# Optimally Ordered Orthogonal Neighbor Joining Trees for Hierarchical Cluster Analysis

Tong Ge, Xu Luo, Yunhai Wang, Michael Sedlmair, Zhanglin Cheng, Ying Zhao, Xin Liu, Oliver Deussen and Baoquan Chen

**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. We optimize NJ trees for their use in visual analysis in two ways. First, we propose to use a novel leaf sorting algorithm that helps users to better interpret adjacencies and proximities within such a tree. Second, we provide a new method to visually distill the cluster tree from an ordered NJ tree. Numerical evaluation and three case studies illustrate the benefits of this approach for exploring multi-dimensional data in areas such as biology or image analysis.

**Index Terms**—Neighbor Joining, Leaf Ordering, Orthogonal Layout

## 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 [13] 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 [54]. seeks to build a cluster hierarchy that enables users to explore possible clusterings at different levels. A commonly used method is Agglomerative Hierarchical Clustering (AHC) [12] 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.

- *Yunhai Wang, Xu Luo, Tong Ge are with the Department of Computer Science, Shandong University, China. E-mail: {tgeconf,cloudseawang, luoxu9days}@gmail.com*
- *Michael Sedlmair is with University of Stuttgart, Stuttgart, Germany. E-mail: michael.sedlmair@visus.uni-stuttgart.de*
- *Zhanglin Cheng is with SIAT, Shenzhen, China. E-mail: zl.cheng@siat.ac.cn*
- *Ying Zhao is with Central South University, China. E-mail: zhaoying@csu.edu.cn*
- *Xin Liu is with Beijing Genomics Institute BGI Research, China. E-mail:liuxin@genomics.cn*
- *Baoquan Chen is with Peking University, Beijing, China. E-mail: baoquan@pku.edu.cn*
- *Oliver Deussen is with University of Konstanz, Konstanz, Germany. E-mail: oliver.deussen@uni-konstanz.de*
- *T. Ge and X. Luo are the joint first author and Yunhai Wang is the corresponding author*

To cluster a given dataset, users move a "similarity bar" over the dendrogram (dashed black line in Fig. 1(d)), and while doing so they interactively "cut" the dendrogram into pieces to find a proper number of clusters. Dendrograms originated from biology [50] for clustering genes, we refer to them more generally as *HC trees* (hierarchical clustering trees).

HC trees have been successfully applied in various visual analysis projects [45], and are part of standard systems for data analysis [18]. However, they have two drawbacks that limit their effectiveness. First, all leaves in a HC tree 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 within general high dimensional data. An example is shown in Fig. 1(d)), where each node of the red cluster has a different distance 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 all the same. This indicates that HC trees do not represent data distances well. As a result, it is hard to see outliers among the leaf nodes of a HC tree (see the two outliers enclosed by the red circle in Fig. 1(c)). Second, HC trees are not able to reveal the *intrinsic cluster structure* of many datasets, since in every step the two closest clusters are merged. Such an approach might fail in cases when clusters are not linearly separable [15], as it does not pay attention to inter-cluster distances. As shown in Fig. 1(b) and (d), the orange cluster is incorrectly merged with the green cluster, although the red and blue clusters are separated. Moreover, organizing clusters into a binary tree might be improper for datasets having three or more major clusters.

To mitigate these problems, we propose to use orthogonal neighbor-joining (NJ) trees for interactive hierarchical cluster analysis. In contrast to HC trees, only a few approaches use NJ trees for multi-dimensional data visualization [11], [16], [35], [52]. These techniques mainly use such trees as a projection technique and show their results subsequently

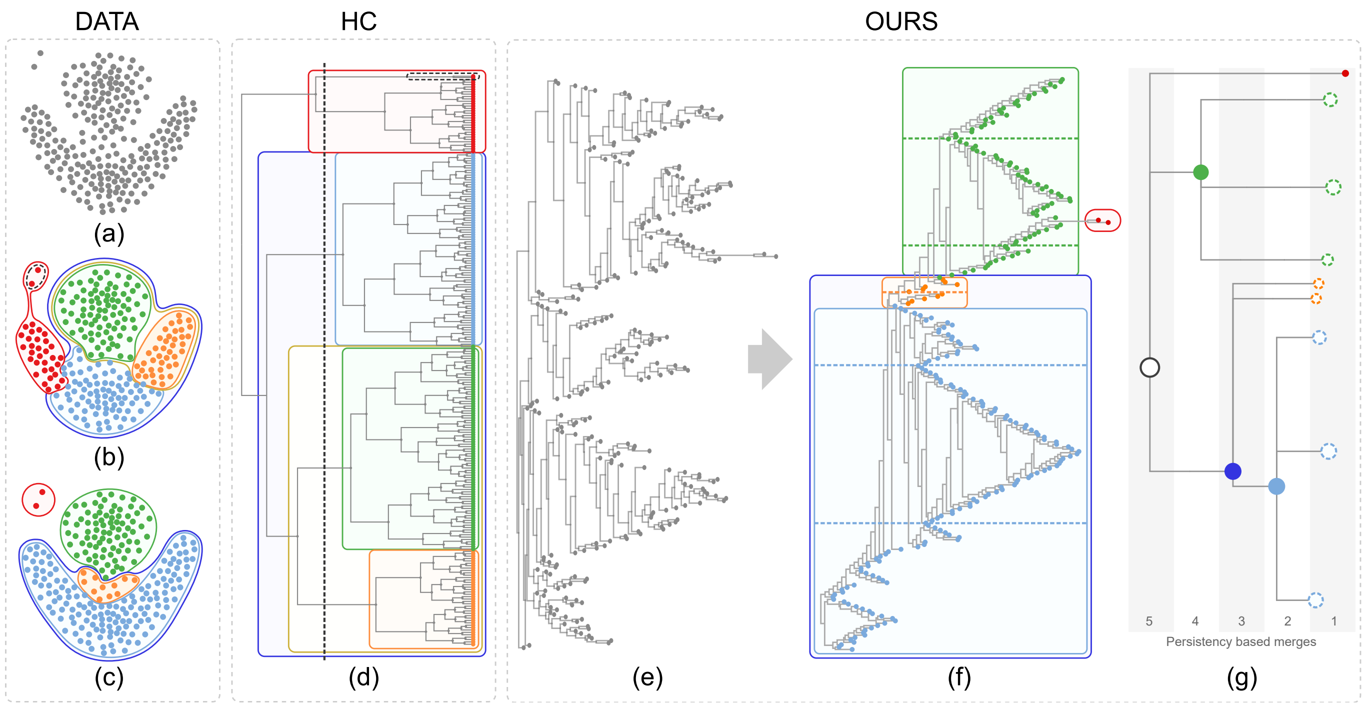


Fig. 1. A given dataset (a) is better clustered by our optimally ordered orthogonal neighbor joining tree ($O^3$NJTree) (f) than by a dendrogram (d) produced by the hierarchical clustering (HC) method with complete linkage. Subfigure (b) shows the nested five clusters generated by cutting the dendrogram 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.

