O^{3}NJ Trees: Optimally Ordered Orthogonal Neighbor Joining Trees for Hierarchical Cluster Analysis

Tong Ge
Shandong University
Qingdao, China
tgeconf@gmail.com

Yunhai Wang
Shandong University
Qingdao, China
cloudseawang@gmail.com

Michael Sedlmair
University of Stuttgart
Stuttgart, Germany
michael.sedlmair@visus.uni-stuttgart.de

Zhanglin Cheng
SIAT
Shenzhen, China
zl.cheng@siat.ac.cn

Ying Zhao
Central South University
Changsha, China
zhaoying@csu.edu.cn

Xin Liu
BGI
Shenzhen, China
liuxin@genomics.cn

Baoquan Chen
Peking University
Beijing, China
baoquan@pku.edu.cn

Oliver Deussen
University of Konstanz
Konstanz, Germany
oliver.deussen@uni.kn

Abstract

We propose to use optimally ordered orthogonal neighbor-joining (O^{3}NJ) trees as a new way to visually explore cluster structures and outliers in multi-dimensional data. Neighbor-joining (NJ) trees are widely used in biology, and their visual representation is similar to that of dendrograms. The core difference to dendrograms, however, is that NJ trees correctly encode distances between data points, resulting in trees with varying edge lengths. O^{3}NJ trees optimize NJ trees for their use in visual analysis in two ways: First, we contribute a novel leaf sorting algorithm that helps users to better interpret adjacencies and proximities within such a tree. Second, we propose a new method to visually distill the cluster tree from an ordered NJ tree. Case studies illustrate the benefits of this approach for exploring multi-dimensional data in areas such as biology or image analysis.

1. Introduction

Cluster analysis, a technique widely used in data science, divides data into groups of similar observations. While many fully automatic clustering algorithms exist, they do not always yield meaningful results. For most methods a proper number of clusters has to be pre-defined, which is not trivial, requires a human in the loop, and often leads to a tedious trial-and-error process. To cope with such challenges, visual interactive clustering [12] has been developed to support users in finding proper clustering parameters and inspecting their results.

Instead of producing a pre-defined number of clusters, hierarchical cluster analysis seeks to build a cluster hierarchy that enables users to explore possible clusterings at different levels. Many different algorithms have been proposed, see the comparison in [51]. A commonly used method is Agglomerative Hierarchical Clustering (AHC) [11] that creates a *dendrogram*, a binary tree diagram that illustrates how clusters are merged at each agglomeration step (an example is given in Fig. 1(d)). In such a diagram the leaf nodes represent data points, the positions of inner "joining" nodes represent the weighted mean of their children such that their position is proportional to the distance between the children in data space. To cluster a given data set, the user moves a "similarity bar" over the dendrogram (dashed black line in Fig. 1(d)), and while doing so s/he interactively "cuts" the dendrogram into pieces to find a proper number of clusters. Dendrograms originated from biology [47] for clustering genes, and we refer to them more generally as *HC trees* (hierarchical clustering trees). HC trees have been successfully applied in various visual analysis projects [43], and are part of standard systems for data analysis [17].

Despite their benefits, HC trees have three drawbacks that limit their effectiveness: First, all leaves with the same lowest common ancestor have the same path length between each other; therefore, their distance within the tree might not fit to the actual *distance in data space*, which is essential for characterizing clusters of general high dimensional

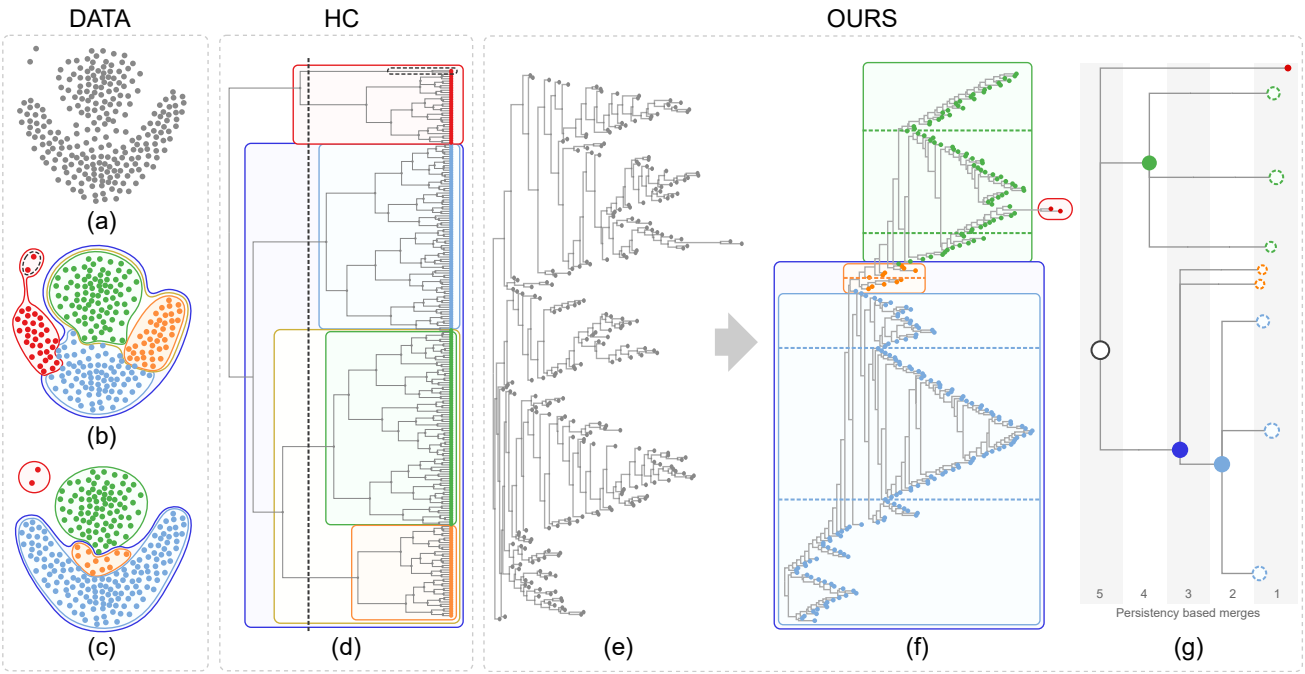

Figure 1. A given data set (a) is better clustered by our optimally ordered orthogonal neighbor joining tree (O^3NJTree) (f) than by a dendrogram (d) produced by the hierarchical clustering (HC) method with complete linkage. Subfigure (b) shows the nested five clusters generated by cutting the dendrogram with a minimum similarity bar; (c) shows four clusters automatically extracted with our method; (e) displays the NJ tree with random leaf order while (f) shows the same tree with optimal leaf order (left) and resulting cluster tree (right) as described in the paper.

data. An example is shown in Fig. 1(b), where each node of the red cluster has different distances to the nodes of the blue cluster, but using HC (see Fig. 1(d)) the tree distances between leaves of the red cluster to leaves of the blue are the same. This indicates that HC trees do not represent data distances well.

Second, the approach is not robust w.r.t. *outliers*. Since all leaves have the same distance to the root, outliers with large distances to other elements in data space are displayed with the same path length to other leaves in the tree. Thus, it is hard to see such outliers among the leaf nodes and therefore users need to explore internal nodes in addition. For example, the two outliers enclosed by the red circle in Fig. 1(c) have the same distance to root as all other leaves in the HC tree of Fig. 1(d).

Finally, HC trees are not able to display the *intrinsic cluster structure* of many data sets, since in every step the two closest clusters are merged. Such an approach fails in cases when clusters are not linearly separable [14], as it does not pay attention to inter-cluster distances. As shown in Fig. 1(b) and (d), the right part of the blue cluster is incorrectly merged with the above green cluster. Moreover, organizing clusters into a binary tree might be improper for data sets having three or more major clusters.

To mitigate these problems, we propose orthogonal neighbor joining (NJ) trees as an alternative for interactive hierarchical cluster analysis. In contrast to HC trees, only a few approaches use NJ trees for multi-dimensional data visualization [10, 15, 33]. These techniques mainly use these trees as a projection technique and show their results subsequently in radial layouts. Trees are used for the aforementioned advantage of showing data distances more precisely than other techniques. An orthogonal tree layout, however, performs better than a radial layout when using NJ trees for hierarchical cluster analysis [8, 30]. Since NJ trees have varying edge lengths, their leaves are not aligned to each other anymore, making it hard for the user to identify the cluster hierarchy (Figure 1(e)). This problem becomes even larger with increasing number of leaves.

