Optimal Pruning of Hierarchical Clustering Dendrograms

Jiacheng Ge *[1] and Robert Tibshirani[2]

[1]Department of Statistics, Stanford University, Stanford, CA 94305
[2]Departments of Biomedical Data Science and of Statistics, Stanford University, Stanford, CA 94305

July 14, 2025

Abstract

Keywords: unsupervised learning, clustering, dendrogram, pruning

Hierarchical clustering is a popular method for identifying distinct groups in a dataset. The most commonly used method for pruning a dendrogram is via a single horizontal cut. In this paper, we propose an "optimal pruning" method. We prove its superiority over horizontal pruning and provide some examples illustrating how the two methods can behave quite differently. Additionally, we compare our approach to dynamic programming. Furthermore, we discuss the selection of the optimal number of clusters.

1 Introduction

Clustering is a widely used technique in machine learning for grouping similar data points together in an unsupervised setting. It has applications in various areas such as genetics, genomics, and marketing analysis. Clustering methods can be categorized into two broad types: partitioning and hierarchical methods.

Partitioning methods, such as K-means clustering (Lloyd) [7], divide the data into a fixed number of clusters. This is achieved by minimizing the distance between each data point and their assigned cluster centroids. Despite its popularity due to its efficiency and simplicity, K-means clustering necessitates the prior specification of the number of clusters, which can be difficult in practice. Additionally, the membership of k clusters and $k-1$ clusters may not be nested, resulting in difficulty in interpreting the results and reusing clustering outcomes. As a result, the model must be retrained each time a different k value is attempted, which can be computationally intensive.

Hierarchical clustering (Ward) [13] is another fundamental technique for identifying distinct groups in data. There are two main types of hierarchical clustering strategies: agglomerative (bottom-up) and divisive (top-down). Both strategies use a greedy approach and we focus on agglomerative clustering in this paper. One benefit of hierarchical clustering is that any legitimate distance metric can be applied. Common distance metrics include average linkage, single linkage,

*Corresponding Author, E-mail: kevinge1@alumni.stanford.edu

and complete linkage. The hierarchy of clusters is usually presented in a dendrogram with desired clusters as its branches. The height of a node is proportional to its within-cluster dispersion. "Pruning" the dendrogram involves the removal of some of its lower branches.

The most popular dendrogram pruning method is the horizontal cut: we merge the leaves into several clusters either by specifying the desired number of clusters or the cut height. Every contiguous branch of objects below the cut is grouped together. This technique is a straightforward, simple method with many appealing qualities. Recall that a dendrogram is constructed from the bottom up, by merging the closest pair of nodes at each stage, and it is displayed with the height of each node equal to the distance at which the merge occurs. From this, one might expect that the horizontal cut yields the tightest set of clusters for a given number of leaves. However, this turns out not to be the case, as we show later.

Dynamic Tree Cut (Langfelder et al.) [6] is a dendrogram pruning method based on investigating the dendrogram's branch shapes. It has two variants: "Dynamic Tree" cut and "Dynamic Hybrid" cut. The Dynamic Tree cut is a top-down approach that involves adaptively breaking down and recombining clusters until a stable number of clusters is reached. The Dynamic Hybrid cut, on the other hand, is a bottom-up process that involves identifying branches that meet certain criteria and then assigning unassigned objects to the clusters found in the first step if they are sufficiently close. Dynamic tree cut is flexible and has been found to work well on bioinformatic tasks. However, it is computationally intensive and hard to interpret. Also, it is still unclear how to select the height and shape parameters and how to estimate the number of clusters.

Dynamic programming (DP) (Bellman) [1], is a technique that is used in both mathematical optimization and computer programming to solve complex problems by breaking them down into simpler sub-problems and solving them recursively. Marti utilizes DP for dendrogram pruning [8]. To optimally cut a tree into k clusters, it equivalently determines the optimal number of groups in the left and right branches of the root (k_l and k_r, respectively), with the condition that $k_l + k_r = k$. This process is then repeated for each branch, recursively until it reaches the leaves of the dendrogram or all remaining nodes fall into one cluster. An important drawback of this method is that it does not necessarily produce a nested sequence of subtrees.

In Section 2, we propose "optimal pruning". In section 3, we prove theoretically its advantage over the horizontal pruning method and provide some examples, including simulated examples with and without clusters, and some popular real datasets, illustrating how the two methods behave differently. In Section 3.5, we embed the two pruning methods into a classification task using a real dataset and compare their performances. In section 4, we compare our method with DP. In Section 5, we combine the optimal pruning sequence with the Gap test to estimate the ideal number of clusters. Finally, Section 6 contains some discussion.

2 Our proposed method

2.1 Overview

The goal is to identify a sequence of the tightest clusters within a given dendrogram for each fixed number of groups. Our proposed pruning method operates independently of the initial construction of the hierarchical clustering tree, regardless of the linkage method (e.g., single, average, complete). Furthermore, our method does not depend on the specific within-cluster dispersion loss function used in the pruning process. Instead, it focuses on optimizing the post-hoc pruning process based

on a given dendrogram and a dispersion loss to achieve the tightest possible clusters for each fixed number of groups.

2.2 Set up

Let T be a dendrogram and t be a node in T. If there is no further splitting at a node, we call it a terminal node or a leaf. Let $\tilde{T}$ denote all terminal nodes in T.

Define $D(x_i, x_{i'})$ as the distance between points x_i and $x_{i'}$. In this paper, we adopt the squared distance as the dispersion loss unless otherwise stated, i.e.

$$D(x_i, x_{i'}) = \|x_i - x_{i'}\|_2^2.$$

This is a natural measure of dispersion, but our method is compatible with other loss functions, such as the L_1-norm, i.e. $D(x_i, x_{i'}) = \|x_i - x_{i'}\|_1$,.

In this paper, the loss function $R(t)$ for a node t is defined as the sum of dispersion losses, i.e.

$$R(t) = \sum_{x_i, x_{i'} \in t} D(x_i, x_{i'})$$

unless specified otherwise. Alternatively, users may choose the average dispersion loss, i.e. $R(t) = \frac{1}{\binom{n_t}{2}} \sum_{x_i, x_{i'} \in t} D(x_i, x_{i'})$, depending on the analysis requirements. The sum of dissimilarities, an extensive measure, penalizes larger clusters, discouraging their formation. The average dissimilarity, an intensive measure, does not penalize cluster size and is suitable when larger clusters are desirable.

The loss function R for T is defined as the total dispersion loss across all leaves, i.e.

$$R(T) = \sum_{t \in \tilde{T}} R(t)$$

Let T_{max} denote an unpruned dendrogram with only one observation in each terminal node. If T is a subtree of T_{max}, we denote $T \subseteq T_{max}$. A subtree refers to a dendrogram derived from a tree by collapsing some of its nodes into a single node. Our goal is to find the "optimal" dendrogram $T \subseteq T_{max}$. However, $R(T)$ alone is not a useful metric for finding the "optimal" subtree because it always prefers larger trees. It is easy to see that $R(T) = 0$ when each leaf only contains one case. So, we introduce a complexity penalty to control the balance between dispersion and tree size. For any subtree $T \subseteq T_{max}$, we define its complexity as $|\tilde{T}|$, the number of terminal nodes in T. Let $\alpha \geqslant 0$ be a real number and called the complexity parameter. We define the cost-complexity measure $R_\alpha(T)$ as:

$$R_\alpha(T) = R(T) + \alpha|\tilde{T}|.$$

Our approach is twofold: (a) for each α, find the subtree that minimizes $R_\alpha(T)$ and then (b) estimate the best value $\hat{\alpha}$ and hence the optimal subtree $T(\hat{\alpha})$ using a method for estimating the best number of clusters. In Section 5 below we illustrate this using the Gap Test, but emphasize that other methods for estimating the number of clusters could be used.

For (a), our goal is to find subtree $T(\alpha) \subseteq T_{max}$ which minimizes $R_\alpha(T)$; $T(\alpha)$ is called the minimizing subtree for α, i.e.

$$R_\alpha(T(\alpha)) = \min_{T \subseteq T_{max}} R_\alpha(T).$$

The smallest such $T(\alpha)$, in terms of complexity, is defined as the smallest minimizing subtree for the complexity parameter α.

2.3 How Optimal pruning works

Our approach follows closely the weakest link pruning method used in classification and regression trees (CART) [2], a supervised learning method. In our unsupervised clustering problem, the only change we make here is in the loss function, where we replace misclassification loss with the within-cluster dispersion loss. To find the smallest minimizing tree, we adopt weakest link pruning: **we successively collapse the internal node that causes the smallest per-node rise in $R(T)$ until a single-node (root) dendrogram is produced**. Given a dendrogram of n leaves, choosing to cut the internal node with the smallest per-node increase in $R(T)$ is equivalent to finding the subtree with fewer than n leaves which has the smallest value of R. By performing weakest link pruning, we generate a sequence of minimal cost-complexity subtrees. Notably, the subtrees are decreasing in their complexity, but their number of leaves may not be consecutive. In other words, some tree sizes could be skipped. If multiple branches incur an identical per-node rise in $R(T)$, we choose the one resulting in the smallest tree complexity. If there is a tie, any one of them can be collapsed first. The resulting cost-complexity path remains the same up to label-swapping. In practice, the software picks the first one.

On pages 66-71 of CART book [2], the following facts are established:

1. For every value of α, there exists a (unique) smallest minimizing subtree.
2. The sequence of minimal cost-complexity trees contains the smallest minimizing subtree.

The proofs of these claims follow directly from the CART results [2].

2.4 A toy example of optimal pruning

To illustrate how optimal pruning works in practice, we provide an example using a dendrogram of four numbers. Although in the paper we adopt $R(t)$ as the sum of dispersion losses and $D(x_i, x_{i'})$ as the squared distance, here we present a more nuanced example where $R(t)$ is defined as the average of dispersion losses and $D(x_i, x_{i'})$ is the L_1-distance (i.e. absolute difference, $|x_i - x_{i'}|$). This example highlights the mechanics of the pruning process, and users can easily adapt the logic to the case of sum of dispersion losses and squared distance on their own.

Consider a dendrogram with four leaves corresponding to the values 0, 2, 10, and 13. The initial clustering process forms a hierarchical structure, and we evaluate three potential collapse points labeled A, B, and C, as shown in Figure 1. Position A corresponds to collapsing 0 and 2, position B corresponds to collapsing 10 and 13, and position C corresponds to collapsing all 0, 2, 10, and 13. We walk through the pruning process in two stages, calculating the average loss and the loss rise per node at each step.