with 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], [32]. 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 the 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 so as to clearly reveal clusters. Such trees look like *terrains* with peaks and valleys (see Fig. 1(f)). We call these representations *Optimally Ordered Orthogonal Neighbor Joining* ($O^3$NJ) Trees. While a number of leaf ordering methods have been proposed for HC trees [3], [10], [14], [42], such methods cannot produce an optimal ordering for NJ trees due to 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 the root, and then performs a persistence-inspired 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, our main contributions are as follows:

- 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;
- We introduce an algorithm to extract the cluster structure from $O^3$NJ trees; and
- We demonstrate the usefulness of our approach for exploring multi-dimensional data by conducting a numerical evaluation and 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. [23] 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 [25]. 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 [33].

Each linkage criterion and associated AHC method has its advantages and disadvantages [54]. Previous studies show that average linkage, centroid linkage, as well as Ward's linkage are quite sensitive to shape and size of clusters [27], [31]. 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 works [27], [31], [43] quantitatively compared these methods using numeric measures such as the cophenetic correlation coefficient [50] 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 [41], [52]. The 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 [34], [51].

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 [11], [35]. 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 datasets. In this paper we show how this tree structure helps to examine hierarchical cluster analysis of large datasets.

## 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 [47]. McGuffin and Robert [30] systematically compare the space efficiency of these methods and provide guidelines for choosing a good representation. In this paper 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) [37], 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 [45] enhances HC trees with dynamic query controls for interactive exploration. Munzner presented tree-juxtaposer [32] which allows users to navigate large hierarchies with a global rectangular focus+context technique.

Etemadpour et al. [16] 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 [32]. Thus, in this paper we investigate how orthogonal tree layouts can be used to support visual analysis of NJ trees.

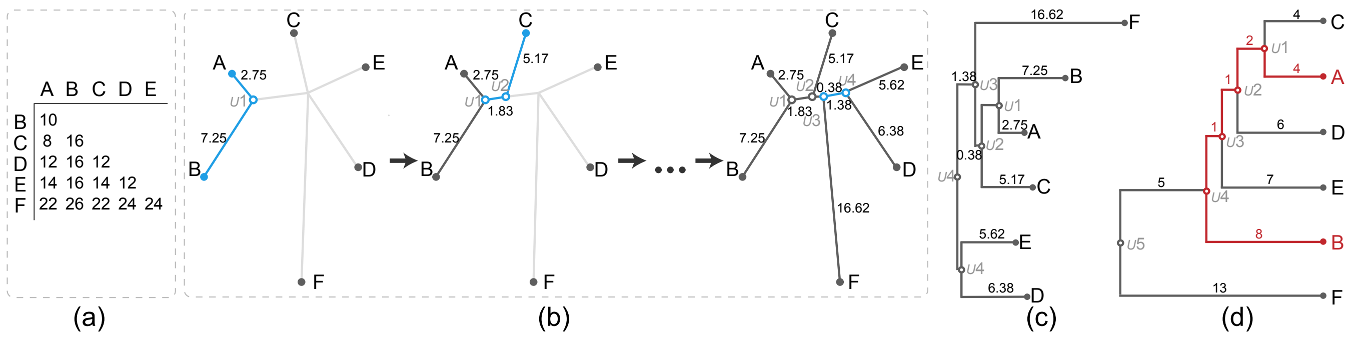
## 2.3 Leaf Ordering for Hierarchical Clustering

For an orthogonal tree diagram, leaf nodes are shown in a linear order along one axis. Adjacent leaves in such a linear ordering are assumed to be related [20], 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], [10], [42] that minimize distances of adjacent leaves.

The classic method was proposed by Bar-Joseph et al. [3] and formulates leaf ordering as a dynamic programming problem, which allows to produce good solutions in reasonable time. Later, the authors extend and improve this algorithm for $k$-ary trees, in which each internal node has at most $k$ children. Rather than only considering the distance between two adjacent leaves, Chae et al. [10] propose to order the leaves based on the bilateral symmetric distance between each two adjacent clusters such that similar objects in the clusters are located at the cluster boundaries. Recently, Sakai et al. [42] further incorporate the orientation of clusters at each merge step into leaf ordering.

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 center cluster is interrupted by three outliers (B, K and E). So far, almost all existing NJ tree visualizations arrange the order of leaf nodes randomly.

To bridge this gap, we propose an efficient leaf ordering algorithm that finds a linear ordering in which the sum of



| | A | B | C | D | E |
|---|---|---|---|---|---|
| B | 10 | | | | |
| C | 8 | 16 | | | |
| D | 12 | 16 | 12 | | |
| E | 14 | 16 | 14 | 12 | |
| F | 22 | 26 | 22 | 24 | 24 |

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

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 classic neighbor joining algorithm [41] 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 $Q$, the NJ algorithm takes into account intra-cluster compactness as well as inter-cluster separation.

Fig. 2 illustrates the algorithm with the input distance matrix. 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 is generated that fits to the actual distance in data space.

**Rooting Strategy** As already mentioned (and shown in Fig. 2(b)), an NJ tree does not have a root by default. However, biologists often explore NJ trees using orthogonal layouts, so a root is needed. To address this issue, we follow a rooting strategy given in Bottu [5] that simply duplicates the last formed internal node to create the root node (U4 in 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 [25] 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 [34], [51]; however, most of these studies are based on biological data. To verify that NJ trees also are better in representing general high dimensional data, we quantitatively compared them against 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 datasets of different sizes 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 datasets. 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 $D_{ij}$ [17], 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 [28].