To address this issue, we propose a new leaf ordering algorithm for NJ trees that places leaves with large similarities adjacent to each other. Such trees look like *terrains* with peaks and valleys (see Fig. 1(f)). We call these representations *Optimally Ordered Orthogonal Neighbor Joining* (O^3NJ) Trees. A number of leaf ordering methods have been proposed for HC trees [3, 9, 13, 40], but such methods cannot produce an optimal ordering for NJ trees because of the unequal edge lengths. As far as we know, our algorithm is the first method for ordering orthogonal NJ trees in which adjacent leaves satisfy two conditions: i) similar distances to the root, and ii) small data distances between them. Moreover, our method is over one order of magnitude faster than all other existing leaf ordering algorithms.

Although our leaf ordering algorithm arranges leaves

with similar root-leaf distances in adjacent positions, the exploration of the cluster hierarchy is not easy, since adjacent leaves might not belong to the same cluster. To improve exploration especially for large O^{3}NJ trees, we propose a method that highlights the intrinsic cluster tree by using its geometric representation as well as the topological features of the tree structure. Specifically, our ordering method first identifies outliers that have large distances to root, and then performs a persistence-based analysis to extract clusters, which are clearly visible "peaks" and "valleys" in the tree structure. In Figure 1(f) the two red leaves enclosed by a red circle define an outlier cluster, while the green, yellow and blue clusters are generated by analyzing peaks and valleys.

In summary, the main contributions of this paper are:

1. We propose a dedicated and fast *optimal leaf ordering algorithm* for orthogonal NJ trees (O^{3}NJ trees) so that clusters and outliers are clearly shown;
2. We introduce an algorithm to extract the cluster structure from O^{3}NJ trees;
3. We demonstrate the usefulness of our approach for exploring multi-dimensional data by conducting case studies in interactive clustering, evolutionary analysis, and image classification.

2. Related Work

Existing related work can be divided into three categories: hierarchical cluster analysis, visualization of cluster hierarchies, and leaf ordering for trees generated by hierarchical clustering.

2.1. Hierarchical Clustering Anaylsis

A complete review of hierarchical clustering is beyond the scope of this paper; we therefore refer the reader to Han et al. [22] and restrict our discussion to the design of Agglomerative Hierarchical Clustering (AHC) methods, which are most commonly used.

These methods work in a bottom-up manner [24]. Given n data points and an $n \times n$ distance matrix, an AHC algorithm first assigns each element to a cluster, then merges the closest pair of clusters into a single cluster and computes the distances between the new cluster and each of the other clusters. This step is repeated until only a single cluster remains. The distance measure between two clusters is referred to as the *linkage criterion*, which can be defined in various ways. Common criteria are single linkage, complete linkage, average linkage, centroid linkage, and Ward's method [31].

Each linkage criterion and associated AHC method has its advantages and disadvantages [51]. Previous studies show that average linkage, centroid linkage, and Ward's linkage are quite sensitive to shape and size of clusters [26, 29]. Single linkage can handle non-elliptical shapes, but is sensitive to noise and outliers. In contrast to them, complete linkage is less susceptible to noise and outliers, but tends to break large clusters. Since average linkage considers all pairwise distances for computing the cluster distance, it is more robust to outliers than other methods, but also computationally more expensive. Previous work [26, 29, 41] quantitatively compared these methods using numeric measures such as the cophenetic correlation coefficient [47] and provided guidelines for choosing appropriate clustering methods. For simplicity, we call all these methods "ordinary AHC methods" in order to distinguish them from our NJ algorithm, which is also an AHC method.

NJ trees are widely used for phylogenetic data analysis [39]. The NJ method resembles ordinary AHC methods, but has some unique properties. Most importantly, it ensures that each two merged clusters are not only close to each other but also far apart from the rest. Hence, the generated cluster hierarchies might differ from the ones produced by ordinary methods. Biological data analysis has shown that NJ trees perform better in many cases [32, 48].

Besides biology, NJ trees have also been used for the visualization of document collections, where NJ clustering results are visualized as a multi-dimensional projection using a radial layout [10, 33]. There is, however, no previous study that compares these methods for general multi-dimensional data. Our comparative evaluation shows that NJ trees not only better preserve input distances but also the rank order for most data sets. In this paper we show how this tree structure helps the hierarchical cluster analysis of large data sets.

2.2. Visualization of Cluster Hierarchies

A variety of graphical representations have been developed for rendering tree structures, including classical node-link diagrams, icicle, nested enclosure, indented outline or treemap representations [45]. McGuffin and Robert [28] systematically compare the space-efficiency of these methods and provide guidelines for choosing a good representation. Here we focus on node-link diagrams, which are most commonly used to visualize cluster hierarchies. For hierarchies generated by ordinary AHC and NJ methods, the corresponding diagrams are referred to as *cladogram* or *phylogram* (in biology) [34], both of them are specific types of dendrograms. For convenience, we refer to cladograms as *HC trees* and phylograms as *NJ trees*. Both tree diagrams encode distances between data elements by using path lengths, with NJ trees having unequal path lengths from the root to the leaves.

Dendrograms are often drawn with an orthogonal layout, either in vertical or horizontal orientation. The hierarchical clustering explorer by Seo and Shneiderman [43] enhances HC trees with dynamic query controls for interactive ex-

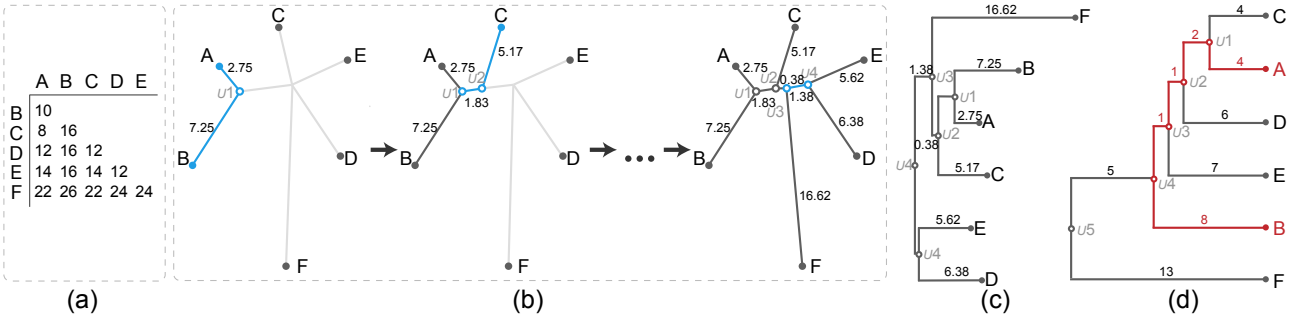

Figure 2. Illustration of the NJ algorithm and comparison between the NJ tree and the HC tree. (a) The input distance matrix; (b) the procedure of the NJ algorithm by using the input matrix shown in (a); (c) the orthogonal NJ tree where the root is the duplication of U_4; (d) the HC tree generated by applying the AHC method with the average linkage to the matrix in (a).

ploration. Munzner presented tree-juxtaposer [30] which allows users to navigate large hierarchies with a global rectangular focus+context technique. Etemadpour et al. [15] have shown that NJ trees drawn with radial layouts [1] can be used as a projection method for multi-dimensional data. But representing NJ trees with a radial layout is less suited for hierarchical structures as reported by Burch et al. [8]. An orthogonal layout seems to be a good compromise between compactness and readability [30]. Thus, in this paper we investigate how orthogonal tree layouts can be used to support visual analysis of NJ trees.

2.3. Leaf Ordering for Hierarchical Clustering

For an orthogonal tree diagram, leaf nodes are shown in linear order along one axis. Adjacent leaves in such a linear ordering are assumed to be related [19], and thus a good leaf ordering helps users to identify clusters of interest and interpret the data. For a tree with n leaves, 2^{n-1} linear orderings are possible. To find a proper leaf ordering, a number of methods have been proposed [3, 9, 40] that minimize distances of adjacent leaves. Most of them, however, aim at sorting ordinary HC trees with equal edge lengths and do not take the special characteristics of NJ trees into account. Specifically, ordinary AHC methods pick the two closest clusters for merging, if the corresponding nodes of these clusters are adjacent in the HC tree, their leaf ordering accurately reflects the merging procedure. The NJ method, however, merges nodes that are not only similar to each other, but also far from other nodes. Thus, if we would simply apply existing ordering methods for NJ trees, the ordered leaf nodes would not accurately represent the cluster structure revealed by the NJ algorithm. Fig. 3(e) shows an example, where the centroid cluster is interrupted by three outliers (B, K and E).

