Hernandez Leon, Patricia; Caro, Miguel A.

Cluster-based multidimensional scaling embedding tool for data visualization

Published in:
Physica Scripta

DOI:
10.1088/1402-4896/ad432e

Published: 09/05/2024

Document Version
Publisher's PDF, also known as Version of record

Cluster-based multidimensional scaling embedding tool for data visualization

Physica Scripta

PAPER

Cluster-based multidimensional scaling embedding tool for data visualization

Patricia Hernández-León* and **Miguel A Caro**

Department of Chemistry and Materials Science, Aalto University, 02150 Espoo, Finland

* Author to whom any correspondence should be addressed.

E-mail: patricia.hernandezleon@aalto.fi and mcaroba@gmail.com

Keywords: dimensionality reduction, data visualization, cluster MDS, machine learning, embedding algorithms

Supplementary material for this article is available online

OPEN ACCESS

RECEIVED
30 November 2023

REVISED
2 April 2024

ACCEPTED FOR PUBLICATION
9 April 2024

PUBLISHED
9 May 2024

Abstract

We present a new technique for visualizing high-dimensional data called cluster MDS (cl-MDS), which addresses a common difficulty of dimensionality reduction methods: preserving both local and global structures of the original sample in a single 2-dimensional visualization. Its algorithm combines the well-known multidimensional scaling (MDS) tool with the *k*-medoids data clustering technique, and enables hierarchical embedding, sparsification and estimation of 2-dimensional coordinates for additional points. While cl-MDS is a generally applicable tool, we also include specific recipes for atomic structure applications. We apply this method to non-linear data of increasing complexity where different layers of locality are relevant, showing a clear improvement in their retrieval and visualization quality.

1. Introduction

Data complexity is a reflection of the world's complexity. This manifests itself in the presence of high-dimensional datasets in all fields of science and the humanities. Even though there are several types of data complexity, dimensionality alone can *per se* drastically reduce the insight (even the scientific knowledge) that we can extract from a given sample. In this context, data visualization can be very valuable, albeit extremely difficult to achieve with a high number of dimensions n (where high means $n > 3$). An obvious approach to tackle this problem is reducing the number of dimensions involved, so as to unravel the original information within our limited 'visual/dimensional grasp'. In practice, this is essentially equivalent to finding a (satisfactory) map between a low-dimensional representation of the data (preferably within a 2–3 dimensional Euclidean space) and the original high-dimensional representation.

The so-called *dimensionality reduction techniques* are an example of how to achieve one such map, increasingly used thanks to the popularization of machine learning (ML) and data mining in most fields of science. These methods generate an (either linear or non-linear) embedding that leads to an optimized candidate low-dimensional representation of the original sample. Note that we refer to *candidate* representations; despite its existence, the map is not necessarily unique nor exact [1]. There is an unavoidable tradeoff between the amount of information preserved and the dimensionality reduction required, leading to a plethora of possible approaches and an extensive literature on the matter. Some of the best established among these techniques are principal component analysis (PCA) [2], t-distributed stochastic neighbor embedding (t-SNE) [3, 4], Isomap [5] and multidimensional scaling (MDS) [6–8]. Such variety of methods is, however, a symptom of a deeper problem: while they aim for the same mathematical object, their outputs differ widely. More importantly, the differing representations/visualizations can lead to disagreement in the interpretation and retained knowledge that can be obtained from a dataset.

Ideally, a suitable goodness-of-fit test would allow the user to choose the best algorithm for a given sample. Unfortunately, there is no universal metric that measures the accuracy of the resulting embedding nor its quality as a visualization of the original sample [1, 9, 10], not to mention both simultaneously. Having no absolute

measure of the quality of a method, a sensible approach is to reexamine the aforementioned techniques in order to overcome some of their particular shortcomings, usually with the objective to either obtain an overall mathematical improvement or a concrete domain-specific one. Examples of the former, such as the uniform manifold approximation and projection (UMAP) method [11], focus on increasing the robustness of previous theoretical foundations, with special emphasis on their mathematically based algorithmic decisions. On the other hand, improvements for domain-specific applications are motivated by the poor performance and visualizations obtained for high-dimensional non-linear data, e.g., biological data in the case of the potential of heat diffusion for affinity-based transition embedding (PHATE) tool [12].