## 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 the closeness of adjacent leaf nodes visually indicates the 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], [10], [42], 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 from determining 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 OLO algorithm for NJ trees (OLONJ) 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 separated well.

### 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 $\phi$ that minimizes the sum of absolute path length differences between adjacent leaves $\phi_i$ and $\phi_{i+1}$:

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

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

**Algorithm 1** Optimal Leaf Ordering of NJ tree

```
1:  function OLONJ($(u, D)$)
2:      if $|u| == 1$ then
3:          $C(u, v, v) = 0$ return $C(u, v, v)$
4:      else
5:          $C(u_l, L, R) = OLONJ(u_l, D)$
6:          $C(u_r, L, R) = OLONJ(u_r, D)$
7:          for $v$ in leaves of $u_l$ do // left subtree of $u$
8:              for $w$ in leaves of $u_r$ do // right subtree of $u$
9:                  if $D_{m,k} \leq t$ then
10:                     $C(u, v, w) =$
11:                     $\min_{m \in u_l, k \in u_r} C(u_l, v, m) + C(u_r, k, w)$
12:                         $+|p_m - p_k|$
13:                 else
14:                     $C(u, v, w) = +\infty$
15:                 end if
16:                 $C(u, w, v) = C(u, v, w)$
17:             end for
18:         end for
19:     end if
20:     return $C(u, L, R)$ // $L$ and $R$ denote all possible
21:                         // pairs of leaves from $u_l$ and $u_r$
22: end function
```

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

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

Since the path length in the HC tree represents the distances between data points, Eq. (5) can be written as:

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

which is the sum of path length between adjacent leaf nodes. Since it is substantially different from Eq. 4, applying this OLO algorithm (Eq. 5) to the NJ tree produces undesired results. For example, the ordering separates node I from the clusters in Fig. 3(e), but places the yellow cluster in the middle of the blue cluster. In contrast, minimizing Eq. 4 clearly reveals three clusters in Fig. 3(f), where all nodes in the blue cluster are arranged together.

### 4.2 Dynamic Programming

Like the OLO algorithms [2], [3] 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

| | $A$ | $B$ | $C$ | $D$ | $E$ | $F$ | $G$ | $H$ | $I$ | $J$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $B$ | 23 | | | | | | | | | |
| $C$ | 11 | 34 | | | | | | | | |
| $D$ | 17 | 33 | 12 | | | | | | | |
| $E$ | 19 | 12 | 31 | 35 | | | | | | |
| $F$ | 15 | 18 | 21 | 16 | 22 | | | | | |
| $G$ | 6 | 18 | 16 | 18 | 16 | 10 | | | | |
| $H$ | 17 | 25 | 19 | 10 | 29 | 7 | 15 | | | |
| $I$ | 29 | 45 | 22 | 12 | 47 | 27 | 30 | 19 | | |
| $J$ | 9 | 22 | 13 | 11 | 23 | 7 | 7 | 8 | 23 | |
| $K$ | 27 | 9 | 35 | 32 | 20 | 15 | 20 | 22 | 41 | 22 |

(a)

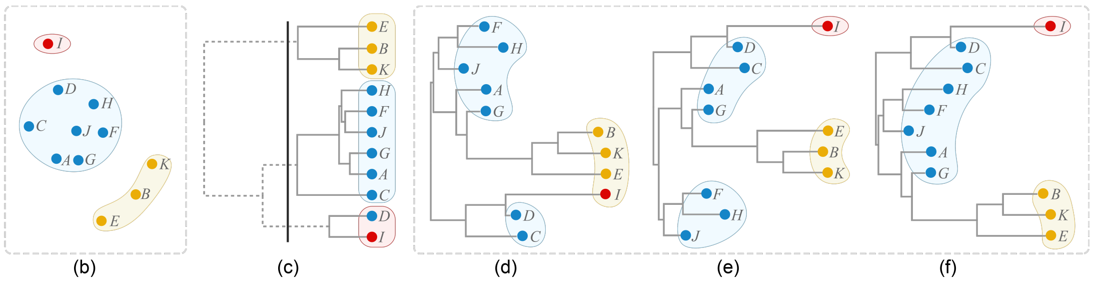


Fig. 3. Comparison between ordinal hierarchical clustering (HC) and NJ trees generated from the same distance matrix as shown in (a). (b) Scatter plot with three clusters produced by multi-dimensional scaling; (c) HC tree where positions do not accurately resemble distances, e.g., the distance between 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 an R package [36]; (e) NJ tree ordered by using the OLO algorithm [3]; (f) NJ tree ordered by our algorithm, three useful clusters are created.

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 $\mu_l$ and $k$ is the leftmost leaf of $\mu_r$. 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 $C(u)$.

**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 computation time:

- compute the minimum path length difference $minP = \min_{v \in u_l, w \in u_r} |p_v - p_w|$;
- 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 $\mu_r$.

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.

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

```
1: $minP = \min_{v \in u_l, w \in u_r} |p_v - p_w|$
2: $curC = +\infty$
3: for $m$ in ordered $C(u_l, v, m)$ do
4:     if $C(u_l, v, m) + C(u_r, k_0, w) + minP \geq curC$ then
5:         $C(u, v, w) = curC$; break
6:     end if
7:     for $k$ in ordered $C(u_r, k, w)$ do
8:         if $C(u_l, v, m) + C(u_r, k, w) + minP \geq curC$ then
9:             break
10:        end if
11:        if $D_{m,k} < t$ then
12:            $tmp = C(u_l, v, m) + C(u_r, k, w) + |p_m - p_k|$
13:        else
14:            $tmp = +\infty$
15:        end if
16:        if $curC > tmp$ then
17:            $curC = tmp$
18:        end if
19:    end for
20: end for
21: $C(u, v, w) = curC$
```

