R_{ij} = cor(x_i, x_j)$ is raised, component-wise, to a power $\beta = 9$ to suppress low correlations that may be due to noise, measurment errors etc, resulting in the gene network adjacency matrix $a_{ij} = R_{ij}^{\beta}$. To identify densely interconnected groups of genes, the Topological Overlap Matrix TOM_{ij} [10, 11] is calculated from the adjacencies. TOM is a pairwise measure of similarity and can be turned into a dissimilarity by subtracting it from 1, $Dissim_{ij} = 1 - TOM_{ij}$. This dissimilarity is used as input of the average linkage hierarchical clustering in all our examples.

References

[1] R. A. Becker, J. M. Chambers, and A. R. Wilks. *The New S Language*. Wadsworth & Brooks/Cole, 1988.

[2] Marc R.J. Carlson, Bin Zhang, Zixing Fang, Steve Horvath, Paul S. Mishel, and Stanley F. Nelson. Gene connectivity, function, and sequence conservation: Predictions from modular yeast co-expression networks. *BMC Genomics*, 7(40), 2006.

[3] Jun Dong and Steve Horvath. Understanding network concepts in modules. *BMC Systems Biology*, 1(1):24, 2007.

[4] B. S. Everitt, S. Landau, and M. Leese. *Cluster Analysis, 4th Edition*. Oxford University Press, Oxford, 2001.

[5] Peter S. Gargalovic, Minori Imura, Bin Zhang, Nima M. Gharavi, Michael J. Clark, Joanne Pagnon, Wen-Pin Yang, Aiqing He, Amy Truong, Shilpa Patel, Stanley F. Nelson, Steve Horvath, Judith A. Berliner, Todd G. Kirchgessner, and Aldons J. Lusis. Identification of inflammatory gene modules based on variations of human endothelial cell responses to oxidized lipids. *PNAS*, 103(34):12741–12746, 2006.

[6] A. Ghazalpour, S. Doss, B. Zhang, C. Plaisier, S. Wang, E.E. Schadt, A. Thomas, T.A. Drake, A.J. Lusis, and S. Horvath. Integrating genetics and network analysis to characterize genes related to mouse weight. *PloS Genetics*, 2(8):e130, 2006.

[7] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning*. Springer, New York, 2001.

[8] Leonard Kaufman and Peter Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, Inc., New York, 1990.

[9] M. Oldham, S. Horvath, and D. Geschwind. Conservation and evolution of gene co-expression networks in human and chimpanzee brains. *Proc. Natl. Acad. Sci. USA*, 103(47):17973–17978, 2006.

[10] E. Ravasz, A. Somera, D. Mongru, Z. Oltvai, and A. Barabási. Hierarchical organization of modularity in metabolic networks. *Science*, 297(5586):1551–1555, 2002.

[11] B. Zhang and S. Horvath. A general framework for weighted gene co-expression network analysis. *Statistical Applications in Genetics and Molecular Biology*, 4(1):Article 17, 2005.