To bridge this gap, we propose an efficient leaf ordering algorithm that finds a linear ordering in which the sum of absolute edge length differences between adjacent elements is minimized. Such an ordering visualizes the separation between large groups very clearly, meanwhile it allows to spot outliers and small clusters on a detailed level.

3. Background

In this section, we first describe the neighbor joining algorithm and then our qualitative comparison between AHC and NJ trees to justify that they better represent general high-dimensional data.

3.1. Neighbor Joining Algorithm

Given a distance matrix with the relations of n data points in d dimensions. The NJ method starts with a star-like tree where each leaf node corresponds to a data element. Iteratively two neighboring nodes are joined until a complete binary tree is obtained:

1. Calculate a new distance matrix Q from D by setting $Q_{ij} = D_{ij} - (S_i + S_j)$ where S_i is the net divergence:

$$S_i = \frac{1}{n-2} \sum_{k \neq i} D_{ik}. \tag{1}$$

2. Identify the pair of nodes i and j with minimal Q_{ij};

3. Join nodes i and j at node x and compute the branch lengths between from node x to i and j:

$$\begin{aligned} l_{xi} &= D_{ij}/2 + (S_i - S_j)/2 \\ l_{xj} &= D_{ij}/2 + (S_j - S_i)/2 \end{aligned} \tag{2}$$

4. Update D by replacing nodes i and j with node x and compute the distance from x to each other node k;

$$D_{xk} = (D_{ik} + D_{jk} - D_{ij})/2 \tag{3}$$

5. Repeat the steps 1-4 until only two nodes remain.

By constructing Q based on the average divergence while joining nodes using the least-distant pair of nodes in

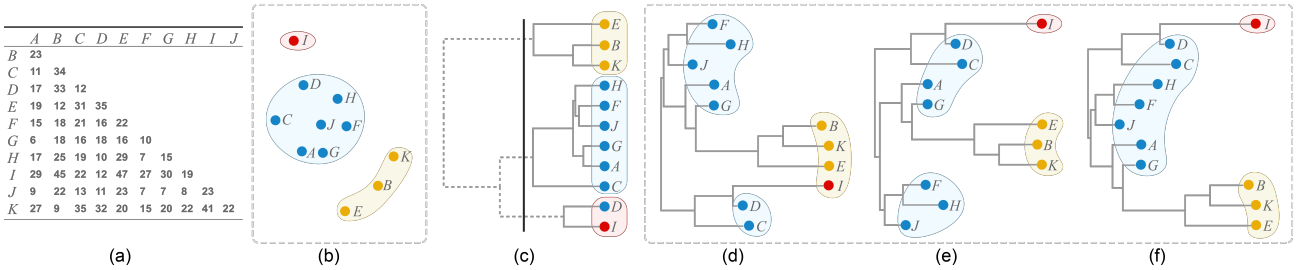

Figure 3. Comparison between ordinal hierarchical clustering (HC) and NJ trees generated from the same distance matrix as shown in (a). (b) Scatter plot with three clusters produced by multi-dimensional scaling; (c) HC tree where positions do not accurately resemble distances, e.g., the distance between I and A is 29 and the distance between I and B is 45, but the horizontal position of their least common ancestor (the root node) has the same horizontal path length to the nodes I, A and B. Moreover, cutting the HC tree into three branches (black line) results in incorrect groupings for I and D. NJ trees accurately encode the distance matrix, but so far no good ordering algorithm exists: (d) NJ tree displayed with a random order, as currently done in standard analysis packages such as R; (e) NJ tree ordered by using the OLO algorithm [3]; (f) NJ tree ordered by our algorithm, three useful clusters are created.

Q, the NJ algorithm takes into account intra-cluster compactness as well as inter-cluster separation.

Fig. 2(b) illustrates the algorithm with the input distance matrix shown in Fig. 2(a). Nodes A and B are first joined with the minimum $Q_{AB} = -27.5$. The distances from these two nodes to their common ancestor U_1 are 2.75 and 7.25. Following this procedure, an un-rooted NJ tree that fits to the actual distance in data space is generated.

Rooting Strategy As already mentioned (and shown in Fig. 2(b)), an NJ tree is un-rooted by default. However, biologists often explore NJ trees using orthogonal layouts, so a root is needed, cf. [24]. To address this issue, we follow a rooting strategy given in Butto [5] that simply duplicates the last formed internal node to create the root node (see Fig. 2(c)). The root can also be specified by a domain expert or determined by other strategies like using the midpoint of the longest path between any two leaves in the tree [5]. Note that we cannot follow the AHC method [36] to form the root by merging the last two remaining nodes, because the inter-cluster distance cannot be computed for two nodes.

3.2. Distance Preservation within NJ Trees

Previous studies have shown that NJ trees much better fit to the input distances than AHC trees [48, 32]; however, most of these studies are based on biological data. To verify that NJ trees also are better in representing general high dimensional data than AHC trees, we quantitatively compared them against AHC trees produced by different variants of the AHC algorithm. We checked how well distances between any pair of data points fit the path lengths between the corresponding leaf nodes of the trees. The path length is defined as the sum of all edge lengths in the path from node i to j.

We collected 47 data sets of different size and dimensionality with substantial variations and measured the differences between data point distances and edge lengths. Experimental details and results can be found in the supplemental material, which show that NJ trees better represent the input data than AHC trees for most data sets.

Figs. 2(c,d) show an example to illustrate why an orthogonal NJ tree performs better than an AHC tree. Path lengths between any pair of nodes in an NJ tree are much closer to the data distances between the nodes than for AHC trees. For example, the path length between leaf nodes A and B in the AHC tree is 16, while the data distance between them is only 10. The paths between node pairs (A,F), (C,F), (D,E), (D,F) and (E,F) show the same issue. This results from the merging procedure within AHC methods, where computing branch lengths is solely based on intra-cluster distances D_{ij} [16], without considering the corresponding inter-cluster distance $S_i + S_j$.

While NJ trees are superior in representing data distances faithfully, they cannot correctly encode the distance for any given distance matrix, especially ones that do not obey Buneman's 4-point condition [7]. However, previous studies show that NJ trees are still one of the best approximations for those matrices [27].

4. Leaf Ordering for Orthogonal NJ Trees

Once we have a rooted orthogonal NJ tree with n leaf nodes, there are 2^{n-1} possible leaf orderings, which is similar to an HC tree. Since closeness of adjacent leaf nodes visually indicates similarity of the underlying data points (such as coming from the same cluster), an optimal leaf ordering (OLO) would help users to detect cluster structures in the data. This problem has been studied well for HC trees [3, 9, 40], but to the best of our knowledge no studies exist for NJ trees.

In Fig. 3(d) we show that random ordering mixes different clusters, hindering the user to determine meaningful

cluster boundaries. For example, nodes I, E, K, and B in Fig. 3(d) are adjacent but node I is an outlier as shown in Fig. 3(b). Furthermore, leaf orderings produced by existing OLO algorithms do not really work for NJ trees, since these algorithms are designed for revealing cluster structures produced by ordinary AHC algorithms, which are different from our proposed NJ algorithm. Instead, our proposed NJ tree oriented OLO algorithm (NJOLO) aims to reveal the cluster structures characterized by the NJ algorithm. An example is shown in Fig. 3(f), where the two clusters and the outlier are clearly separated.

4.1. Problem Definition

As shown in Fig. 3(f), nodes belonging to the same cluster have similar path lengths to the root and should be adjacent to each other. Hence, our desired ordering places nodes with similar path lengths from the root at adjacent locations. Only optimizing this objective, however, would not be enough, since nodes with similar path lengths to the root might not be similar. For example, the leaf nodes B, E, K and I in Fig. 3(d) have similar path lengths from the root, but nodes B, E, and K form one cluster in the MDS-based scatterplot (see Fig. 3(b)) while node I is an outlier. By carefully analyzing the result, we see that the distances between nodes B, E, K and I are quite large, which is verified by the scatterplot shown in Fig. 3(b). In other words, if two nodes with similar path lengths from the root have a large pairwise distance, they should not be adjacent. Thus, we enforce an additional constraint: the data distance between two adjacent nodes should not be larger than a given threshold, otherwise the nodes have to be separated.