**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

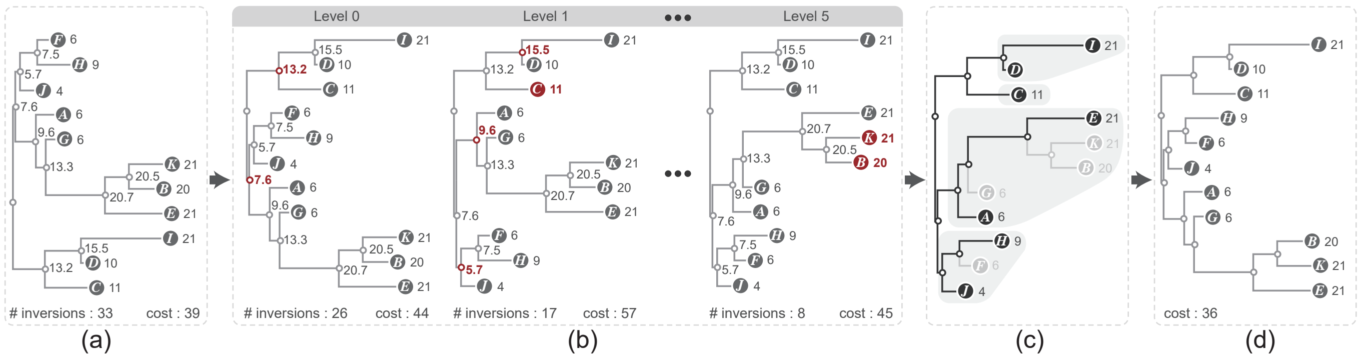


Fig. 4. Overview of our leaf ordering algorithm: (a) an orthogonal NJ tree with unequal edge length where each node is assigned a weight based on the edge length; (b) by performing monotonicity based flipping within a breadth-first traversal the inversion is reduced; (c) extraction of the ordered sub-trees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

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 $\mu$, 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.
- 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 $i$th 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 with 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. 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 value as outliers if their values lie above a threshold:

$$\omega = q75 + 1.5(q75 - q25) \tag{8}$$

where $q75$ and $q25$ are the 75th and 25th percentile of the ARD values. Once we defined $\omega$ 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.

**Persistence-inspired 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-inspired topological simplification by Weinkauf et al. [53] to remove such noisy

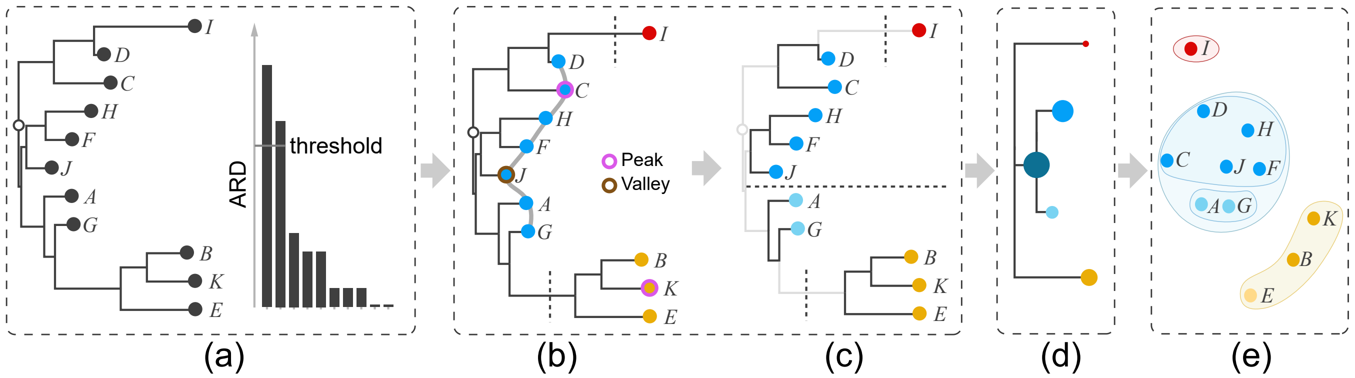


Fig. 5. Distilling the cluster tree from an O$^3$NJ tree includes splitting and hierarchy extraction: (a) input O$^3$NJ tree and a histogram of the ARD values; (b) the selected threshold from the histogram in (a) cuts the tree into three parts; (c) identified peaks and valleys as well as corresponding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

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 $\beta$ is reached. Since most persistence values are quite small (see Fig. 6(b)), we set $\beta$ to a value of 10% of the maximal persistence following the suggestion of Gyulassy et al. [21]. As shown in Fig. 6(c), this default threshold removes the most noisy peaks and valleys, while resulting in meaningful clusters.

## 6 Evaluation

We implemented O$^3$NJ tree in Javascript and created a functioning interface, which is available on GitHub[1]. Two aspects of the effectiveness of our method were evaluated: its clustering accuracy with default settings and the insights revealed by interactive exploration of the distilled cluster trees. Hence, we performed a quantitative comparison with state-of-the-art methods and two case studies using real datasets. All the experiments were done on a Windows desktop computer with an Intel Core i7-8700K processor with 16GB memory. Since the time complexity of our OLONJ algorithm is $O(n^4)$, ordering the NJ tree shown in Fig. 1 takes around 500 ms, whereas generating the tree takes less than 50 ms.

### 6.1 Quantitative Evaluation

We quantitatively compared our method against three widely used clustering methods: the AHC method with complete linkage, k-means clustering, and the robust continuous clustering (RCC) method [46], which correspond to three different types of methods: nonparametric methods, center-based methods, and continuous objective-based methods without requiring prior knowledge about cluster numbers. The default value of parameter $\omega$ is defined in Eq. 8, the values of the thresholds $t$ and $\beta$ are half of the average data distance and 10% of the maximal data distance and persistence, respectively.