- Stage 1: At the start, the total average loss is 0 since no prunings have occurred. We evaluate the options of collapsing A, B, or C.
  - Collapsing A (0 and 2): The average loss is $\frac{2-0}{1} = 2$, with a loss rise per node of $\frac{2}{2-1} = 2$.
  - Collapsing B (10 and 13): The average loss is $\frac{13-10}{1} = 3$, with a loss rise per node of $\frac{3}{2-1} = 3$.
  - Collapsing C (0, 2, 10, and 13): The average loss is computed over all pairwise differences between elements in the two clusters: $(13-0)+(13-2)+(10-0)+(10-2)+(2-0)+$

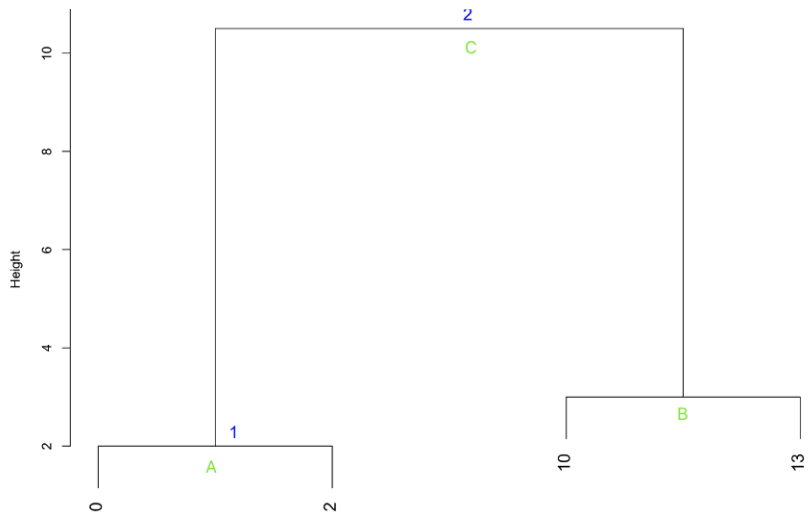


Figure 1: A dendrogram of four numbers (0, 2, 10, 13).

$(13 - 10) = 47$, divided by the number of pairs $\binom{4}{2} = 6$, so $\frac{47}{6}$. The loss rise per node is $\frac{\frac{47}{6}}{4-1} = \frac{47}{18} \approx 2.61$.

Since $2 < \frac{47}{18} < 3$, we choose to merge at A, as it results in the smallest per-node rise in $R(T)$.

- Stage 2: After collapsing A, the total average loss is now 2. We compare merging at B versus C.
  - Collapsing B (10 and 13): The average loss is $\frac{13-10}{1} = 3$, with a loss rise per node of $\frac{3}{2-1} = 3$.
  - Collapsing C (cluster(0, 2), 10, and 13): The average loss is still $\frac{47}{6}$, but we must deduct the loss from the previous merge at A (which is 2) to compute the net effect: $\frac{47}{6} - 2 = \frac{35}{6}$. The loss rise per node is $\frac{\frac{35}{6}}{3-1} = \frac{35}{12} \approx 2.92$.

  Since $\frac{35}{12} < 3$, we choose to merge at C.

This example demonstrates the key steps in optimal pruning: at each stage, we select the merge that minimizes the per-node rise in $R(T)$, adjusting for previous merges by deducting their associated losses. One needs to be more cautious when using the average loss: the number of pairwise comparisons, given by the binomial coefficient $\binom{n_t}{2}$, can change before and after nodes are collapsed.

2.5 Time complexity

Let n be the number of data points, v be the number of vertices and e be the number of edges in the dendrogram. Given a full dendrogram (an `hclust` object produced by `hclust` package [9] in

R), we need $O(n^2)$ to calculate the pair-wise distances, $O(n)$ to generate the offspring info in our R function, and $O(n*(v+e))$ to decide the internal node to merge. Here v and e are n and $n-1$. So, the total time complexity to generate the sequence of minimal cost-complexity trees is $O(n^2)$. If we specifically want k clusters instead of the whole sequence, then the time complexity to decide the optimal cut is $O((n-k)*(v+e))$.

3 Optimal pruning versus horizontal pruning

3.1 Proposition: Theoretical performance

First, we claim that our optimal pruning always yields a total dispersion loss equal to or less than the horizontal cut for a given number of terminal nodes. We will prove this by contradiction. Let T_{max} be a dendrogram and $\{T_{s_i}\}$ be its sequence of minimal cost-complexity subtrees. Assume that there exists a horizontally pruned dendrogram $T_h \subseteq T_{max}$ that is not in $\{T_{s_i}\}$ and the dendrogram has a smaller loss than any of the optimal pruned subtrees, i.e. $R(T_1) \leqslant \min_{s_i} R(T_{s_i})$. The horizontal cut is denoted by h_1. All the nodes equal to or below h_1 are already collapsed while all nodes above are not. Denote the pruned dendrogram before h_1 happening by $T_{h'}$. Hence:

- If all the nodes above the h_1 have a larger per-node rise in $R(T_{h'})$ than all the nodes equal to or below h_1 when being collapsed, then by the definition of our optimal pruning, T_1 must be in $\{T_{s_i}\}$.
- If there exists a node t_1 above h_1 and a node t_2 equal to or below h_1 such that per-node rise of $R(T_{h'})$ when cutting t_1 is smaller than cutting t_2, then we can reduce $R(T_h)$ by collapsing t_1 and expanding t_2.

In both cases, there's a contradiction to our assumption. Therefore, the horizontal cut solution is either in the weakest link merge sequence or the resultant subtree has a higher loss, implying that our optimal pruning is never worse than horizontal cut for a fixed number of leaves.

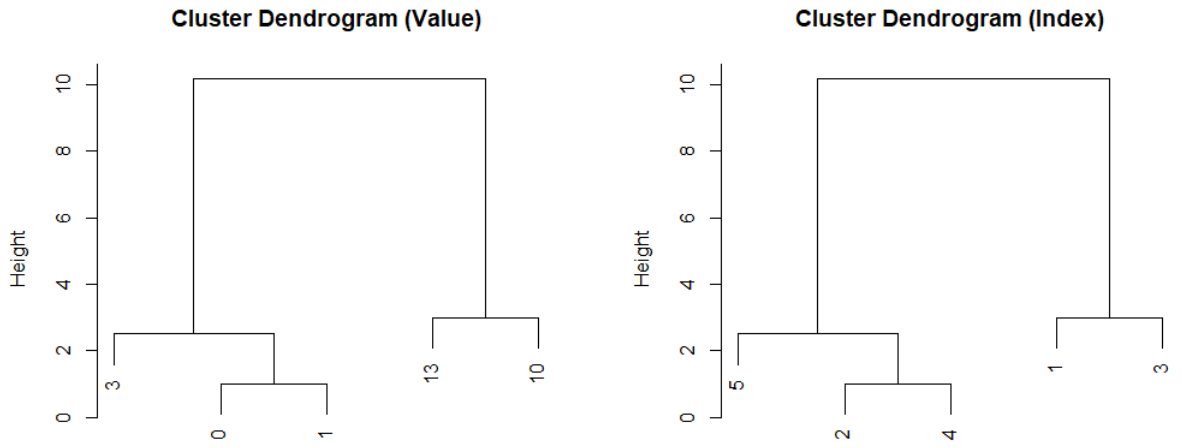
Second, recall that a sequence of minimal cost-complexity subtrees may skip some tree sizes. Without loss of generality, let's assume the tree complexities in the optimal subtree sequence are $(n, n-1, ..., k+c+1, k+c, ..., k, ..., 1)$. When we have a subtree of $k+c+1$ nodes, we have the option to prune it into k+c, $k+c-1, ..., k$ leaves. Since we choose the next merge by comparing the smallest per-node increase in R, then the best subtree with $k+c-1, k+c-2, ..., k+1$ leaves must have bigger losses than the best subtree with $k+c$ leaves or k leaves. By transitivity, the horizontal pruned trees with $k+c-1, k+c-2, ..., k+1$ leaves will have even larger losses than the best minimal cost-complexity subtree with $k+c$ leaves or k leaves.

To summarize, given a horizontally pruned dendrogram $T_h \subseteq T_{max}$, we can always find a T in the sequence of minimal cost-complexity subtrees of T_{max} such that $R(T_h) \geqslant R(T)$. This implies that horizontal pruning can never do better than our optimal pruning.

3.2 A simple example

Here we illustrate that the optimal pruning and horizontal cuts can behave differently. Consider a dataset of five (scalar) data points: 13, 0, 10, 1, 3, indexed by 1, 2, 3, 4, 5. We created a dendrogram for the dataset via average linkage, which is shown in Figure 2. The left plot labels the leaves by their data values and the right one shows the node numbers 1–5.

Given that some datasets contain decimal points or high-dimensional data which can clutter the display, we opted to label all subsequent figures in the paper by index to maintain consistency and clarity.



(a) *Leaves labelled by their data values* (b) *Leaves labelled by the node numbers 1–5*

Figure 2: *Cluster dendrograms for a simple example*

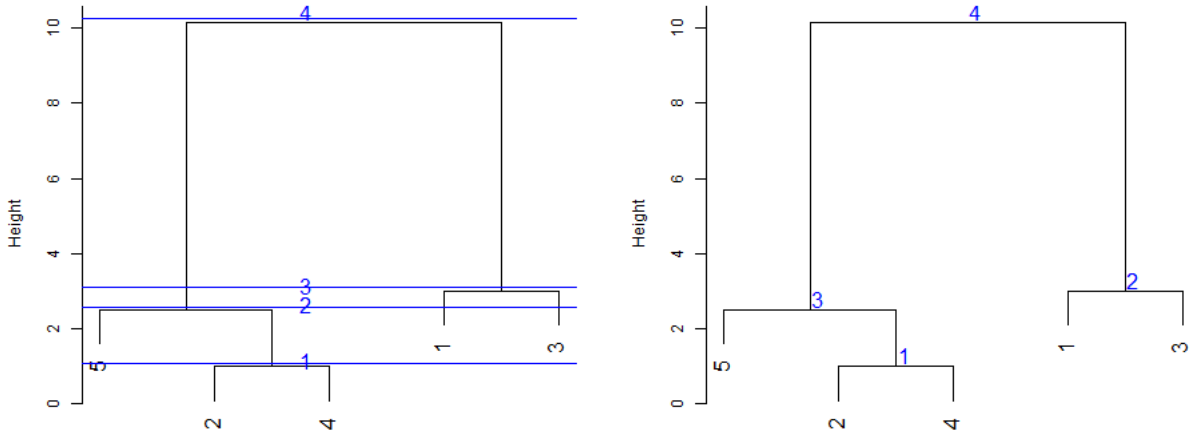
The left plot of Figure 3 shows the cuts in the horizontal pruning process using the `cutree` function in the R language package `stats` [9]. This process collapsed the lowest internal node at each step from bottom to top. The right plot labels the merges in the optimal pruning process. Notice that the cuts were not increasing in height at each step. (It is worth mentioning that it's also common for some internal nodes to be skipped.)

Figure 4 details the subtrees produced by two different pruning methods. Notably, resulting subtrees with 5, 4, 2, and 1 terminal nodes were the same for both methods. However, optimal pruning merged node 1 and node 3 before merging node (2 4) and node 5. And its resulting within-cluster sums of squares (WSS) is 10 when there were 3 leaves, smaller than the value 14 for the horizontal cuts.