For short, the path length from the root to a leaf node i is denoted as p_i. We aim to find an ordering ϕ that minimizes the sum of absolute path length differences between adjacent leaves ϕ_i and ϕ_{i+1}:

$$\min_{\phi} \sum_{i=1}^{n-1} |p_{\phi_i} - p_{\phi_{i+1}}|, \tag{4}$$

with a hard constraint that $D_{i,i+1}$ should be smaller than a given threshold t.

Relationship to HC Tree Ordering. The OLO for HC tree aims to find an ordering that minimizes the difference of the adjacent leaves in the ordering:

$$\min_{\phi} \sum_{i=1}^{n-1} D_{\phi_{i,i+1}}. \tag{5}$$

If the path length in the HC tree can represent the distance between data points, Eq. (5) can be written as:

$$\min_{\phi} \sum_{i=1}^{n-1} p_{\phi_i,\phi_{i+1}}, \tag{6}$$

Algorithm 1 Optimal Leaf Ordering of NJ tree

```
function OLONJ(u, D)
    if |u| == 1 then
        C(u, v, v) = 0 return C(u, v, v)
    else
        C(u_l, L, R) = OLONJ(u_l, D)
        C(u_r, L, R) = OLONJ(u_r, D)
        for v in leaves of u_l do // left subtree of u
            for w in leaves of u_r do // right subtree of u
                if D_{m,k} ≤ t then
                    C(u, v, w) =
                        min_{m∈u_l, k∈u_r} C(u_l, v, m) + C(u_r, k, w)
                            + |p_m − p_k|
                else
                    C(u, v, w) = +∞
                end if
                C(u, w, v) = C(u, v, w)
            end for
        end for
    end if
    return C(u, L, R) // L and R denote all possible
                      // pairs of leaves from u_l and u_r
end function
```

which is the sum of path length between adjacent leaf nodes. Since it is substantially different from Eq. 4, applying it to the NJ tree produces undesired results. Fig. 3(e) shows an example in which the ordering separates node I from the clusters, but places the yellow cluster in the middle of the blue cluster.

4.2. Dynamic Programming

Like the OLO algorithms [3, 2] for HC trees, Eq. (4) can also be solved by using dynamic programming (DP). Since this problem is defined on the NJ tree, we can take the advantage of having a binary tree structure to decompose this problem into subproblems for ordering sub-trees.

Hence, we associate the ordering of each leaf subsequence with an internal node. For each internal node u, let its children be u_l, u_r, the number of leaves $|u|$, and the cost of the optimal ordering of its subtree $C(u)$. For every pair of leaves $v \in u_l$ and $w \in u_r$, $C(u, v, w)$ is the cost for optimal ordering of the subtree rooted at u with the leftmost and rightmost leaf nodes v and w, and $C(u)$ is the minimum of all possible $C(u, v, w)$. Based on the tree decomposition, Eq. (4) can be re-written in a recursive form:

$$C(u, v, w) = \min_{m \in u_l, k \in u_r} C(u_l, v, m) + C(u_r, k, w) + d(m, k) \tag{7}$$

$$\text{subject to } \; D_{m,k} \leq t$$

where the leaf m is the rightmost leaf of μ_l and k is the

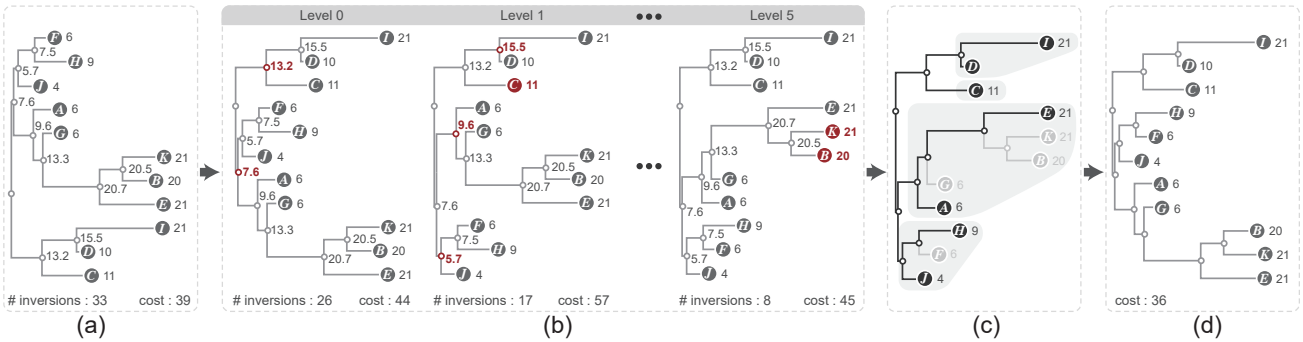

Figure 4. Overview of our leaf ordering algorithm: (a) an orthogonal NJ tree with unequal edge length where each node is assigned a weight based on the edge length; (b) by performing monotonicity based flipping within a breadth-first traversal the inversion is reduced; (c) extraction of the ordered sub-trees with a depth-first traversal; (d) the optimal leaf ordering result is obtained by applying the DP algorithm to the ordered sub-trees.

leftmost leaf of μ_r. By default, $d(m,k)$ is the absolute path length difference between the nodes m and k. To integrate the hard constraint $D_{m,k} \leq t$ to the DP procedure, we set $d(m,k)$ to $+\infty$ if the distance between nodes m and k is larger than t. Hence, $d(m,k)$ is defined as:

$$d(m,k) = \begin{cases} |p_m - p_k|, & \text{if } D_{m,k} \leq t \\ +\infty & \text{if } D_{m,k} > t. \end{cases}$$

As shown in Algorithm 1, this ordering works in a bottom-up way. When calculating the C values for the subtree rooted by u, the C values of u_l and u_r are already computed and stored in a table. Once $C(\mu, v, w)$ for all pairs of v and w are computed, the smallest value of $C(u,v,w)$ is $C(u)$. Meanwhile, early termination is applied to optimize the search for the smallest $C(u,v,w)$. The pseudo-code is given in Algorithm 2.

Time Complexity. Because of the tabled values, $C(u,v,w)$ is computed only once for each of the $O(n^2)$ pairs of leaves v and w. Each computation of $C(u,v,w)$ involves all the possible m,k leaves that lie in the intersection of u_l and u_r and thus results in at most $O(n^2)$ time. In all, the time complexity of the whole algorithm is $O(n^4)$.

4.3. Acceleration techniques

We propose two acceleration techniques to dramatically decrease the running time of our algorithm while producing an accurate and optimal leaf ordering.

Early Termination. Since all values of $C(u_l,v,R)$ and $C(u_r,L,w)$ are already computed when we compute $C(u,L,R)$, we can terminate the search of best pairs of m and k early when $C(u,L,R)$ cannot be further improved. To achieve this goal, we take the following pre-processing steps to reduce the computation time:

- compute the minimal path length difference $minP = \min_{v \in u_l, w \in u_r} |p_m - p_k|$;
- sort $C(u_l,v,R)$ and $C(u_r,L,w)$ in ascending order, where R denotes all possible right leaves of u_l when v is the leftmost leave of u_l and L is the same for w and μ_r.

Accordingly, we compute $C(u,v,w)$ by performing two loops of m and k according to the order of $C(u_l,v,R)$ and $C(u_r,L,w)$. Denote by $curC$ the current minimal cost we have for $C(u,v,w)$. For a given pair of leaves m and k, if we have $C(u_l,v,m) + C(u_r,k,w) + minP \geq curC$, then any leaf pair m, k' coming after k cannot produce a value that is smaller than $curC$ and thus the loop of k can be terminated. Likewise, the loop for m can also be terminated earlier. Algorithm 2 outlines this procedure.

Sequence Simplification. Looking at the array $P = \{p_1, \cdots, p_n\}$ we see that Eq. 4 reaches its minimum when P is a monotonic sequence. While this does not allow us to overlook the underlying NJ tree, it suggests us to reduce the search space of the DP algorithm by partitioning the whole tree into a few sub-trees whose path lengths to the root form a few monotonic sub-sequences. Since the leaf ordering in each of these monotonic sub-trees could be seen as fixed, we can simplify each sub-tree by its leftmost and rightmost leaves for finding the optimal leaf ordering and the computations time is further reduced. To this end, we perform the following two steps to construct a few monotonic sub-trees.