This paper introduces a new member of the last category that we call cluster MDS (cl-MDS) [13], motivated by previous work with atomic structures. This method arises from the need for an embedding tool which hierarchically preserves global and local features in the same visualization, given their significance when analyzing atomic databases. Due to the inherent local or global nature of the algorithms, just retaining local structures without detriment to global ones poses a serious challenge to the majority of dimensionality reduction techniques, especially to those favoring local distances over other scales (e.g., t-SNE and UMAP). In this context, methods that focus on preserving the global distance structure (e.g., MDS and Isomap) tend to perform better, but are heavily reliant on the distribution of local features of the dataset. cl-MDS is our attempt at combining the strengths of (metric) MDS with a data clustering technique. While similar ideas have already been explored in different local MDS methods [14–16], we use a carefully devised algorithm that always includes at least one global MDS embedding. This is a crucial difference, that allows us to retrieve and to embed several layers of locality consistently.

Despite our background and motivation, we have developed this algorithm bearing a general approach in mind; while the nuances of each sample may differ across domains, some of them are likely to possess global and local structures that could benefit from this new tool. As a result, this paper is organized as follows. Section 2 presents a detailed description of the cl-MDS algorithm and its general features. Section 3 includes a diverse selection of examples as well as some comparisons with other methods. The advantages and disadvantages of cl-MDS are discussed there too. We expand on our motivation in section 3.2, where we highlight the value of cl-MDS for visualizing kernel similarities of atomic environments. We summarize and conclude in section 4. Domain-specific recipes for atomic structure visualization are given in appendix.

2. Cluster MDS algorithm

The main purpose of cl-MDS is to obtain a low-dimensional representation of some high-dimensional data, where the distances between data points resemble the original ones as much as possible, similarly to metric MDS [8]. However, the key additional constraint relates to preserving the maximum amount of local information while improving the visualization of global structures with a sole embedding. As discussed earlier, most of the current dimensionality reduction methods typically fail to capture that interplay between the local and global details, since they usually focus on either the former or the latter.

With this in mind, let us consider a sample of N data points $X = \{x_1,\ldots,x_N\}$ contained in a high-dimensional metric space $\mathbb{R}^n$, where $n > 2$ and $\mathcal{I} = \{1,\ldots,N\}$ denotes its set of indices. Given an associated distance matrix $\mathbf{D}$ with elements D_{ij} for each $i, j \in \mathcal{I}$, the intended output of the algorithm is a low-dimensional representation $Y = \{y_1,\ldots,y_N\}$ of X within $\mathbb{R}^2$. Unlike other methods, the target number of dimensions m is fixed ($m = 2$) because cl-MDS has been developed as a visualization tool, rather than a general dimensionality reduction technique. Also, note that the notion of distance refers here to any metric function defined on X, which will be used as a dissimilarity measure.

The algorithm consists of three parts. The first and second are responsible for identifying the local and global structure of X, respectively, and computing their corresponding (independent) 2-dimensional embeddings. Since those mappings give rise to different representations in $\mathbb{R}^2$, the third part of the algorithm seeks to reconcile the information (local and global) into a single representation, leading to $Y \subset \mathbb{R}^2$. More specifically, the following steps are performed (see figure 1):

A. Identify local structures

1. Clustering using k-medoids, with $k = N_{\mathrm{cl}}$, separates the dataset into N_{cl} data clusters.
2. MDS embedding of each cluster, separately (N_{cl} independent local maps).