3.3 Experiments with simulated datasets

3.3.1 Random datasets with no clusters

We randomly generated 200 datasets X_i with number of observations n_i between 30 and 100, and number of features p_i between 1 and 50. We chose n_i and p_i uniformly sampled and $x_{ij} \overset{\text{iid}}{\sim} \mathcal{N}(0, I_{p_i})$. Then, we ran the two pruning methods, horizontal and our optimal pruning, on the 200 sets of data for comparison. Figure 15 shows the log of WSS when the dendrogram was pruned down to between 2 and 25 terminal nodes. Each colored circle represents the mean of the (log) WSS and the error bar represents one standard error. For optimal pruning, if the subtree with a certain number of leaves was not in the sequence of optimal pruned subtrees, we increased the



(a) *Cuts in horizontal pruning* (b) *Cuts in optimal pruning*

Figure 3: *The cuts in two different pruning processes*

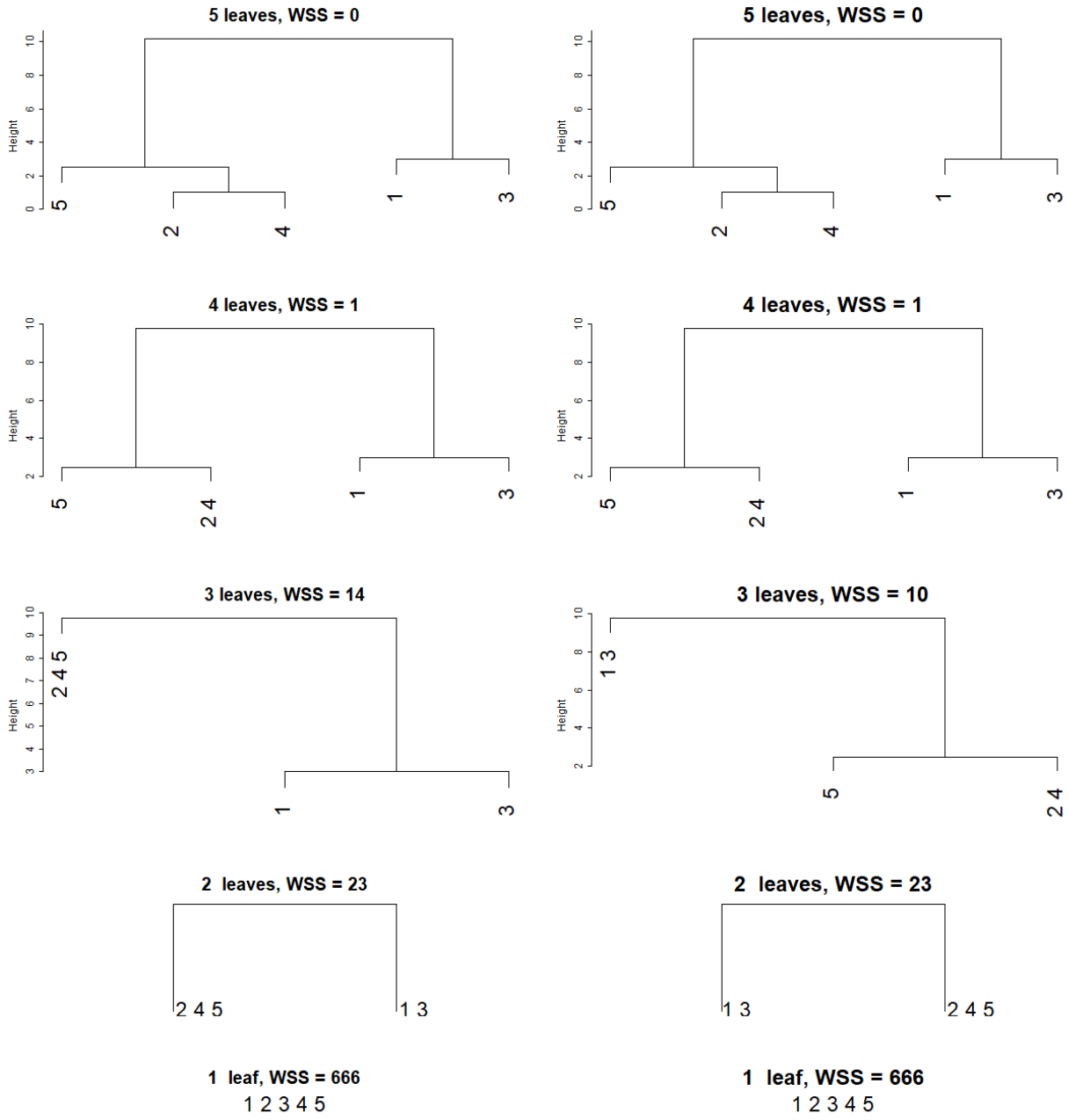
number of leaves to the closest available number. For the `optimal_skip` pruning, if the subtree with a certain number of leaves was not in the sequence of optimal pruned subtrees, we skipped this subtree, thus the loss is defined as NA.

We see that the optimal cut has consistently smaller WSS, sometimes by a large margin. For some specific numbers of clusters, the variance of the `optimal_skip` WSS is higher because some number of clusters are skipped and the amounts of losses are reduced.

Figure 5 shows the relative improvement in the loss for our optimal pruning relative to horizontal cut. In this plot, the relative differences were only calculated when the subtree with the clusters was a member of the sequence of minimizing subtrees. The loss reduction is quite remarkable, and the median of loss reduction (after removing NAs due to skipping numbers of clusters) was 0.51. As demonstrated by both plots, optimal pruning shows better performance than the horizontal cut when the data is completely random.

3.3.2 Random datasets with equal-sized well-separated clusters

We randomly generated 200 datasets with the number of observations n_i between 30 and 100, the number of features p_i between 1 and 50, and the number of clusters c_i between 3 and 15, all uniformly sampled. We generated $x_{ij} \overset{\text{iid}}{\sim} \mathcal{N}(0, I_{p_i})$ and then shifted the data points in each cluster by a vector of length p_i whose components repeat an integer uniformly sampled from 1 to c_i without replacement, scaled by a constant factor. The sizes of all clusters were equal, except for the last cluster, which was adjusted to account for indivisibility. A small amount of noise was added with a standard deviation of 1. The 2D projection of an example dataset produced by the `Rtsne` R package [5] is illustrated in Figure 16. The clustered data are well separated with a small amount of overlap.



(a) *Horizontal pruning process* (b) *Optimal pruning process*

Figure 4: *The sequence of subtrees from two different pruning strategies.*

We applied the horizontal and optimal pruning methods to these 200 datasets. Figure 17 shows the log of WSS when the dendrogram was pruned down to between 2 and 25 terminal nodes. The colored circles are the mean of the log WSS, and the error bar represents one standard error. For optimal pruning, if the subtree with a certain number of leaves was not in the sequence of

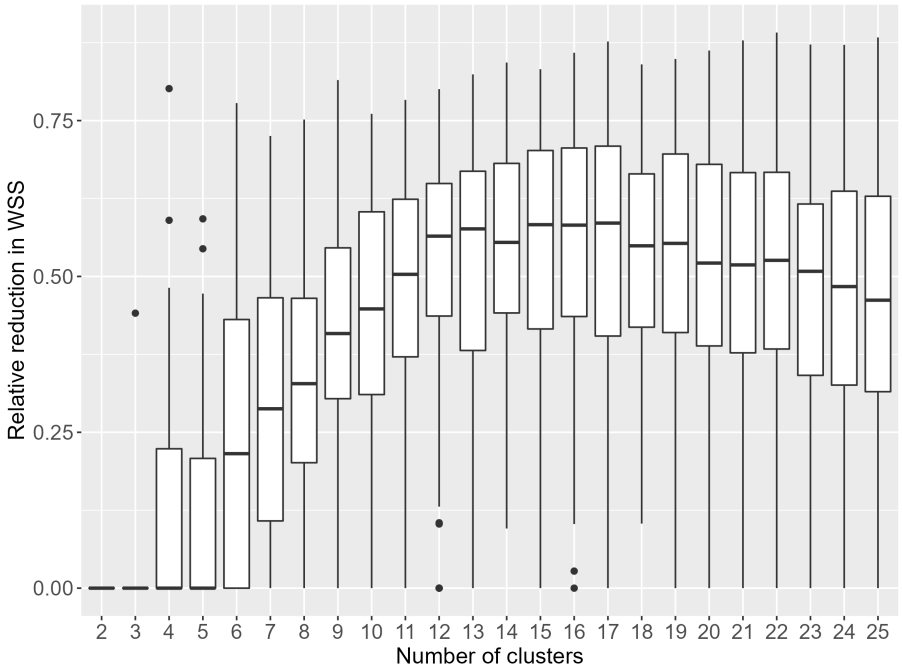


Figure 5: *Relative reduction in WSS, for optimal pruning versus horizontal cut on 200 random datasets with no clusters.*

optimal pruned subtrees, we increased the number of leaves to the closest available number. For the `optimal_skip` pruning, if the subtree with a certain number of leaves was not in the sequence of optimal pruned subtrees, we skipped this subtree and thus the loss was set to NA.

Figure 6 illustrates the relative reduction in WSS for optimal pruning compared to the horizontal cut, calculated only when the subtree with the specified number of clusters was part of the sequence of minimizing subtrees. Consistent with the no-cluster setting, optimal pruning reduces WSS, though the improvement is less pronounced, with a median relative reduction in loss of 0.085 (after removing NAs due to skipped numbers of clusters). To further evaluate the clustering quality, we compared both methods to the simulated ground truth using the Adjusted Rand Index (ARI) [3], setting the number of clusters to the true number ($k = c_i$). As shown in Figure 18, both methods achieve high mean ARI values (0.969 for optimal and 0.966 for horizontal), indicating strong agreement with the ground truth. Unlike WSS, there is no theoretical guarantee that one method will consistently outperform the other in ARI. In this scenario, optimal pruning slightly outperforms horizontal pruning in ARI, further demonstrating that it achieves WSS reduction without sacrificing clustering quality in the equal-sized well-separated case.