- Initialization. A weight is assigned to each node. For a leaf node, the weight is its path length to the root; for each internal node μ, its weight is the average of all weights of its leaves. Fig. 4(a) shows an example, where all nodes have been assigned weights except the root.

Algorithm 2 Early Termination for computing $C(u, v, w)$

```
minP = min_{v∈u_l, w∈u_r} |p_m − p_k|
curC = +∞
for m in ordered C(u_l, v, m) do
    if C(u_l, v, m) + C(u_r, k_0, w) + minP ≥ curC then
        C(u, v, w) = curC; break
    end if
    for k in ordered C(u_r, k, w) do
        if C(u_l, v, m) + C(u_r, k, w) + minP ≥ curC then
            break
        end if
        if D_{m,k} < t then
            tmp = C(u_l, v, m) + C(u_r, k, w) + |p_m − p_k|
        else
            tmp = +∞
        end if
        if curC > tmp then
            curC = tmp
        end if
    end for
end for
C(u, v, w) = curC
```

- Flipping. A monotonic sequence does not have any *inversions*. Two elements p_i and p_j form an inversion if $p_i > p_j$ and $i < j$. We reduce the number of inversions within a sub-tree by flipping nodes at each level using a breadth-first traversal. For each internal node u at the ith level, we check if the weight of its right child node u_r is smaller than the one of the left child node u_l. If this is the case, we flip the two sub-trees rooted at u. This level by level flipping gradually reduces the number of inversions. In Fig. 4(b) the number of inversions are reduced from 33 to 8 after traversal. Since nodes at the same level might not be adjacent because of unequal edge lengths, flipping cannot reduce the cost of leaf ordering.

On the right of Fig. 4(b), a few ordered sub-trees with monotonically decreasing edge lengths are generated and then such sub-trees can be extracted by performing the depth first traversal. Based on sub-trees, we employ the DP algorithm with early termination to find an optimal leaf ordering. Fig. 4(d) shows the final ordering result for the input NJ tree, noticing that the threshold t is by default the average of the distance matrix.

5. Cluster Tree Distilling

As mentioned above, the leaves of an O^{3}NJ tree have unequal root-leaf distances and thus do not align to each other. Moreover, two adjacent leaves might not belong to the same cluster. For example, the two red leaves enclosed in a red box in Fig. 1(f) form a cluster of outliers, while their adjacent green leaves belong to another cluster. These two characteristics create difficulties for the user to explore a hierarchy when using an NJ tree. To address this issue, we propose a new method to distill a cluster tree from an O^{3}NJ tree by taking it as a visual representation. Fig. 5 shows the pipeline of our method that consists of two steps: 1) tree splitting based on an analysis of root-leaf distances and 2) hierarchical clustering of the 1D curve (the gray curve in Fig. 5(b)) formed by connecting the positions of adjacent leaves with line segments.

Tree Splitting. In this step, we first compute the **A**bsolute Adjacent **R**oot-leaf Distance **D**ifference (ARD) for all pairs of adjacent leaves and then find a proper threshold to cut the tree. The histogram on the right of Fig. 5(a) shows the sorted ARD set of the tree on the left. We see that only two adjacent leaf pairs have large distance differences. This is reasonable, since our leaf ordering algorithm guarantees that most adjacent leaves have a similar root-leaf distance. Thus, we consider ARD values as outliers if their values lie above a threshold:

$$\omega = q75 + 1.5(q75 - q25)$$

where $q75$ and $q25$ are the 75th and 25th percentile of the ARD values. Once we defined ω according to that formula, we can cut the tree into different branches, cf. Fig. 5(b).

Hierarchy generation. For each branch in the split O^{3}NJ tree, adjacent leaves have similar root-leaf distances. Connecting them with line segments forms a curve with peaks and valleys. In Fig. 5(c), such peaks and valleys are highlighted for the middle branch. For each leaf we find the lowest common ancestor (LCA) for its left and right adjacent nodes. Once they are found, we distill a new cluster tree based on the parent-child relationship between these LCA nodes. Fig. 5(d) illustrates the two-level hierarchy of clusters obtained by such a peak-valley analysis, the corresponding clusters are shown in Fig. 5(e).

Visual Encoding. We visualize the distilled cluster tree with an orthogonal tree where the number of children of each parent is determined by cluster structures. For example, the root nodes in Fig. 1 and Fig. 5 have two and three children, respectively. Meanwhile, the size of each node is proportional to the number of elements in the corresponding cluster, while each node has a unique color.

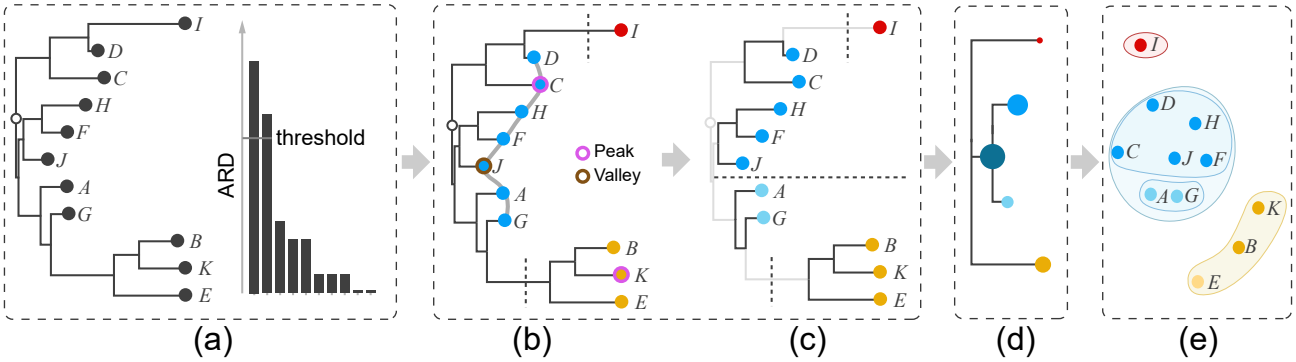

Figure 5. Distilling the cluster tree from an O^{3}NJ tree includes splitting and hierarchy extraction: (a) input O^{3}NJ tree and a histogram of the ARD values; (b) the selected threshold from the histogram in (a) cuts the tree into three parts; (c) identified peaks and valleys as well as the corresponding two-level cluster tree for the middle branch; (d) distilled cluster tree and nested clusters shown as scatter plot (right).

Persistence based Simplification. Because of our sub-monotonic DP algorithm, the leaves of an O^{3}NJ tree do not satisfy strict monotonicity, which results in many small noisy peaks and valleys. In order to reveal the hierarchy of major clusters, we use the persistence-based topological simplification by Weinkauf et al. [50] to remove such noisy peaks and valleys, where the persistence is defined by the root-leaf distance difference between adjacent peaks and valleys. We successively remove adjacent peaks and valleys with the smallest persistence values and merge them to adjacent large ones until a given persistence threshold β is reached. Since most persistence values are quite small (see Fig. 6(b)), we set β to a value of 10% of the maximal persistence following the suggestion of Gyulassy et al. [20]. As shown in Fig. 6(c), this default threshold removes most noisy peaks and valleys, while resulting in meaningful clusters.

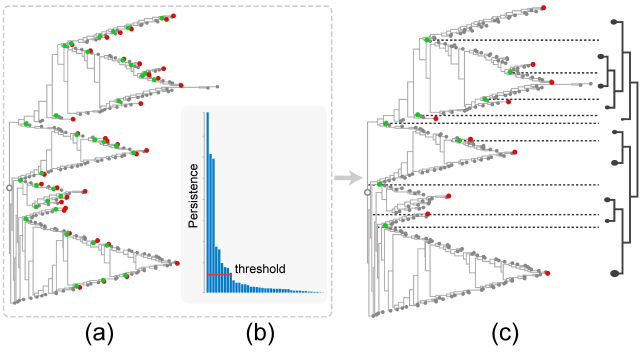

Figure 6. Persistence based simplification of an O^{3}NJ tree: the optimally ordered version of Fig. 1(e) is the input shown in (a), a default threshold of 10% of the maximal persistence value shown in the persistence histogram in (b) removes most noisy peaks and valleys and results in the cluster hierarchy shown in (c).

6. Evaluation

We implemented our O^{3}NJ tree in Javascript and provide a corresponding interface[1]. Although their default values produce reasonable results for most data, fine-tuning these parameters allows the user to generate better clusters for challenging datasets. Using this interface, we conducted three case studies, using one synthetic dataset and two real datasets from biology and computer vision.