**Datasets.** We collected 52 labeled datasets of different sizes and dimensionalities having substantial variations in terms of

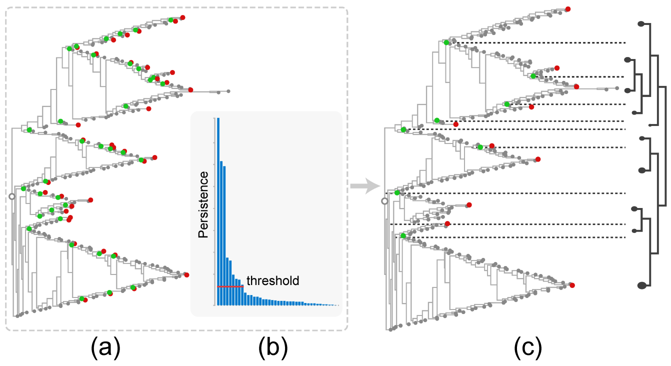


Fig. 6. Persistence-inspired simplification of an O$^3$NJ tree: the optimally ordered version of Fig. 1(e) is the input shown in (a), a default threshold of 10% of the maximal persistence value shown in the persistence histogram in (b) removes most noisy peaks and valleys and results in the cluster hierarchy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

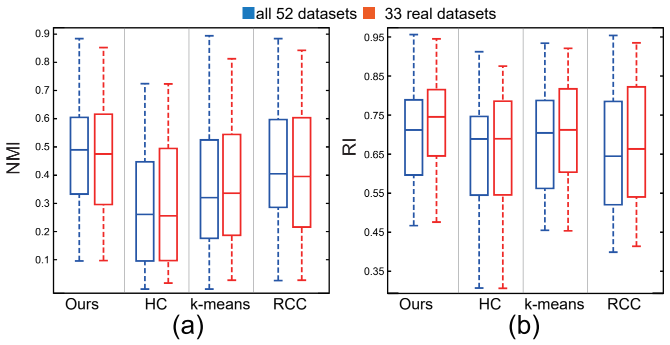


Fig. 7. Values of NMI (a) and RI (b) for the four different clustering methods: the blue boxplots show the score distributions over *all* 52 datasets and red boxplots give the score of the 33 *real* datasets.

data size (ranging from 24 to 1473) and data dimensionality, (ranging from 2 to 617). Among them, 19 synthetic datasets are often used for the evaluation of clustering methods [19], and the other 33 real datasets are coming from the UCI repository [29]. For these datasets, the number of classes is taken as input for k-means, while for our method and AHC

1. https://github.com/Ideas-Laboratory/O3NJ

they are used to guide the cut for the cluster trees.

**Measures.** We chose Normalized Mutual Information (NMI) [49] and Rand Index (RI) [40], as measures, both have been widely used for evaluating the accuracy of various clustering techniques. NMI is a normalization of the mutual information (MI) score which measures the similarity between cluster labels and class labels, while RI measures the likelihood that a pair of points is either in the same cluster or in different clusters for two clustering results. Hence, the range of both measures is between 0 and 1, where 0 indicates that the cluster results do not agree with the class label and 1 indicates that the cluster results are exactly the same as the class labels. Note that the clustering method does not return class labels, thus we need to find the best one-to-one match between our cluster labels and the ground truth.

**Results.** All individual NMI and RI values generated by the four different clustering methods for each dataset, as well as screenshots of all the distilled cluster trees and AHC cluster trees can be found in the supplemental material. To facilitate the comparison between different clustering methods, we summarize the NMI and RI scores over *all 52 datasets* and over the *33 real datasets*. The results are shown by the blue and red boxplots in Figures 7(a,b).

On average, our method produces better NMI and RI scores than the other methods. In terms of NMI, our method shows its benefits compared to the others, while HC performs the worst. However, the RI scores of the best cases of all four methods are very close. Comparing the red boxplots with the blue ones, we can see that all methods have smaller variances for the real datasets, while our method has the smallest one. In sum, our method works better than the tested existing methods and works well for most real datasets. An example is shown in Fig. 8, where our distilled cluster tree clearly separates the classes J and C, while the HC tree does not do that.

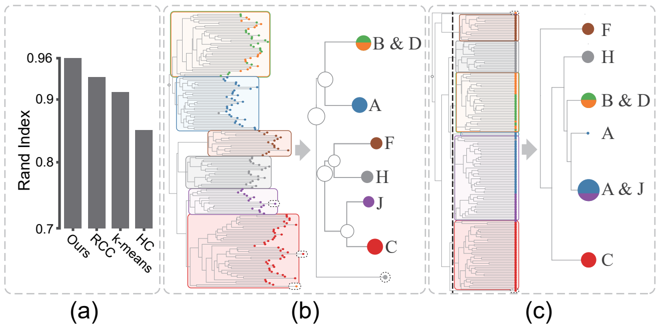


Fig. 8. (a) Clustering accuracy of the *Isolet* data generated by different methods, (b) cluster trees produced by our method (b) and the AHC method (c).

## 6.2 Case Studies

**Cucumber Data.** In evolutionary analysis [24], [39], 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 method not only combines these two functionalities, but also produces more accurate results as illustrated by the following case study using a *cucumber* genome dataset, provided by one of our collaborators, a biologist, who has more than 10 years of experience in evolutionary analysis.

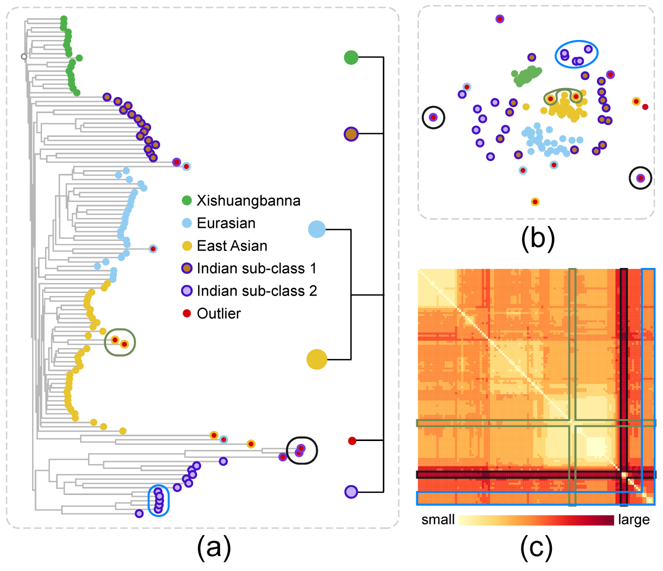


Fig. 9. Exploration of the *Cucumber* data: (a) $O^3$NJ tree and distilled hierarchy; (b) MDS plot; (c) heat map showing the input distance matrix. Three kinds of inconsistencies between our $O^3$NJ tree representation and the MDS results are highlighted by black, green and blue circles in the $O^3$NJ tree, the corresponding nodes are also highlighted in the MDS plot and the heap map. Class label and cluster index of each node are encoded by border and fill color.

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 [44], 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 produced five major clusters and one outlying cluster. The $O^3$NJ tree visualizes this inconsistency by encoding the class label and cluster index of the corresponding nodes into their border and fill color (see Fig. 9 (a)).