B. Identify the global structure

1. Selection of reference (*anchor*) points in each cluster.

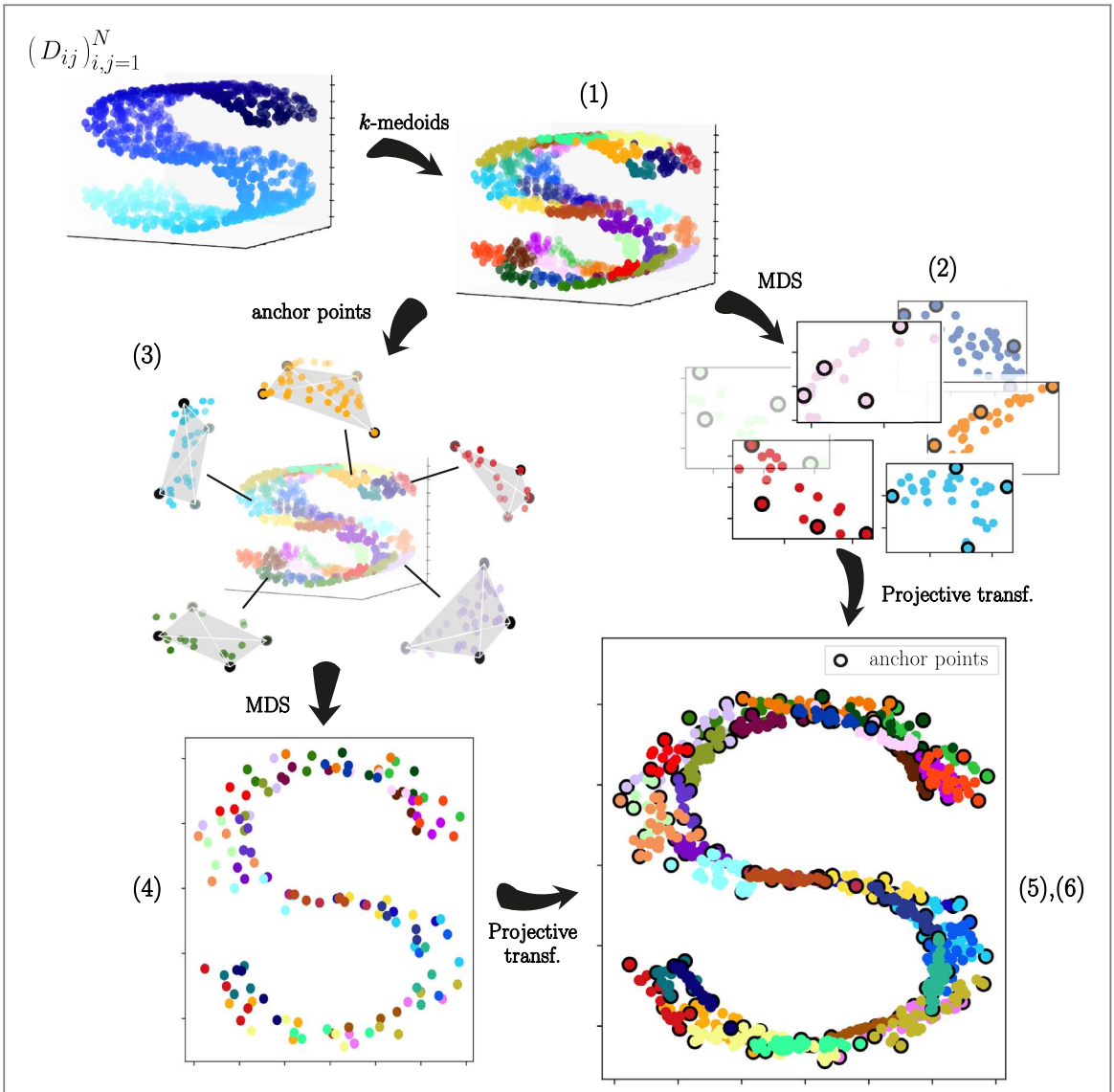

Figure 1. Steps of the cl-MDS algorithm: (1) *k*-medoids clustering of the data; (2) MDS-based local embedding of the individual clusters; (3) anchor-point selection within the individual clusters; (4) MDS-based global embedding of the anchor points only; (5, 6) global embedding of all data points based on transformations derived from (2, 4).