6.1. Synthetic Data

To demonstrate the effectiveness of our O^{3}NJ tree in clustering, we did a first experiment with the synthetic *Compound* data [18], which has been used for the evaluation of various clustering algorithms. It is a challenging 2D dataset that contains arbitrarily shaped clusters with various densities. Fig 7 depicts our O^{3}NJ tree and clustering results generated by hierarchal clustering with complete-linkage and DBSCAN [14].

Fig 7(a) shows the O^{3}NJ tree (left) and its two-level distilled cluster tree (right). The corresponding clustering result is shown in Fig 7(b). Compared to the ground truth (see Fig 7(f)), the four clusters on the left are correctly identified, while the inner green cluster on the right side is mis-grouped with a part of the outer orange cluster. By looking at the histogram of the ARD values of the green cluster (Fig 7(c,d)), we found that most of the inner green nodes have smaller ARD values than the mis-grouped nodes. Hence, we use the default ARD threshold to separate the inner green cluster from such outlier nodes, shown with a black border in Fig 7(d,e). Combining this result with the automatically found clusters in Fig 7(b) we obtained the clustering result shown in Fig 7(i), where five major clusters are clearly separated.

Although the green outlier nodes are still not merged with the orange nodes in the bottom right, this result is

[1] http://www.njvis.net/

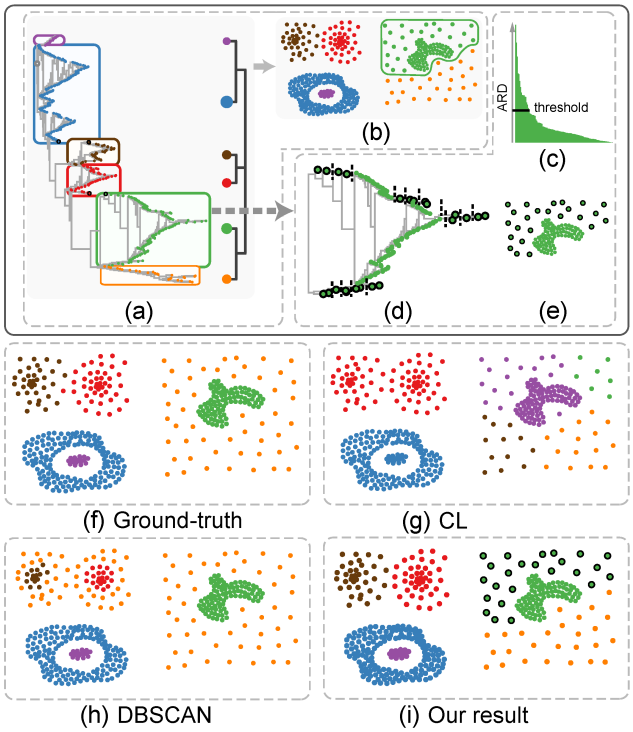

Figure 7. Cluster analysis of the *Compound* data (a-e) with our O^{3}NJ tree and comparing our result (i), to the ground truth (f), as well as to clusterings generated by the complete-linkage method (g) and DBSCAN (h). The cluster tree is distilled from the O^{3}NJ tree (a) the result is shown in (b), where the mis-grouped green cluster is highlighted. By thresholding its ARD histogram (c), green outlier nodes are separated from the inner green clusters (d,e).

better than the ones generated by the ordinary AHC methods and DBSCAN. More specifically, the complete-linkage method is not able to correctly identify any major cluster, while DBSCAN identifies four major clusters but mis-groups the two clusters on the top left (see Fig 7(h)). Thus, our automatic result is already comparable to DBSCAN (see Fig 7(b) vs. Fig 7(h)). Including user feedback, it quickly produces almost perfect results, as shown in Fig 7(i). This is a first evidence that clustering with our O^{3}NJ tree approach could work better than state-of-the-art clustering methods for certain challenging datasets.

6.2. Cucumber data

In evolutionary analysis [23, 37], biologists often use multi-dimensional scaling (MDS) [4] to reduce high-dimensional genome data collected from various species into 2D scatterplots. These scatterplots are then used to explore class separation and study evolutionary relationships among species with NJ trees. Our O^{3}NJ tree not only combines these two functionalities but also produces more accurate results as illustrated by the following case study of a

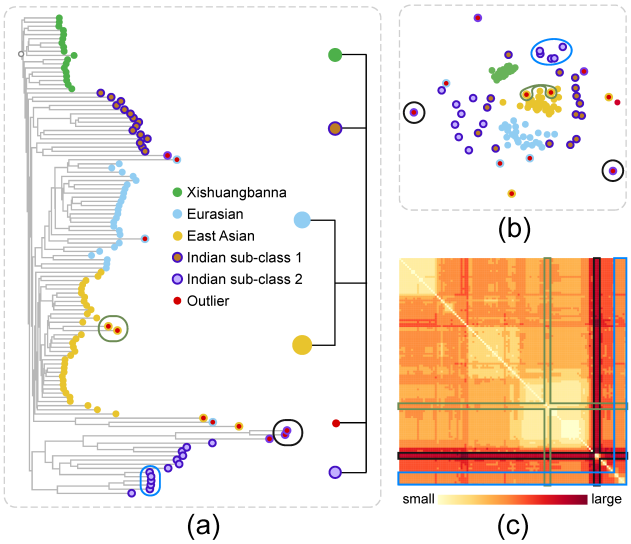

Figure 8. Exploration of the *Cucumber* data: (a) O^{3}NJ tree and the distilled hierarchy; (b) MDS plot; (c) heat map showing the input distance matrix. Three kinds of inconsistencies between our O^{3}NJ tree representation and MDS results are highlighted by black, green and blue circles in O^{3}NJ tree, while the corresponding nodes are also highlighted in MDS plot and the heap map. Class label and cluster index of each node are encoded by border and fill color.

cucumber genome dataset, provided by one of our collaborators, a biologist, who has more than 10 years of experience in evolutionary analysis.

His data was collected by sequencing 23,436 genes from 115 cucumber species classified into 4 geographical classes: East Asian, Eurasian, Indian, and Xishuangbanna. Using this data, our collaborator had two analysis goals: i) verifying if the classification matches with the inherent clusters in the data; and ii) comparing the distribution of the Indian class to the other classes, because he assumed that the Indian group should be closer to the wild type [42], while the other three groups belong to the cultivated type.

We analyzed the data together with our collaborator. Using default parameters, we obtained an O^{3}NJ tree and a distilled two-level cluster tree with six clusters. Our collaborator expected a one-to-one mapping between class labels and cluster index (Task 1). While there were only four classes, our method produces five major clusters and one outlying cluster. Our O^{3}NJ tree tool visualizes such inconsistency by encoding the class label and cluster index of the corresponding nodes into their border and fill color (see Fig. 8 (a)).

After examining the tree in Fig. 8 (a), our collaborator came up with three observations: i) the green, cyan, and yellow clusters match well with the classes of *Xishuangbanna*, *Eurasian* and *East Asian*; ii) the brown and purple clusters both belong to the Indian class; iii) a few outliers

are identified from all clusters except the green one. From the first two observations, he concluded that most classifications align with the data characteristic and the division of the *Indian* class is also as expected. Based on the last observation, he then concentrated on the *Xishuangbanna* class, where the edge lengths from the root i) are smaller than other classes and ii) have fewer variations. He did not expect this because the *Xishuangbanna* class uniquely accumulates β-carotene in its fruit [35]. Similarly, the five outliers in the *East Asian* class, especially the three adjacent to the *Indian* sub-class 2 were also unexpected. He told us that the *East Asian* class is often assumed to be well cultivated and he was not aware of a species close to the wild-type in this class.

To verify this observation, he checked if our O^{3}NJ tree is correct by comparing it to an MDS projection of the input distance matrix (see Fig. 8 (b), colors are like in subfigure (a)). The classes shown in green, cyan, and yellow are also well-separated in the MDS plot, but the distribution of two Indian sub-classes in brown and purple is quite different from our O^{3}NJ tree shown on the right of Fig. 8 (a). Some nodes have small distances in the tree but large distance in the MDS plot (two outliers denoted by a black circle), others have large distances in the tree but small distances in the MDS plot (two outlier nodes denoted by a green circle), and nodes of a class in the tree are close to others in the MDS plot (five nodes from the Indian sub-class 2 denoted by a blue circle adjacent to Indian sub-class 1). We jointly investigated the corresponding input distance matrix (Fig. 8 (c)). The heat map rows and columns of the outlier nodes with black circles show that they are close to each other but far from other nodes which is consistent to the tree. Thus, our collaborator concluded that our result more accurately reflects the input data, which we further verified by the stress error [21] in the O^{3}NJ tree and MDS plot (210.4 vs. 564.9).