After examining the tree in Fig. 9 (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 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 for other classes and ii) have fewer variations. He did not expect this because the *Xishuangbanna* class uniquely accumulates $\beta$-carotene in its fruit [38]. 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 the O$^3$NJ tree was correct by comparing it to an MDS projection of the input distance matrix (see Fig. 9 (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 the O$^3$NJ tree shown on the right of Fig. 9 (a). Some nodes have small distances in the tree but large distances in the MDS plot (see the two outliers denoted by a black circle), others have large distances in the tree but small distances in the MDS plot (the two outlier nodes denoted by a green circle). Finally, some nodes of a class in the tree are close to others in the MDS plot (the five nodes from the Indian sub-class 2 are denoted by a blue circle adjacent to Indian sub-class 1).

We jointly investigated the corresponding input distance matrix (Fig. 9 (c)). The rows and columns of the heat map that correspond to 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 measuring the stress error [22] 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 O$^3$NJ trees and MDS plots to further investigate, which genes produce the unexpected outliers in the *East Asian* class. Regarding the small edge lengths of the *Xishuangbanna* class, he hypothesized that the distance measure we used might not reflect the data accurately enough. He will verify this by using different measures in the future.

**CIFAR-10 Image data.** To demonstrate the usefulness of our tool for larger datasets, we conducted a second case study with the *CIFAR-10* image dataset [26] with 10,000 $32 \times 32$ color images labelled into 10 classes (different animals and vehicles). For this case study, we worked together with two deep learning experts, who have more than five years of experience in deep learning research. They trained a convolutional neural network (CNN) and used this model to generate a 64-dimensional feature vector for each image. The goal of our collaborators was to use an O$^3$NJ-tree to validate their CNN model and help answer two core questions: i) which classes can be discriminated from others by the model and which ones not; and ii) why this does not work well for some classes.

The generated feature vectors correctly group the images from different classes to a good model. Our collaborators frequently do that by clustering the data, and then check how well the clustering results match with the class labels (a typical task in multi-dimensional data analysis [6]). We hence tested this dataset with the four clustering algorithms

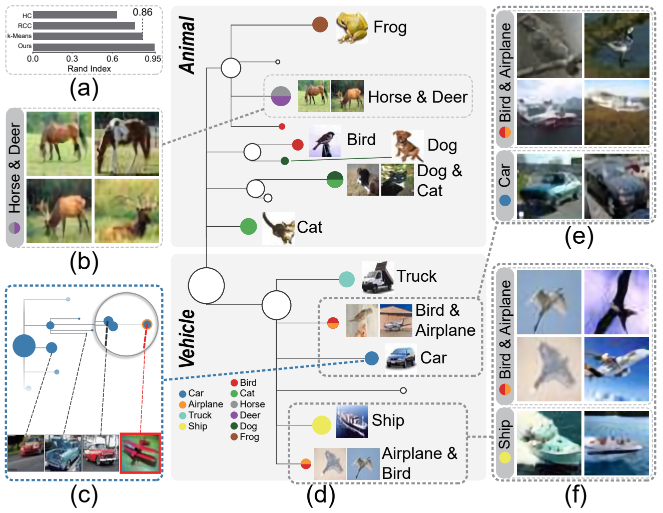


Fig. 10. Exploration of the *CIFAR-10* image data with our distilled cluster tree: (a) clustering accuracy of several algorithms measured by the Rand Index; (b) four representative images of the cluster consisting of Horse and Deer classes; (c) sub-tree of the Car class shown in the O$^3$NJ tree with images of one outlier and three selected nodes; (d) the distilled cluster tree, where each node is associated with a class label; (e) representative images of two clusters: Bird & Airplane and Car class; (f) representative images of two clusters: Bird & Airplane and Ship class.

used in Section 6.1 and measured the accuracy between the clustering results and the existing class labels using the RI. Fig. 10(a) shows the RI values of the different methods, where our method performs the best, its improvement over the second best is around 10%.

As shown in Fig. 10(d), two major clusters: animals and vehicles are separated except that a subset of the bird class is mixed with the airplane class. In the animal cluster, the Frog and Bird classes are clearly separated and the Truck, Car and Ship classes are well discriminated in the vehicle cluster, whereas each of the other classes forms compound clusters. To clearly indicate the overlapping degree of these clusters, we visualize the percentages of mixed classes with a pie chart as an overlay for each node.

To verify that the compound clusters are reasonable, our collaborators explored the tree structure for each class. Fig. 10(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. 10(c)). They also investigated images from clusters with mixed classes to learn why such images cannot be discriminated by their model. Fig. 10(e) shows six representative images of the cluster mixed from birds and airplanes. Flying birds have very similar shapes to flying airplanes. Further investigating the bird and airplane clusters (see Fig. 10(e,f)) 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, cars). Hence, they want to find better ways to combine different properties. Finally, they confirmed that our method and associated clusters as well as the outliers are more accurate and intuitive than their current practices, so they will continue to use it.

## 7 Discussion

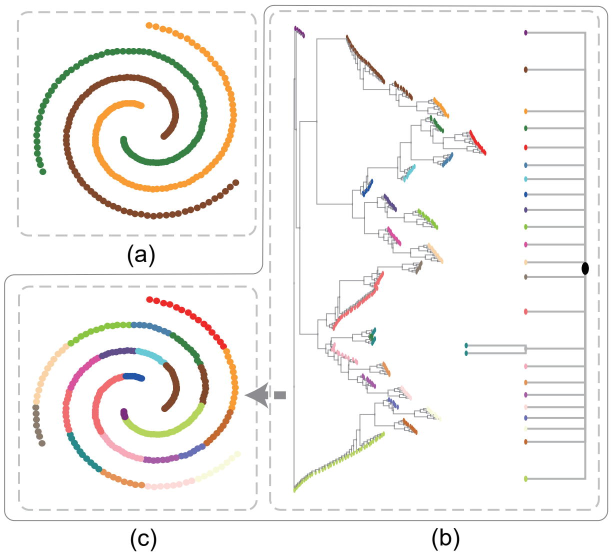


Fig. 11. A failure case of our O$^3$NJtree. (a) the ground truth; (b) An input O$^3$NJ tree and the distilled cluster tree; (c) our clustering result.