3.3.3 Random datasets with unequal-sized well-separated clusters

Following the same setting as previously, but unlike equal-sized cluster generation, this simulation assigns random cluster sizes with the biggest cluster being up to ten times larger than the smallest cluster. The 2D projection of an example dataset produced by the `Rtsne` R package [5] is illustrated in Figure 19. The clustered data are well-separated, but the varying cluster sizes introduce additional complexity compared to the equal-sized scenario. Figure 20 shows the

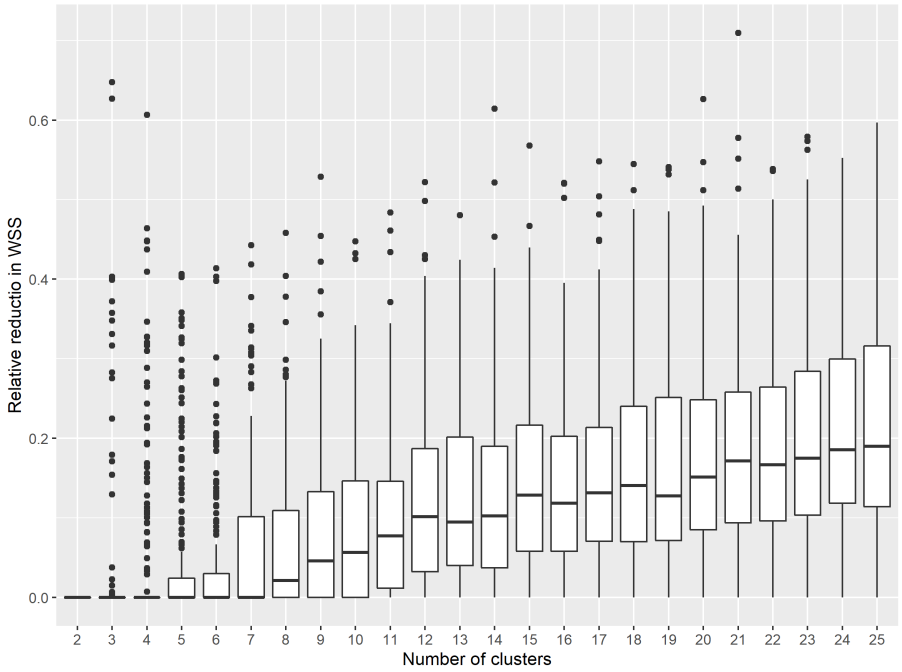


Figure 6: *Relative reduction in WSS, for optimal pruning versus horizontal cut on 200 random datasets with equal-sized well-separated clusters.*

log WSS when the dendrogram was pruned down to between 2 and 25 terminal nodes. Figure 7 presents the relative reduction in WSS for our optimal pruning compared to the horizontal cut. Optimal pruning demonstrates a greater improvement over the horizontal cut than in the equal-sized well-separated scenario, with a median relative reduction in loss of 0.40 (after removing NAs due to skipped numbers of clusters). To assess clustering quality against the ground truth, we also computed the ARI with the number of clusters set to the true number ($k = c_i$). As shown in Figure 21, horizontal pruning achieves a higher mean ARI (0.972) compared to optimal pruning (0.905). Optimal pruning produces the tightest clustering; however, when true cluster sizes vary significantly, this grouping may not align with the ground truth, resulting in a lower ARI. Despite this, optimal pruning's primary advantage lies in its substantial within-cluster dispersion reduction, making it well-suited for applications prioritizing dissimilarity reduction.

3.3.4 Well-separated clusters of unequal sizes under average-based loss

In the same setting as previously, this simulation replaces the $R(t)$ with the average dispersion loss. Figure 8 presents the relative reduction in the sum of average distances for our optimal pruning compared to the horizontal cut, which is similar to the previous results. The median relative reduction in total average dispersion loss (after removing NAs due to skipping numbers of clusters) was 0.132.

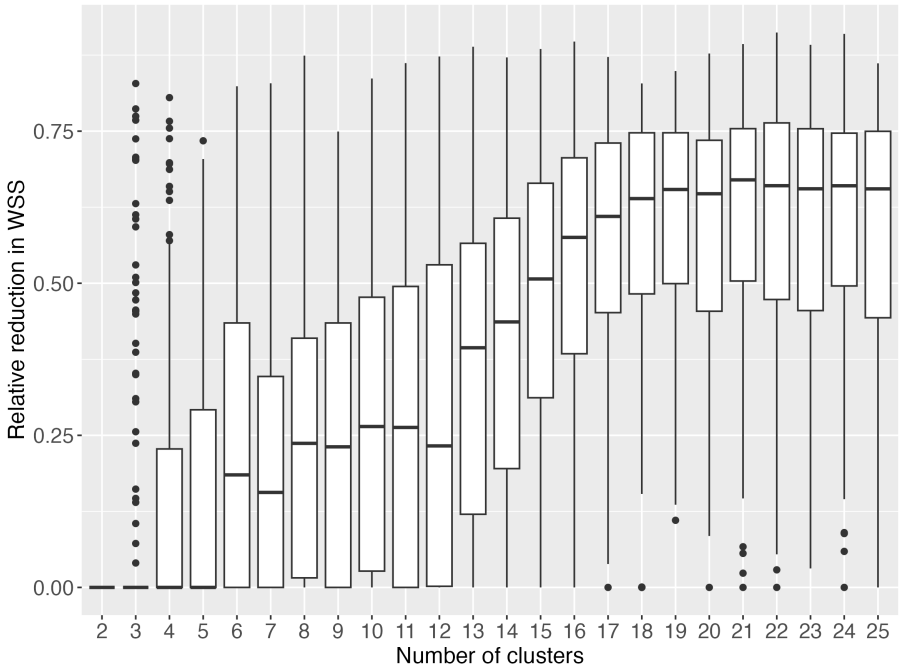


Figure 7: *Relative reduction in WSS, for optimal pruning versus horizontal cut on 200 random datasets with unequal-sized well-separated clusters.*

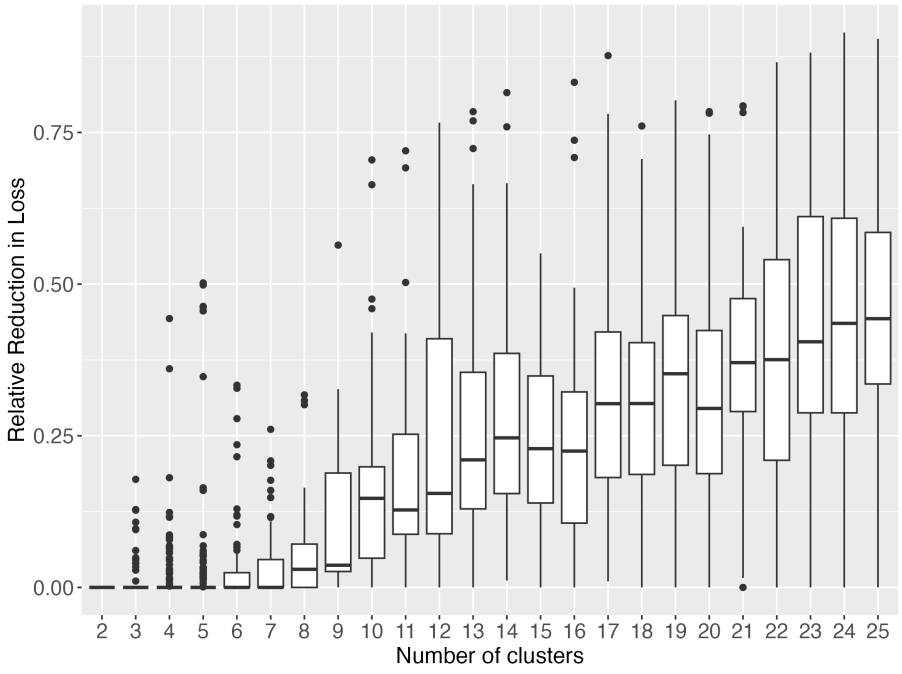


Figure 8: *Relative reduction in the sum of average distances, for optimal pruning versus horizontal cut on 200 random datasets with unequal-sized well-separated clusters.*

3.4 Some real data examples

Here we applied the pruning methods to six popular datasets from the book "Introduction to Statistical Learning", Second Edition (ISLR2) [4]: Boston, Khan(train), Khan (test), NCI60, Portfolio, and USArrests. Details can be found in table 1.

Dataset	Number of observations	Number of features
Boston	506	13
Khan (train)	63	2308
Khan (test)	20	2308
NCI60	64	6831
Portfolio	100	2
USArrests	50	4

Table 1: *Six real datasets from the ISLR2 text.*

We ran the two pruning methods on the six real datasets for comparison. Combining results for all six datasets together, Figure 22 shows the log of WSS when each dendrogram was pruned to between 2 and 19 terminal nodes. The colored circles represent the mean of log WSS and the error bars represent one standard error. If the subtree with a certain number of leaves was not in the sequence of optimal pruned subtrees, we increased the number of leaves to the closest available number. Figure 9 shows the relative reduction in loss when we replaced the horizontal cut with our optimal cut. In this plot, the relative differences were only calculated when the subtree with the input number of clusters was a member of the sequence of minimizing subtrees. The reduction in loss for optimal pruning versus horizontal cut was quite substantial.

3.5 NCI60 classification Problem

The NCI60 dataset comes from cancer cell line microarrays, consisting of 6,830 gene expression measurements on 64 cancer cell lines. There are 14 different cancer types in this dataset: "BREAST", "CNS", "COLON", "K562A-repro", "K562B-repro", "LEUKEMIA", "MCF7A-repro", "MCF7D-repro", "MELANOMA", "NSCLC", "OVARIAN", "PROSTATE", "RENAL", "UNKNOWN". First ignoring the cancer labels, we built a dendrogram using average linkage, and performed optimal and horizontal pruning methods, pruning down to 14 terminal nodes in each case.

Figure 23 shows the sequence of horizontal cuts. We label every 5 cuts with transparent thin blue lines and the final cut with the solid thicker blue line. Figure 24 shows the sequence of optimal cuts. We label cuts with transparent blue numbers and the last cut with the bigger solid blue "46".

The final dendrogram produced by horizontal pruning is shown in figure 25 and the optimal pruned one is shown in figure 26. The WSS for the horizontal pruned tree was $2,544,265.78$ while the final WSS for the optimal pruned tree was $915,484.12$, around only one-third of the value from the horizontal counterpart. The median relative reduction in WSS (after removing NAs due to skipping numbers of clusters) was 0.36. As we can see from Figure 10, optimal cut dominates