Based on these experiences, our collaborator plans to use a combination of our O^{3}NJ tree tool and MDS plots to further investigate which genes produce the unexpected outliers in the *East Asian* class. Regarding the small edge length of *Xishuangbanna* class, he hypothesized that the distance measure we used might not reflect the data accurately enough. He thus will try different measures in the future.

6.3. CIFAR-10 Image data

To demonstrate the usefulness of our tool for larger datasets, we conducted a third case study with the *CIFAR-10* image dataset [25] with 10,000 32 $\times$ 32 color images labeled into 10 classes (animals and vehicles). For this case study, we worked together with two deep learning experts, who have more than 5 years of experience in deep learning research. They trained a convolutional neural networks (CNN) and used this model to generate a 64-dimensional

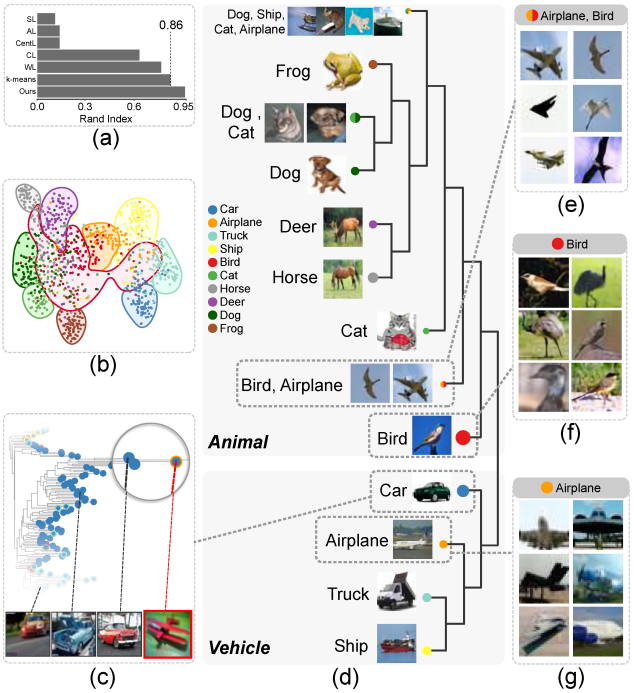

Figure 9. Exploration of *CIFAR-10* Image data with our distilled cluster tree (d). (a) clustering accuracy of several algorithms measured by Rand Index; (b) overlaying k-mean clustering results to the MDS plot, where the lasso color indicates the cluster index and the class label is encoded by the point color; (c) sub-tree of the car class shown in the O^{3}NJ tree with images of one outlier and three selected nodes; (d) distilled cluster tree, where each node is overlaid a pie chart indicating the percentages of the mixed different classes; (e-g) representative images of clusters.

feature vector for each image. The goal of our collaborators was to use O^{3}NJ to validate their CNN model and help answering two core questions: i) which classes can be discriminated from other classes by the model, and which ones not; and ii) learn why it does not work well for some classes.

The generated feature vectors correctly group the images from different classes in a good model. Our collaborators frequently do that by clustering the data, and then check how well the clustering results match with class labels (a typical task in multi-dimensional data analysis [6]. We thus tested this data with a few clustering algorithms such as SL, AL, CentL, WL, k-means (k=10) [31] and our method. We measured the accuracy between the clustering results and the existing class labels using the Rand Index (RI) [38]. RI=0 indicates clustering results do not at all match class labels, and RI=1 they exactly match. Fig. 9(a) shows the RI values of the different methods.

To compare the clustering results generated by k-means (2-nd best method) and our method, we visualized the k-means clustering results in an MDS plot, where the colors of the dots show the ground truth class labels and the

hulls around them show the k-means clustering result (see Fig. 9(b)). Some small classes were well identified, but a large cluster in the middle (with the red hull) is a mix of different elements. In contrast, our distilled cluster tree shown in Fig. 9(d) separates the classes of animals and vehicles well. Opposed to the k-means-based analysis, this indicates that the trained model has enough ability to discriminate these two high-level classes. Meanwhile, our cluster tree revealed three small compound clusters. For those we visualized the nodes with a pie chart to show the percentage of each class. Since we use dots as visual encodings and an accurate judgment of the mixing level is not important, we opted for pie charts to ensure consistency with the interface. Such information is not revealed by k-means clustering. Considering the higher RI measure of our approach, our collaborators concluded that our clustering result is the better solution for their problem.

To verify that the small compound clusters are reasonable, our collaborators explored the tree structure of each class. Fig. 9(c) shows a sub-tree of the car class, where an outlier (airplane) is clearly shown, the three other selected nodes correspond to cars of different poses and colors. This indicates that the feature vectors generated by this model have a strong intra-class discrimination ability. During further exploration they found that the adjacent nodes of the outlier airplane correspond to images with red foreground objects, (third image on the bottom of Fig. 9(c)). They also investigated images from clusters with mixed classes to learn why such images cannot be discriminated by their model. Fig. 9(e) shows six representative images of the cluster mixed from birds and airplanes. The flying birds have very similar shapes to the flying airplanes. Further investigating the bird and airplane clusters (see Fig. 9(f,g)) shows that most images in these two clusters have very different shapes.

Our collaborators also investigated the other two clusters with mixed classes and obtained similar findings. They found their CNN model might pay too much attention to the global shape and too little to the local context and that more local shape information has to be incorporated. In addition, they concluded that the model might give too large weights to the color of some objects (e.g, car). Hence, they want to find better ways to combine different properties. Finally, they confirmed that our method and associated clusters and outliers are more accurate and intuitive than their current practices, and they will thus continue to use it.

7. Conclusion

We have presented a novel visualization representation, O^3NJ tree, for hierarchical analysis of multi-dimensional data. This representation is based on NJ trees, which so far have been mostly used for phylogenetic data analysis in biology. A quantitative comparison between NJ trees and dendrograms produced by various ordinary AHC methods shows that NJ trees characterize the inherent clusters for general multi-dimensional data better than other methods. Orthogonal NJ trees are thus an alternative approach for interactive hierarchical cluster analysis. Since such trees have varying edge lengths, identifying major clusters is hard. To address this issue, we proposed a dedicated ordering algorithm that helps users to better interpret adjacencies and proximities within such trees and a new method to visually distill a cluster tree from an ordered NJ tree. Finally, we demonstrated the benefits of this approach for exploring multi-dimensional data with three case studies.

Limitations: The NJ algorithm is slower than the ordinary AHC method, which hampers the application of our approach for large data analysis. A few state-of-the-art accelerating strategies [33, 46] generate appropriate NJ trees and have to be investigated. Second, our approach might not automatically generate reasonable clustering results for some challenging datasets using the default parameters. This limitation is shared with almost all other clustering approaches. In the future, we plan to explore autotuning strategies [49] on parameters. Third, some potential scalability issues in color may exist in the visual representation of the distilled cluster trees. Finally, our current approach does not allow users to learn why some nodes have unequal edge lengths; such information would be important for domain scientists. Inspired by the rank-by-feature framework [44], we plan to integrate our O^3NJ tree into a multiple-coordinated view system that allows exploring relationships between clusters and data attributes.

Acknowledgement

This work was supported in part by the grants of the NSFC (62141217, 62132017), Shenzhen Basic Research Program (JCYJ20180507182222355), and the CAS grant (GJHZ1862).