While O$^3$NJ trees provide a superior overview of clustering results with respect to other methods, there are still some drawbacks. First, minimizing the differences of root-leaf path lengths of adjacent leaves, instead of the distances of points in the data space, may cause additional distortion. For example, some leaves might be in the opposite direction to the root and might be mis-grouped together. Although most of such leaves can be filtered by the hard distance threshold $t$, we plan to incorporate the distance between adjacent nodes in data space into Eq. 4.

Second, O$^3$NJ trees still fail to reveal some clusters because the root-leaf distances do not capture the relative differences between clusters for some datasets. For instance, the three curves shown in Fig. 11 can be separated easily by the distance from each other, but our O$^3$NJ tree over-segments each curve and groups multiple segments from different curves to the same cluster. For analyzing such data, we recommend using manifold learning algorithms [9].

Third, our approach introduces three parameters: the distance threshold $t$, the ARD value $\omega$, and the persistence threshold $\beta$, which heavily influence the clustering results. The default values of these parameters might not automatically generate reasonable clustering results for some challenging datasets. Although our linked view interface enables interactive clustering by tuning these parameters, it is an indirect interaction and might be misleading. In the future, we therefore plan to explore guidelines for setting up these parameters to control the level of detail (LoD) for the O3NJ trees according to the requirements of visual readability and space limitation.

Last, the NJ algorithm is slower than the ordinary AHC method and shows worse spatial efficiency than HC trees, which hampers the application of our approach for large data sets. Accelerating strategies [35], [48] generate appropriate NJ trees and have to be investigated if we can apply them to our trees. To improve the spatial efficiency, we will combine our ordering technique with radial NJ trees together to reduce the required space.

## 8 Conclusion

We presented a novel visualization representation, O$^3$NJ trees, 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 by a quantitative evaluation and three case studies.

## Acknowledgments

The authors like to thank the anonymous reviewers for their valuable input, this work was supported by the grants of the National Key R&D Program of China (2022ZD0160805), NSFC (No. 62132017, 62141217), Shandong Provincial Natural Science Foundation (No. ZQ2022JQ32) as well as by the DFG (German Research Foundation) under Germany's Excellence Strategy-EXC2117-422037984 and the Project-ID 251654672 – TRR 161.

## References

[1] C. Bachmaier, U. Brandes, and B. Schlieper. *Drawing phylogenetic trees*. Springer, 2005. DOI: 10.1007/11602613_110
[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. DOI: 10.1093/bioinformatics/btg030
[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. DOI: 10.1093/bioinformatics/17.supp_1.S22
[4] I. Borg and P. J. Groenen. *Modern multidimensional scaling: Theory and applications*. Springer, 2005.
[5] G. Bottu. *The phylogenetic handbook: a practical approach to DNA and protein phylogeny*. Cambridge University Press, 2003.
[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)*, pp. 1–8. ACM, 2014. DOI: 10.1145/2669557.2669559
[7] P. Buneman. A note on the metric properties of trees. *Journal of combinatorial theory, series B*, 17(1):48–50, 1974. DOI: 10.1016/0095-8956(74)90047-1

[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. *IEEE Trans. Vis. & Comp. Graphics*, 17(12):2440–2448, 2011. DOI: 10.1109/TVCG.2011.193

[9] L. Cayton. Algorithms for manifold learning. *Univ. of California at San Diego Tech. Rep*, 12(1-17):1, 2005.

[10] M. Chae and J. J. Chen. Reordering hierarchical tree based on bilateral symmetric distance. *PloS one*, 6(8):e22546, 2011. DOI: 10.1371/journal.pone.0022546

[11] 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*, pp. 99–106, 2007. DOI: 10.1109/VAST.2007.4389002

[12] W. H. Day and H. Edelsbrunner. Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of classification*, 1(1):7–24, 1984. DOI: 10.1007/BF01890115

[13] 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. DOI: 10.1109/TVCG.2003.1207445

[14] 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. DOI: 10.1073/pnas.95.25.14863

[15] 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, pp. 226–231, 1996.

[16] 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. DOI: 10.1109/TVCG.2014.2330617

[17] M. Farach, S. Kannan, and T. Warnow. A robust model for finding optimal evolutionary trees. *Algorithmica*, 13(1-2):155–179, 1995. DOI: 10.1007/BF01188585

[18] C. Fraley and A. E. Raftery. Mclust: Software for model-based cluster analysis. *Journal of classification*, 16(2):297–306, 1999. DOI: 10.1007/s003579900058

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

[20] G. Gruvaeus and H. Wainer. Two additions to hierarchical cluster analysis. *British Journal of Mathematical and Statistical Psychology*, 25(2):200–206, 1972. DOI: 10.1111/j.2044-8317.1972.tb00491.x

[21] A. Gyulassy, V. Natarajan, V. Pascucci, P.-T. Bremer, and B. Hamann. A topological approach to simplification of three-dimensional scalar functions. *IEEE Trans. Vis. & Comp. Graphics*, 12(4):474–484, 2006. DOI: 10.1109/TVCG.2006.57

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

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

[24] 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. DOI: 10.1098/rspb.2002.2218

[25] S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967. DOI: 10.1007/BF02289588

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

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

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

[29] M. Lichman. UCI machine learning repository, 2013.