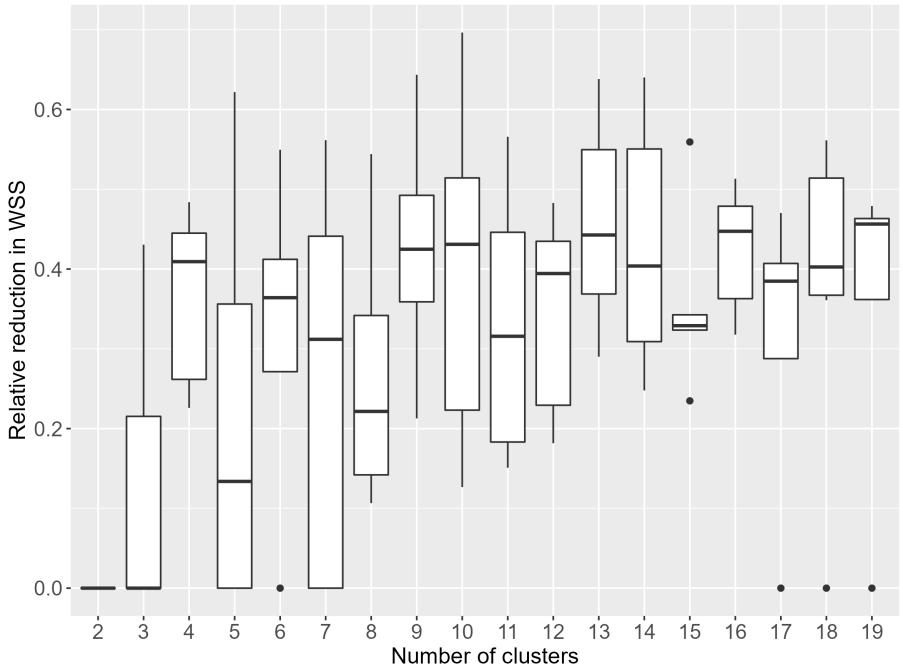
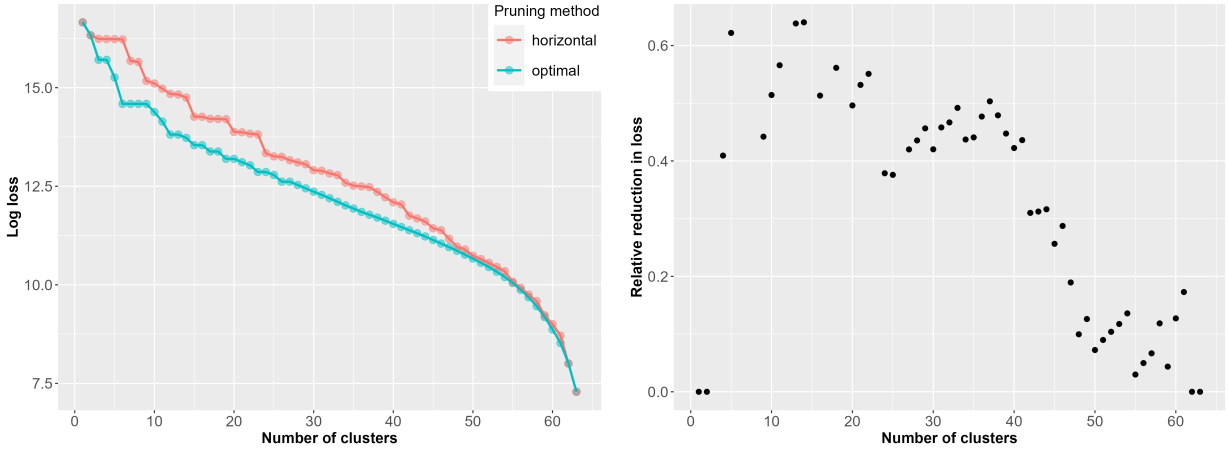


Figure 9: *Relative reduction in WSS, for optimal pruning versus horizontal cut on 6 real datasets*

horizontal cut for all numbers of clusters and the discrepancy is more significant when the number of clusters is between 5 and 40.

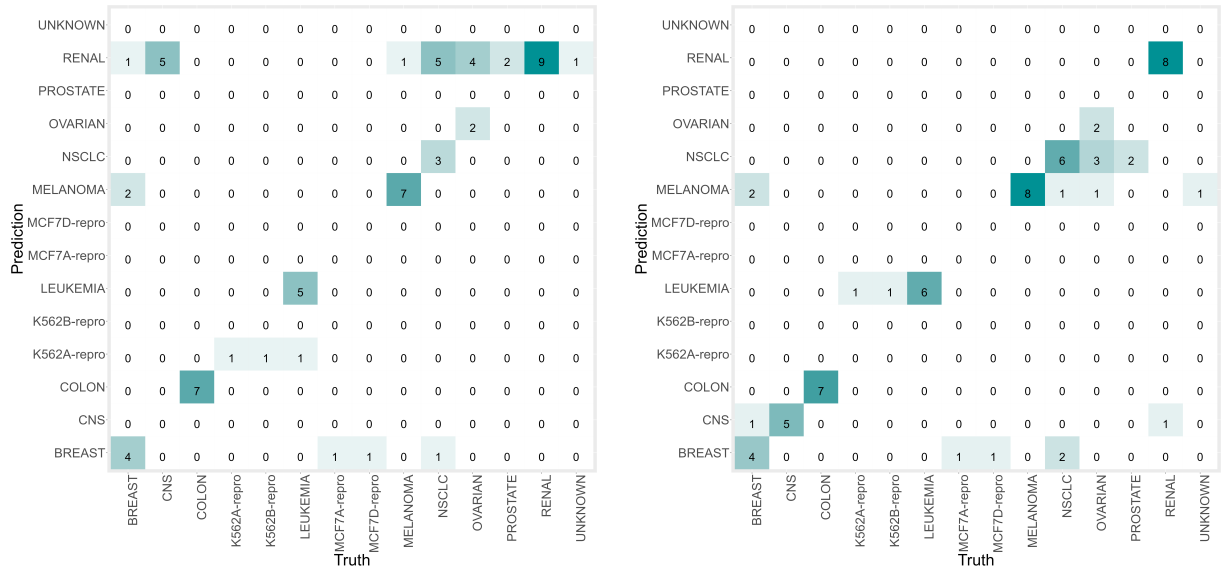


(a) *WSS comparison* (b) *Relative reduction in WSS*

Figure 10: *Comparison of two pruning methods NCI-60*

Finally, for both pruning methods, we focused on the resulting trees with 14 terminal nodes (the number of cancer classes). We used a majority vote of the cell lines in each cluster to determine a predicted cancer label for that cluster. We then compared these predictions to the true sample labels. The confusion matrices are shown in Figure 11. The classification error rate was 0.41 for the

horizontal pruned tree, and 0.28 for the optimal pruned tree. The results suggest that our optimal pruning has yielded a more meaningful grouping of the observations.



(a) *Using horizontal pruning* (b) *Using optimal pruning*

Figure 11: *Confusion matrices for classifying 14 cancer cell lines*

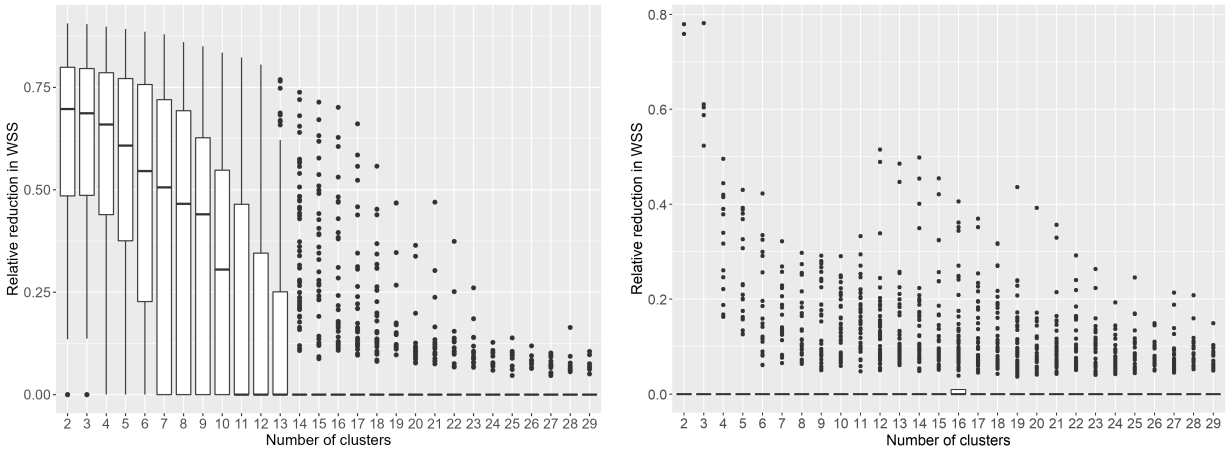
4 Optimal pruning versus dynamic programming

Dynamic programming exhaustively searches for all possible combinations of branch cuts. We want to compare our optimal pruning with DP. We ran two simulations with the same setting as in section 3 – 200 random datasets with no clusters and 200 random datasets with well-separated clusters. The two variants of optimal pruning – "optimal_pruning" and "optimal_skip" pruning are introduced in section 3.

In both simulations, the differences in WSS between optimal_skip pruning and dynamic programming were 0. Figure 12 shows the relative reduction in WSS for "optimal_pruning" relative to DP. We can see that even when a certain number of leaves was not in the sequence of optimal pruned subtrees and we increased the number of leaves to the closest available number, our optimal pruning still yielded a smaller loss.

In conclusion, when the subtree with the input number of clusters is a member of the sequence of minimizing subtrees, our optimal pruning is equally good to DP. Since DP does a thorough search, this suggests that our method can find the tightest clustering given a number of clusters. Even if the subtree with the input number of clusters is not in the sequence of minimizing subtrees, our method still provides a tighter clustering solution than DP. However, if one wants a specific number of distinct groups, then our method may not function well.

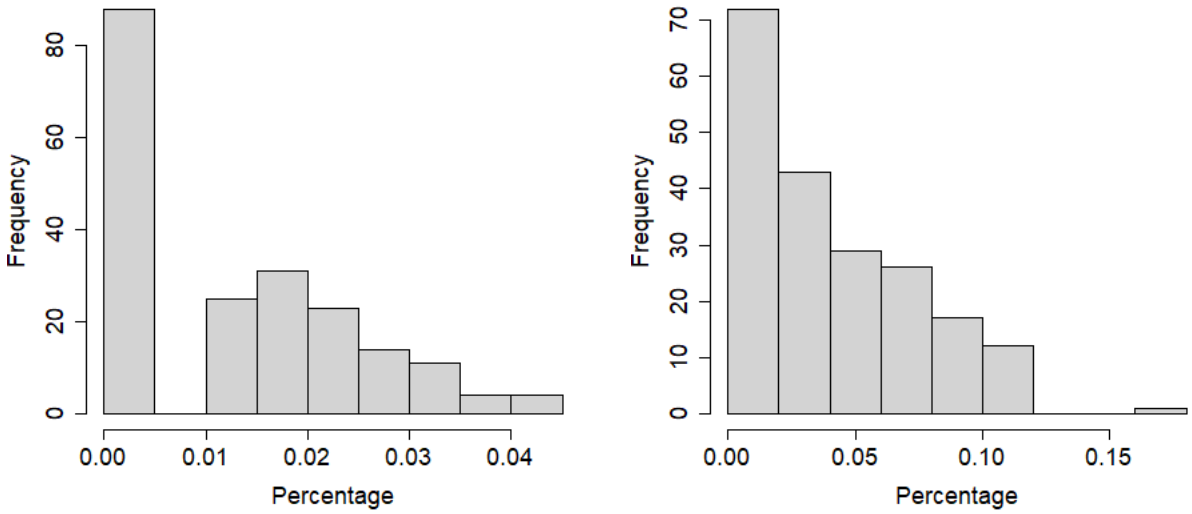
For the purpose of interpretability, a nested sequence of memberships may be preferred. By the nature of our optimal pruning, the optimal sequence of memberships is guaranteed to be nested. However, dynamic programming has no such property. Figure 13 shows the occurrences of unnested sequence of memberships produced by DP. In the completely random simulation above, 112 out of



(a) *200 random datasets with no clusters* (b) *200 random datasets with well-separated clusters*

Figure 12: *Relative reduction in WSS, for optimal pruning versus dynamic programming*

200 trials saw at least one unnested pair of consecutive memberships. Some trials contained around 4% of unnested pairs of consecutive memberships. In the random simulation with well-separated clusters above, 148 out of 200 trials saw at least one unnested pair of consecutive memberships. One trial contained around 17% of unnested pairs of consecutive memberships.