2. Joint MDS embedding of reference points only (global map).

C. From local to global embedding

1. Checking for pathological configurations.

2. Carry local representation over to the global map, preserving the local structure of the data.

These steps constitute the core of the cl-MDS algorithm and are further explained in the following subsections.

Since the division between local and global data structure is necessarily dataset-specific, we have extended the base cl-MDS algorithm outlined above to accommodate an arbitrarily complex nested hierarchy of the data structure. That is, the cl-MDS algorithm can perform an arbitrary number of levels of embedding, in practice usually limited to a few, by hierarchically grouping small clusters together into bigger ones. This feature can be useful for particularly complex datasets and is introduced in section 2.4. Additional features of the algorithm are presented in appendix, focused on a case application to representing atomic structures.

2.1. Local structures

The first part of cl-MDS seeks to identify all the local information present in the sample *X*. While there are global dimensionality reduction techniques, such as MDS, that in principle preserve most local structures, their performance significantly deteriorates as the size and complexity of *X* increases (see section 3 for examples). We

can ameliorate this problem by providing a sensible division of the sample into subsets of related data points and then computing a 2-dimensional embedding individually for each subset. Though it might seem that we are replacing one problem with another, this solution is quite convenient for our purposes as will become clear in the next subsections. Additionally, since the algorithm and its output are based on dissimilarity data, we have a natural criterion for a classification into subsets. Using that information, we can divide X in N_{cl} subsets known as *clusters*. Their main characteristic is the low dissimilarity values between members of a same cluster compared with those for points in different ones. As a result, each subset has a smaller distribution of dissimilarities than the complete sample, which will also help in our search for a finer embedding.

Therefore, the cl-MDS algorithm starts with the computation and optimization of a clustering of X (step 1), with the distance matrix $\mathbf{D}$ and the total number of clusters N_{cl} as input parameters. A suitable clustering technique is the k-medoids method [17, 18], with $k = N_{\mathrm{cl}}$ in our case. Unlike similar (and arguably more popular) barycenter-based algorithms such as k-means [19, 20] or spectral clustering [21], k-medoids builds each cluster considering a *medoid*. This is the element in a cluster with minimal average dissimilarity to the remaining points in that same cluster. Thus, the k-medoids clustering process relies on selecting actual points from the sample as centroids, rather than these centroids being calculated as the coordinates in $\mathbb{R}^n$ with, respectively, the smallest average intracluster distances. The distinction is important because, even though all points in X belong to $\mathbb{R}^n$, not all points in $\mathbb{R}^n$ can necessarily be mapped back to a meaningful or interpretable data point. We provide a concrete example of this lack of bijection in appendix for the smooth overlap of atomic positions (SOAP) high-dimensional representation of atomic structures [22], which motivates our choice of clustering technique.

For computational efficiency, we have reimplemented Baukhage's k-medoids Python recipe [18] in Fortran. Our implementation, that we call `fast-kmedoids` [23], can be easily built into a Python package with `F2PY` [24]. `fast-kmedoids` incorporates several other new features beyond increased speed, including optimization of initial medoid selection. The latter constitutes one of the main problems with centroid-based techniques, whose result heavily relies on the initialization. Since random selection is too volatile on its own, we combine it with *farthest point sampling* to select the `n_iso` most isolated points from the sample. By default, we set `n_iso` = 1. The complete clustering process in cl-MDS is performed as follows:

1.1 Use the k-medoids algorithm to obtain a set of clusters $\mathcal{C} = \{\mathcal{C}_1,\ldots,\mathcal{C}_{N_{\mathrm{cl}}}\}$ and their corresponding medoids $\mathcal{M} = \{m_1,\ldots,m_{N_{\mathrm{cl}}}\}$, where $m_k \in \mathcal{C}_k$, $\mathcal{C}_k \subset \mathcal{I}$ and $x_{m_k} \in X$ for any $k = 1,\ldots,N_{\mathrm{cl}}$.