[30] M. J. McGuffin and J.-M. Robert. Quantifying the space-efficiency of 2d graphical representations of trees. *Information Visualization*, 9(2):115–140, 2010. DOI: 10.1057/ivs.2009.4

[31] G. W. Milligan. An examination of the effect of six types of error perturbation on fifteen clustering algorithms. *Psychometrika*, 45(3):325–342, 1980. DOI: 10.1007/BF02293907

[32] 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. DOI: 10.1145/1201775.882291

[33] F. Murtagh and P. Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012. DOI: 10.1002/widm.53

[34] 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. DOI: 10.1111/j.1467-968X.2005.00149.x

[35] 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. DOI: 10.1109/TVCG.2011.212

[36] E. Paradis and K. Schliep. ape 5.0: an environment for modern phylogenetics and evolutionary analyses in R. *Bioinformatics*, 35(3):526–528, 2019. DOI: 10.1093/bioinformatics/bty633

[37] 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. DOI: 10.1038/nmeth.1434

[38] 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. DOI: 10.1038/ng.2801

[39] 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. DOI: 10.1038/nature06945

[40] W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971. DOI: 10.1080/01621459.1971.10482356

[41] 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. DOI: 10.1093/oxfordjournals.molbev.a040454

[42] 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. DOI: 10.12688/f1000research.4784.1

[43] 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. DOI: 10.1186/1029-242X-2013-203

[44] 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. DOI: 10.1073/pnas.1005338107

[45] J. Seo and B. Shneiderman. Interactively exploring hierarchical clustering results. *IEEE Computer*, 35(7):80–86, 2002. DOI: 10.1109/MC.2002.1016905

[46] S. A. Shah and V. Koltun. Robust continuous clustering. *Proceedings of the National Academy of Sciences*, 114(37):9814–9819, 2017. DOI: 10.1073/pnas.1700770114

[47] B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Trans. on Graph.*, 11(1):92–99, 1992. DOI: 10.1145/102377.115768

[48] M. Simonsen, T. Mailund, and C. N. Pedersen. Rapid neighbour-joining. In *WABI*, vol. 8, pp. 113–122. Springer, 2008. DOI: 10.1007/978-3-540-87361-7_10

[49] N. Slonim and N. Tishby. Agglomerative information bottleneck. In *Advances in neural information processing systems*, pp. 617–623, 2000.

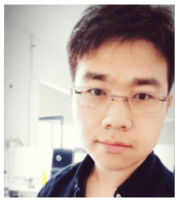
[50] R. R. Sokal and F. J. Rohlf. The comparison of dendrograms by objective methods. *Taxon*, 11(2):33–40, 1962. DOI: 10.2307/1217208

[51] 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. DOI: 10.1093/oxfordjournals.molbev.a025590

[52] G. P. Telles, G. S. Araujo, M. E. Walter, M. M. Brigido, and N. F. Almeida. Live neighbor-joining. *BMC Bioinformatics*, 19(1):1–13, 2018. DOI: 10.1186/s12859-018-2162-x

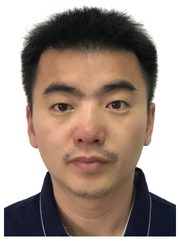
[53] T. Weinkauf and D. Günther. Separatrix persistence: Extraction of salient edges on surfaces using topological methods. *Computer Graphics Forum*, 28(5):1519–1528, 2009. DOI: 10.1111/j.1467-8659.2009.01528.x

[54] 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*, pp. 515–524. ACM, 2002. DOI: 10.1145/584792.584877



**Tong Ge** is a research scientist at Beke. He received his Ph.D. degree from Shandong University, where he works with Prof. Yunhai Wang and Prof. Baoquan Chen in the IDEAS Lab. His research interests are data visualization, application system, and machine learning. Specifically, he has been working on programming languages (e.g. Canis) and authoring tools (e.g. CAST) for data-driven chart animation.

**Xu Luo** is currently a second-year graduate student at the School of Computer Science and Technology, Shandong University. His research interests include information visualization and human-computer interaction.



**Yunhai Wang** is a professor in the School of Computer Science and Technology at Shandong University. He serves as the associate editor of the Computer Graphics Forum and IEEE Computer Graphics and Applications. His interests include scientific visualization, information visualization, and computer graphics.

**Michael Sedlmair** is a professor at the University of Stuttgart and leads the research group for Visualization and Virtual/Augmented Reality there. He received his Ph.D. degree in Computer Science from the University of Munich, Germany, in 2010. Further stops included the Jacobs University Bremen, the University of Vienna, the University of British Columbia in Vancouver, and the BMW Group Research and Technology, Munich. His research interests focus on visual and interactive machine learning, perceptual modeling for visualization, immersive analytics and situated visualization, novel interaction technologies, as well as the methodological and theoretical foundations underlying them.

**Zhanglin Cheng** received the Ph.D. degree from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2008. He is a Professor with the Shenzhen Key Laboratory of Visual Computing and Analytics (VisuCA), Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen. His research interests include computer graphics and visualization.

**Ying Zhao** is currently a professor at the School of Computer Science and Engineering, Central South University. He received his bachelor's degree from Central South University in 2002 and his postgraduate degree from Tsinghua University in 2005. He obtained his Ph.D. degree in Computer Application Technology from Central South University in 2014. His main research interests include information visualization and visual analytics.

**Xin Liu** studied at Peking University (PKU) from 2005 to 2009 and obtained his bachelor's degree in biological sciences there in 2009. In July 2009, he started his Ph.D. study in the Chinese University of Hong Kong (CUHK). He obtained his Ph.D. degree in medical sciences at CUHK in September 2013. In the meantime, he worked at BGI (formerly known as Beijing Genomics Institute). And currently, he is working at BGI as a research fellow. His research has been directed toward understanding the genomic features, and evolution of plant species, and developing/applying genomic tools to help improve crops.

**Oliver Deussen** graduated at Karlsruhe Institute of Technology and is now a professor of visual computing at the University of Konstanz (Germany) and visiting professor at the Shenzhen Institute of Applied Technology (Chinese Academy of Science). He is one of the speakers of the Excellence Cluster "Centre for the Advanced Study of Collective Behavior" and vice speaker of the SFB Transregio "Quantitative Methods for Visual Computing", a large research project conducted together with the University of Stuttgart. He is President of the Eurographics Association and served as Co-Editor in Chief of the Computer Graphics Forum from 2012 to 2015. His areas of interest are modeling and rendering of complex biological systems, non-photorealistic rendering as well as Information Visualization. He also contributed papers on geometry processing, sampling methods, and image-based modeling.

**Baoquan Chen** is a Professor at Peking University, where he is the Executive Director of the Center on Frontiers of Computing Studies. Prior to the current post, he was the Dean of the School of Computer Science and Technology at Shandong University, and the founding director of the Visual Computing Research Center, Shenzhen Institute of Advanced Technology (SIAT), Chinese Academy of Sciences (2008-2013), and a faculty member at Computer Science and Engineering at the University of Minnesota at Twin Cities (2000-2008). His research interests generally lie in computer graphics, visualization, and human-computer interaction, focusing specifically on large-scale city modeling, simulation, and visualization.