(a) *200 random datasets with no clusters* (b) *200 random datasets with well-separated clusters*

Figure 13: *Percentage of unnested pairs of consecutive memberships*

5 Choosing the best-sized subtree

With the sequence of optimal subtrees in hand, we can estimate the best-sized subtree among the sequence. This is equivalent to choosing the optimal value of the cost-complexity parameter α. In CART, this is done via cross-validation, but this approach is not available in the unsupervised setting. So instead we apply a method for choosing the optimal number of clusters. There are many such methods, with two popular ones being the silhouette test (Rousseeuw) [10] and the Gap test (Tibshirani et al.) [11]. For illustration, we use the Gap test here.

The Gap test compares the changes in within-cluster dispersion to what would be expected under a suitable reference null distribution using the output of any clustering algorithm, such as hierarchical clustering. More specifically, it compares the curve of log within-cluster dissimilarity against the number of clusters to the curve achieved from data uniformly distributed throughout a rectangular space. The optimal number of groups can be chosen based on many criteria, here we chose the place where the gap between the two curves is most significant.

We randomly generated 20 datasets X_i with the number of observations n_i between 20 and 30, the number of features p_i between 1 and 30, and the number of well-separated clusters equal to 4. We sampled both n_i and p_i uniformly. Then we used the `monte` function in the R package `fungible` [12] to generate the x_{ij}. The sizes of all clusters were equal, except for the last cluster which adjusted for indivisibility. Then we ran the Gap test with the optimal pruning method on the 20 datasets. If the subtree with a certain number of leaves was not in the sequence of optimal pruned subtrees, we increased the number of leaves to the closest available number. The Gap test chose the correct number of clusters (4) 18 out of 20 times. Figure 14 shows an example that the optimal number of clusters was 4. Of course, the Gap test (and other tests) will not perform as well when the clusters are not well separated.

6 Discussion

In this paper, we propose an optimal pruning method for dendrograms using a weakest-link approach. Given a dendrogram, this method generates a sequence of the tightest clusters for each fixed number of groups. This sequence can be used for exploratory analysis or combined with procedures such as the Gap test to estimate the optimal set of clusters. Through theoretical analysis and simulations across various scenarios, we compare our method with horizontal pruning and dynamic programming. Our results demonstrate the superiority of our approach. A public R language package implementing this method will soon be available at `https://github.com/kevinge006/Optimal-Pruning-of-Hierarchical-Clustering-Dendrograms`.

Acknowledgments. We thank Jacob Bien for helpful comments, and Tal Galili for help with the dendextend package R package. RT is funded by the National Institutes of Health (5R01 EB001988-16) and the National Science Foundation (19 DMS1208164)

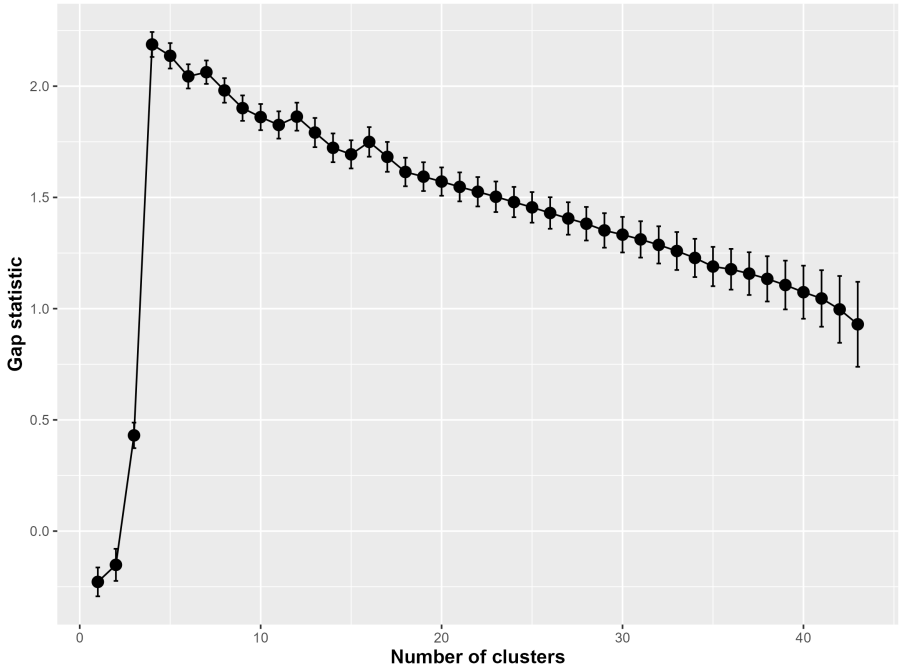


Figure 14: *A Gap test example of 44 data points and 25 dimensions. The estimated number of clusters is 4, the true number.*

References

[1] Bellman, R. (1957). *Dynamic Programming*. Dover Publications.

[2] Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and Regression Trees*. Taylor & Francis, Andover, England, UK.

[3] Hubert, L. and Arabie, P. (1985). Comparing partitions. *Journal of Classification*, 2(1):193–218.

[4] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning: with Applications in R*. Springer.

[5] Krijthe, J. H. (2015). *Rtsne: T-Distributed Stochastic Neighbor Embedding using Barnes-Hut Implementation*. R package version 0.16.

[6] Langfelder, P., Zhang, B., and Horvath, S. (2007). Defining clusters from a hierarchical cluster tree: the Dynamic Tree Cut package for R. *Bioinformatics*, 24(5):719–720.

[7] Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137.

[8] Marti, G. (2017). [clustering] how and where should you cut a dendrogram?

[9] R Core Team (2021). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

[10] Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65.

[11] Tibshirani, R., Walther, G., and Hastie, T. (2001). Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423.

[12] Waller, N. G. (2022). *fungible: Psychometric Functions from the Waller Lab.* version 2.2.1.

[13] Ward, J. H. (1963). Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58:236–244.

A Figures

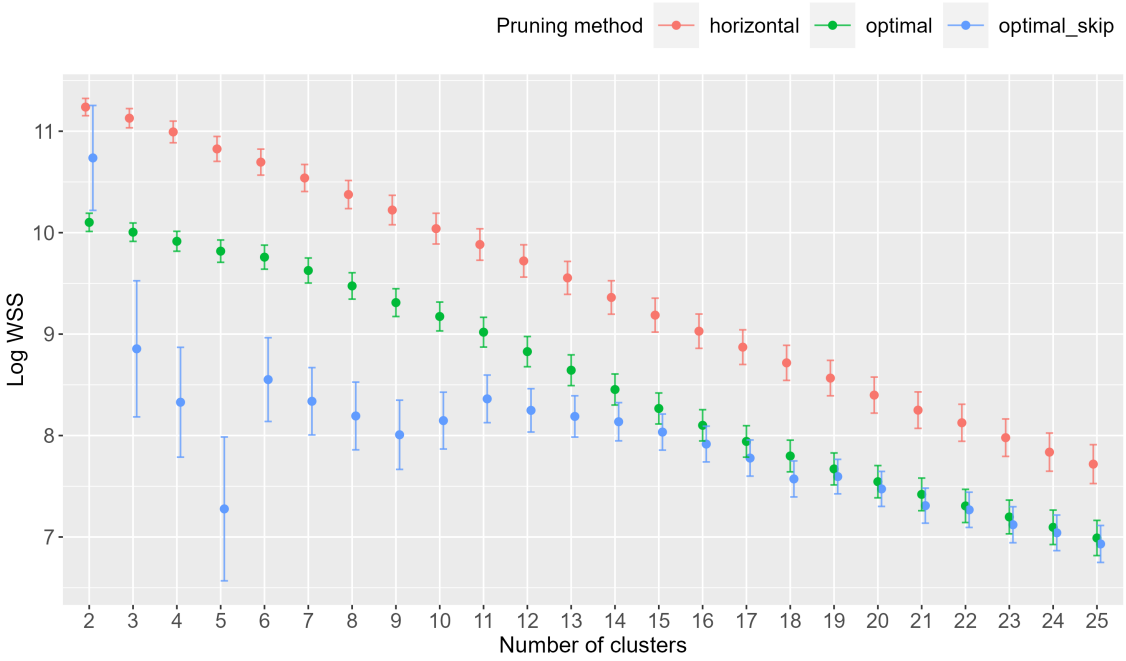


Figure 15: *Log WSS mean and SE comparison of the two pruning methods on 200 random datasets with no clusters. "Optimal pruning" increases the number of leaves to the closest available number if the subtree with a certain number of leaves is not in the sequence of optimal pruned subtrees, while "Optimal_skip" skips the subtree.*

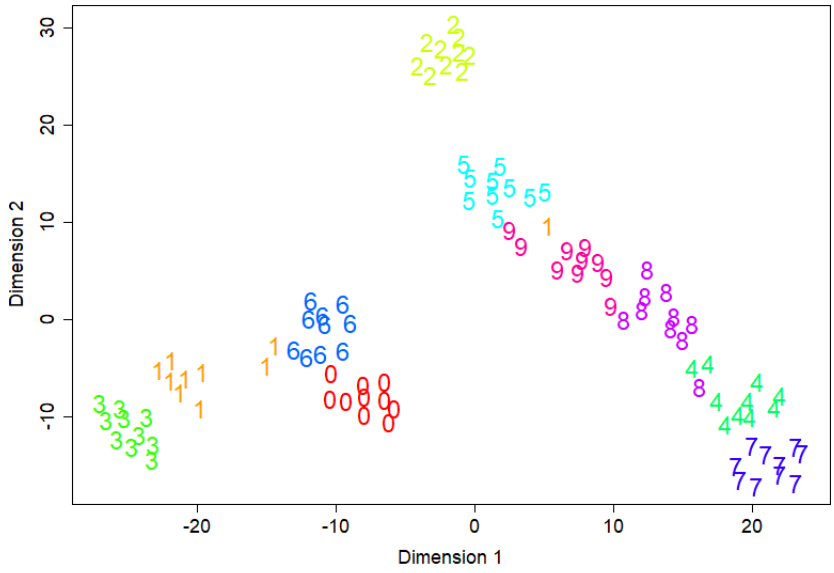


Figure 16: *TSNE projection of a random dataset with equal-sized well-separated clusters: 100 points, 15 dimensions, 10 clusters*