References

[1] C. Bachmaier, U. Brandes, and B. Schlieper. *Drawing phylogenetic trees*. Springer, 2005. 4

[2] Z. Bar-Joseph, E. D. Demaine, D. K. Gifford, N. Srebro, A. M. Hamel, and T. S. Jaakkola. K-ary clustering with optimal leaf ordering for gene expression data. *Bioinformatics*, 19(9):1070–1078, 2003. 6

[3] Z. Bar-Joseph, D. K. Gifford, and T. S. Jaakkola. Fast optimal leaf ordering for hierarchical clustering. *Bioinformatics*, 17(suppl 1):S22–S29, 2001. 2, 4, 5, 6

[4] I. Borg and P. J. Groenen. *Modern multidimensional scaling: Theory and applications*. Springer, 2005. 10

[5] G. Bottu. *The phylogenetic handbook: a practical approach to DNA and protein phylogeny*. Cambridge University Press, 2003. 5

[6] M. Brehmer, M. Sedlmair, S. Ingram, and T. Munzner. Visualizing dimensionally-reduced data: Interviews with analysts

and a characterization of task sequences. In *IEEE VIS Workshop Beyond Time and Errors: Novel Evaluation Methods for Visualization (BELIV)*, pages 1–8. ACM, 2014. 11

[7] P. Buneman. A note on the metric properties of trees. *Journal of combinatorial theory, series B*, 17(1):48–50, 1974. 5

[8] M. Burch, N. Konevtsova, J. Heinrich, M. Hoeferlin, and D. Weiskopf. Evaluation of traditional, orthogonal, and radial tree diagrams by an eye tracking study. *TVCG*, 17(12):2440–2448, 2011. 2, 4

[9] M. Chae and J. J. Chen. Reordering hierarchical tree based on bilateral symmetric distance. *PloS one*, 6(8):e22546, 2011. 2, 4, 5

[10] A. M. Cuadros, F. V. Paulovich, R. Minghim, and G. P. Telles. Point placement by phylogenetic trees and its application to visual analysis of document collections. In *Proceedings of the IEEE Symposium on Visual Analytics Science and Technology*, pages 99–106, 2007. 2, 3

[11] W. H. Day and H. Edelsbrunner. Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of classification*, 1(1):7–24, 1984. 1

[12] M. F. De Oliveira and H. Levkowitz. From visual data exploration to visual data mining: a survey. *IEEE Trans. Vis. & Comp. Graphics*, 9(3):378–394, 2003. 1

[13] M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proc. of the National Academy of Sciences*, 95(25):14863–14868, 1998. 2

[14] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231, 1996. 2, 9

[15] R. Etemadpour, R. Motta, J. G. de Souza Paiva, R. Minghim, M. C. F. de Oliveira, and L. Linsen. Perception-based evaluation of projection methods for multidimensional data visualization. *IEEE Trans. Vis. & Comp. Graphics*, 21(1):81–94, 2015. 2, 4

[16] M. Farach, S. Kannan, and T. Warnow. A robust model for finding optimal evolutionary trees. *Algorithmica*, 13(1-2):155–179, 1995. 5

[17] C. Fraley and A. E. Raftery. Mclust: Software for model-based cluster analysis. *Journal of classification*, 16(2):297–306, 1999. 1

[18] P. Fränti and S. Sieranoja. Clustering datasets, 2015. 9

[19] G. Gruvaeus and H. Wainer. Two additions to hierarchical cluster analysis. *British Journal of Mathematical and Statistical Psychology*, 25(2):200–206, 1972. 4

[20] A. Gyulassy, V. Natarajan, V. Pascucci, P.-T. Bremer, and B. Hamann. A topological approach to simplification of three-dimensional scalar functions. *TVCG*, 12(4):474–484, 2006. 9

[21] M. Halle and J.-R. Vergnaud. *An essay on stress*. MIT press, 1990. 11

[22] J. Han, J. Pei, and M. Kamber. *Data mining: concepts and techniques*. Elsevier, 2011. 3

[23] P. D. Hebert, A. Cywinska, S. L. Ball, et al. Biological identifications through dna barcodes. *Proc. of the Royal Society of London B: Biological Sciences*, 270(1512):313–321, 2003. 10

[24] S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967. 3

[25] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical Report, University of Toronto, 2009. 11

[26] F. K. Kuiper and L. Fisher. A monte carlo comparison of six clustering procedures. *Biometrics*, pages 777–783, 1975. 3

[27] S. Landau, M. Leese, D. Stahl, and B. S. Everitt. *Cluster analysis*. John Wiley & Sons, 2011. 5

[28] M. J. McGuffin and J.-M. Robert. Quantifying the space-efficiency of 2d graphical representations of trees. *Information Visualization*, 9(2):115–140, 2010. 3

[29] G. W. Milligan. An examination of the effect of six types of error perturbation on fifteen clustering algorithms. *Psychometrika*, 45(3):325–342, 1980. 3

[30] T. Munzner, F. Guimbretière, S. Tasiran, L. Zhang, and Y. Zhou. Treejuxtaposer: scalable tree comparison using focus+ context with guaranteed visibility. *ACM Trans. on Graph.*, 22(3):453–462, 2003. 2, 4

[31] F. Murtagh and P. Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012. 3, 11

[32] L. Nakhleh, T. Warnow, D. Ringe, and S. N. Evans. A comparison of phylogenetic reconstruction methods on an indo-european dataset. *Transactions of the Philological Society*, 103(2):171–192, 2005. 3, 5

[33] J. G. Paiva, L. Florian, H. Pedrini, G. Telles, and R. Minghim. Improved similarity trees and their application to visual data classification. *IEEE Trans. Vis. & Comp. Graphics*, 17(12):2459–2468, 2011. 2, 3, 12

[34] J. B. Procter, J. Thompson, I. Letunic, C. Creevey, F. Jossinet, and G. J. Barton. Visualization of multiple alignments, phylogenies and gene family evolution. *Nature methods*, 7:S16–S25, 2010. 3

[35] J. Qi, X. Liu, D. Shen, H. Miao, B. Xie, X. Li, P. Zeng, S. Wang, Y. Shang, X. Gu, et al. A genomic variation map provides insights into the genetic basis of cucumber domestication and diversity. *Nature genetics*, 45(12):1510–1515, 2013. 11

[36] P. Rai and S. Singh. A survey of clustering techniques. *International Journal of Computer Applications*, 7(12):1–5, 2010. 5

[37] A. Rambaut, O. G. Pybus, M. I. Nelson, C. Viboud, J. K. Taubenberger, and E. C. Holmes. The genomic and epidemiological dynamics of human influenza a virus. *Nature*, 453(7195):615, 2008. 10

[38] W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971. 11

[39] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4(4):406–425, 1987. 3

[40] R. Sakai, R. Winand, T. Verbeiren, A. V. Moere, and J. Aerts. dendsort: modular leaf ordering methods for dendrogram representations in r. *F1000Research*, 3, 2014. 2, 4, 5

[41] S. Saraçli, N. Doğan, and İ. Doğan. Comparison of hierarchical cluster analysis methods by cophenetic correlation. *Journal of Inequalities and Applications*, 2013(1):203, 2013. 3

[42] P. Sebastian, H. Schaefer, I. R. Telford, and S. S. Renner. Cucumber (cucumis sativus) and melon (c. melo) have numerous wild relatives in asia and australia, and the sister species of melon is from australia. *Proc. of the National Academy of Sciences*, 107(32):14269–14273, 2010. 10

[43] J. Seo and B. Shneiderman. Interactively exploring hierarchical clustering results. *IEEE Computer*, 35(7):80–86, 2002. 1, 3

[44] J. Seo and B. Shneiderman. A rank-by-feature framework for interactive exploration of multidimensional data. *Information visualization*, 4(2):96–113, 2005. 12

[45] B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *TOG*, 11(1):92–99, 1992. 3

[46] M. Simonsen, T. Mailund, and C. N. Pedersen. Rapid neighbour-joining. In *WABI*, volume 8, pages 113–122. Springer, 2008. 12

[47] R. R. Sokal and F. J. Rohlf. The comparison of dendrograms by objective methods. *Taxon*, 11(2):33–40, 1962. 1, 3

[48] T. Stefan Van Dongen and B. Winnepenninckx. Multiple upgma and neighbor-joining trees and the performance of some computer packages. *Mol. Biol. Evol*, 13(2):309–313, 1996. 3, 5

[49] M. Sugiyama, M. Yamada, M. Kimura, and H. Hachiya. On information-maximization clustering: Tuning parameter selection and analytic solution. In *Proc. of the 28th International Conference on Machine Learning*, pages 65–72, 2011. 12

[50] T. Weinkauf and D. Günther. Separatrix persistence: Extraction of salient edges on surfaces using topological methods. *Computer Graphics Forum*, 28(5):1519–1528, 2009. 9

[51] Y. Zhao and G. Karypis. Evaluation of hierarchical clustering algorithms for document datasets. In *Proc. of the eleventh international conference on Information and knowledge management*, pages 515–524. ACM, 2002. 1, 3