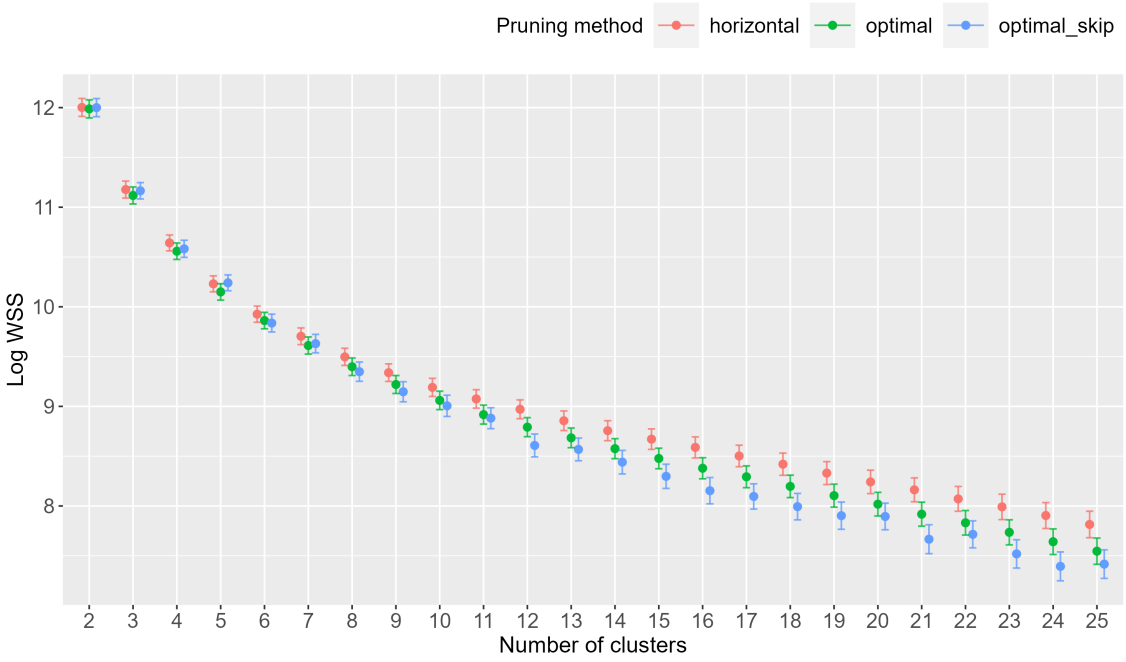


Figure 17: *Log WSS mean and SE comparison of the two pruning methods on 200 random datasets with equal-sized well-separated clusters. Optimal pruning increases the number of leaves to the closest available number if the subtree with a certain number of leaves is not in the sequence of optimal pruned subtrees, while Optimal_skip skips the subtree.*

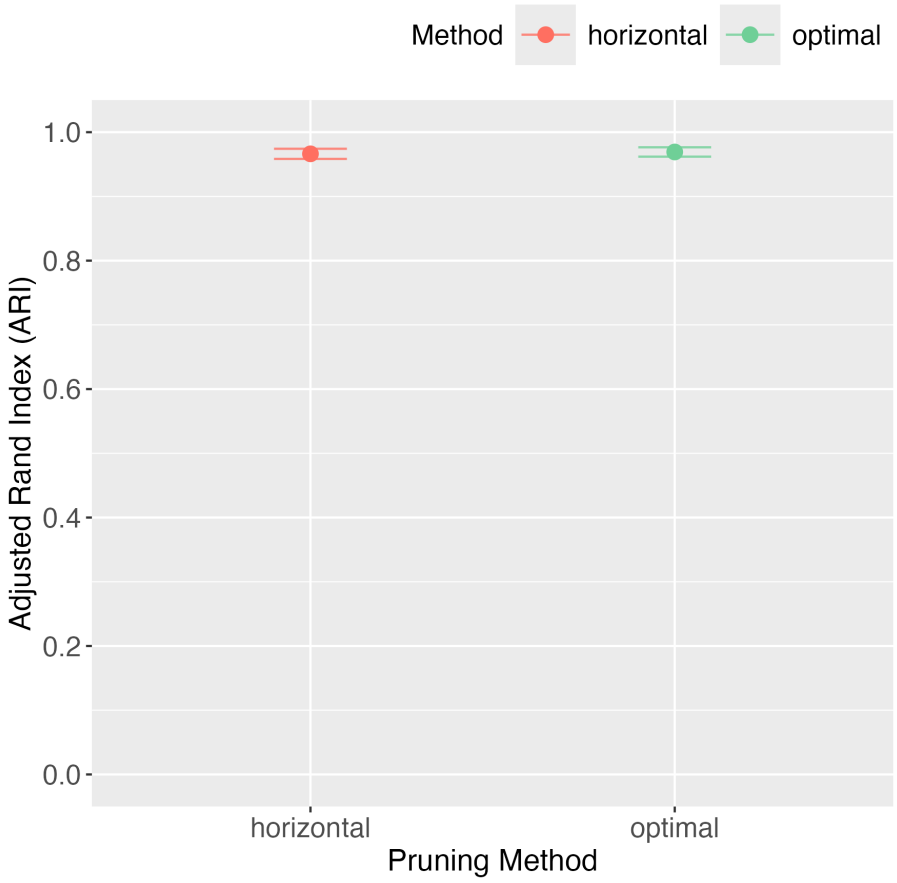


Figure 18: *ARI for horizontal and optimal pruning on 200 random datasets with equal-sized well-separated clusters.*

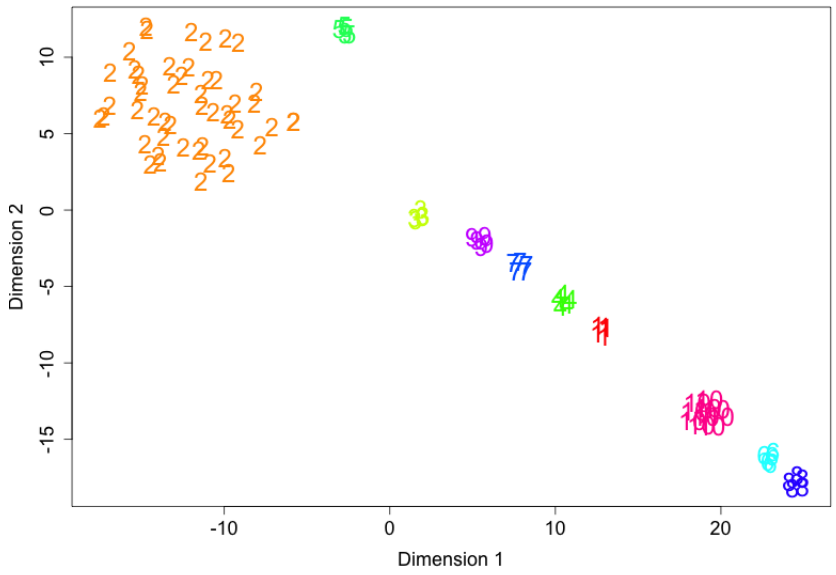


Figure 19: *TSNE projection of a random dataset with unequal-sized well-separated clusters: 100 points, 15 dimensions, 10 clusters*

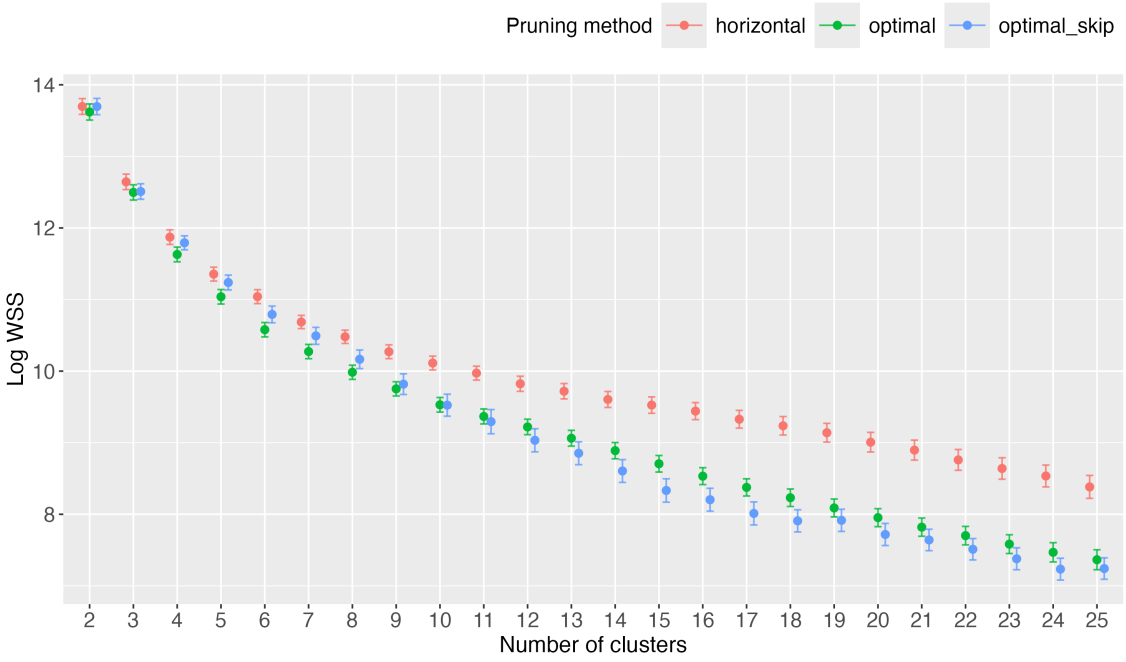


Figure 20: *Log WSS mean and SE comparison of the two pruning methods on 200 random datasets with unequal-sized well-separated clusters. Optimal pruning increases the number of leaves to the closest available number if the subtree with a certain number of leaves is not in the sequence of optimal pruned subtrees, while Optimal_skip skips the subtree.*

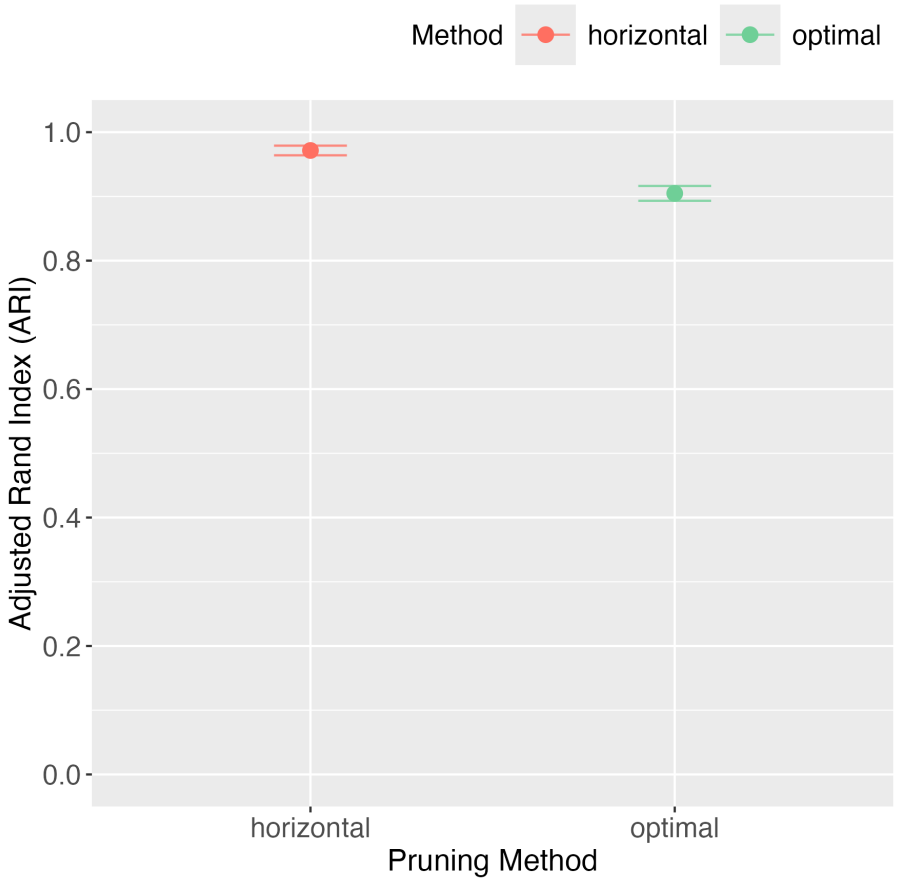


Figure 21: *ARI for horizontal and optimal pruning on 200 random datasets with unequal-sized well-separated clusters.*

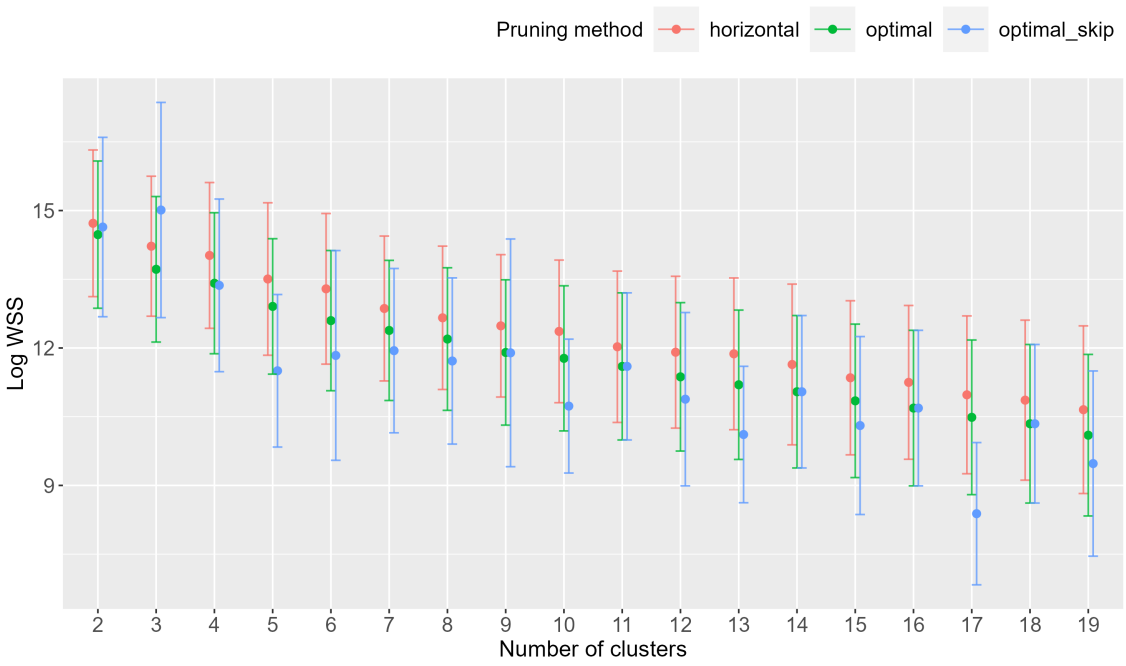


Figure 22: *Log WSS mean and SE comparison of two pruning methods on six real datasets. Optimal pruning increases the number of leaves to the closest available number if the subtree with a certain number of leaves is not in the sequence of optimal pruned subtrees, while Optimal_skip skips the subtree.*

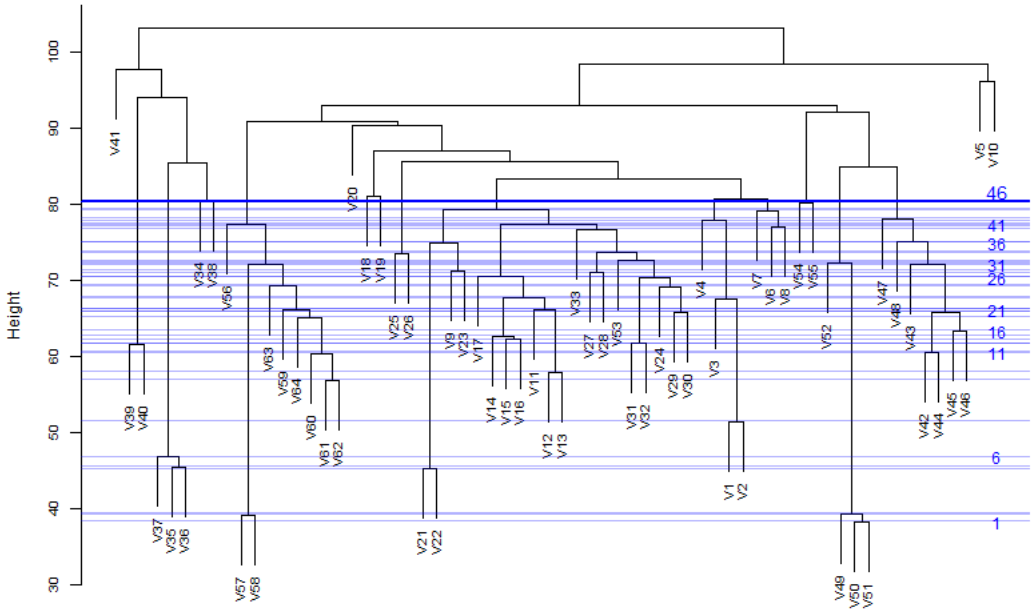


Figure 23: *The horizontal pruning steps resulting in 14 leaves for NCI-60*

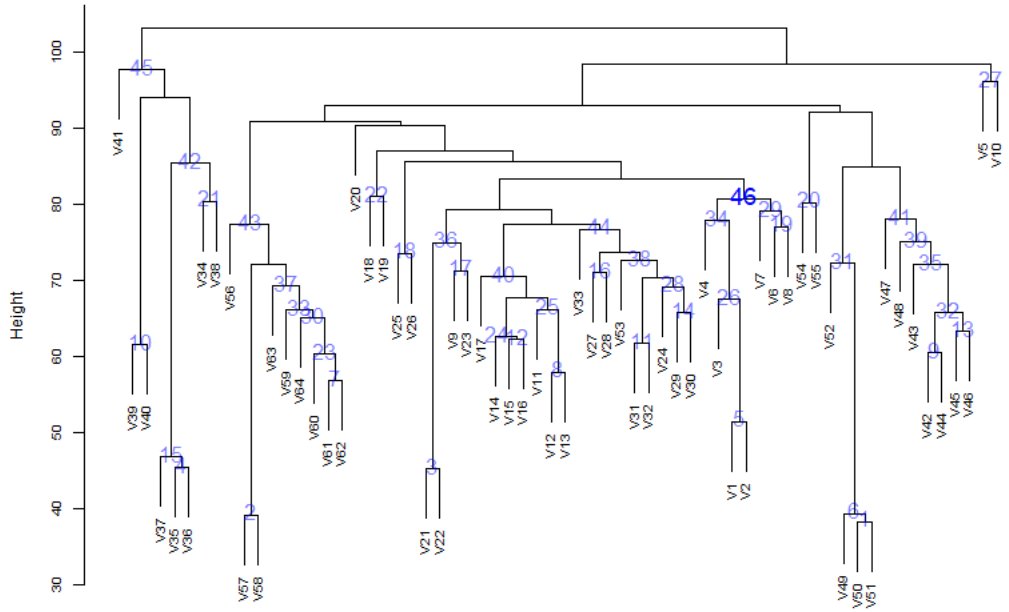


Figure 24: *The optimal pruning steps resulting in 14 leaves for NCI-60*

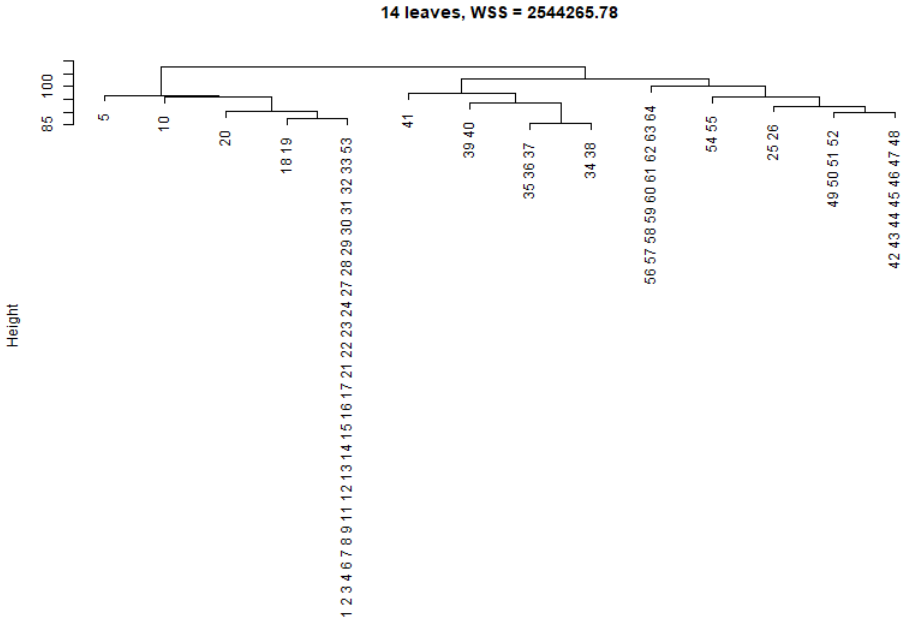


Figure 25: *NCI-60: The horizontal pruned tree with 14 leaves*

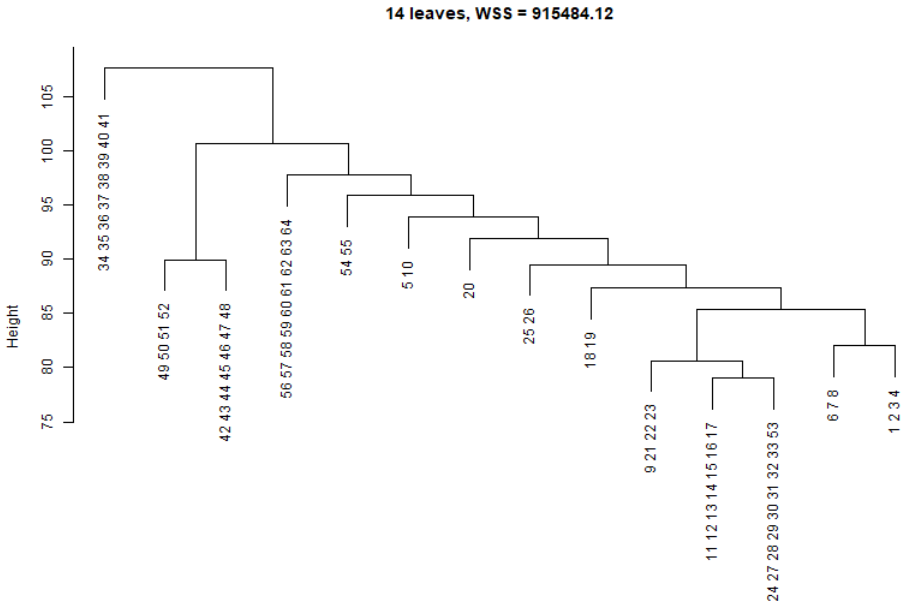


Figure 26: *NCI-60: The optimal pruned tree with 14 leaves.*