1.2 Compute the relative intra-cluster *incoherence* [25],

$$\mathrm{I}_{\mathrm{rel}} = \sum_{k=1}^{N_{\mathrm{cl}}} \frac{1}{N_{\mathcal{C}_k}} \left(\sum_{i \in \mathcal{C}_k} D_{i,m_k} \right), \tag{1}$$

where $N_{\mathcal{C}_k}$ denotes the cardinality of $\mathcal{C}_k$.

1.3 Repeat until a fixed maximum number of tries is reached, for a different medoid initialization. Keep the set with minimal $\mathrm{I}_{\mathrm{rel}}$, ensuring the lowest internal incoherence of the clusters among all candidate k-medoids solutions.

In our cl-MDS Python implementation [13], the number of repetitions is set by the parameter `iter_med`, while the initialization of k-medoids is controlled by `init_medoids="random"|"isolated"` (`default`). The default option forces the inclusion of (at least) the most isolated point in the initialization. We recommend to test different proportions of isolated initial medoids (`n_iso_med`), especially for highly uneven data distributions where certain data types are infrequent but very relevant (e.g., when interesting areas of the configuration space are poorly sampled). Also, note that extensive iteration is needed for a robust initialization, and it is not guaranteed to find the global minimum configuration of medoids. The user can provide their custom initial medoids, e.g., when opting for a more thorough search using alternative optimization approaches.

Next, we proceed with the embedding, step 2. A low-dimensional representation $Y_{\mathcal{C}_k}^{(l)}$ for each cluster $\mathcal{C}_k$ is computed applying the standard MDS method, with $k = 1,\ldots,N_{\mathrm{cl}}$. In particular, we use a weighted variation [13] of the metric MDS implementation included in the Python module `scikit-learn` [26]. In this version of the method, the coordinates in the low-dimensional space are optimized such that the pairwise distances between the embedded data points reproduce the input dissimilarity data as closely as possible. This is achieved by minimizing the stress, an objective function defined as

$$\sigma = \sum_i \sum_{i>j} w_{ij}(D_{ij} - d_{ij})^2, \tag{2}$$

where d_{ij} and w_{ij} correspond to the computed distances in the low-dimensional space and their weights respectively, with $i, j \in \mathcal{I}$. Therefore, each data point gets an optimized representation using a non-linear map. Note that we need to partition $\mathbf{D}$ considering the previous set of clusters before computing the MDS embeddings; that is, each representation $Y^{(l)}_{\mathcal{C}_k}$ is obtained from a stress σ_k restricted to $i, j \in \mathcal{C}_k$. An additional remark: the current cl-MDS implementation does not address missing values in the distance matrix, since it only accepts weights per cluster (w_k for all $i, j \in \mathcal{C}_k$). That said, the code could be carefully adapted in the future to accommodate missing dissimilarities [27]; meanwhile, a data-specific pre-processing step is needed in such case. Similarly, we could weight the importance of each data point in the stress inversely to some expected noise, handling noisy data.

Once step 2 concludes, a set of embeddings $Y^{(\text{local})} = \{Y^{(l)}_{\mathcal{C}_1}, \ldots, Y^{(l)}_{\mathcal{C}_{N_{\text{cl}}}}\}$ is obtained. Hence, the output is a collection of N_{cl} 2-dimensional representations containing distinct 'slices' of local information, despite reaching the same target space. As we will show next, they do not correspond to the final representation Y of our algorithm either.

2.2. Global structures

The second part of the cl-MDS algorithm gathers the global information and maps it to a 2-dimensional Euclidean space $\mathbb{R}^2$. As for the local embeddings, we utilize the same dimensionality reduction technique, MDS. As we discussed in section 2.1, MDS mapping of the whole sample may lead to unsatisfactory results for large datasets, whereas applying it to a small subset made of selected points could give us enough insight into its global features. That subset must also contain several reference points from each cluster, enclosing as much local information as possible. Since the distribution of those points also determines the global map, they will act as *anchor points* between local and global representations. Therefore, this part of cl-MDS is divided in two steps: finding a suitable collection of reference points that 'frames' all clusters (step 3) and, then, proceeding with their embedding (step 4).

Step 3 selects as anchor points a subset of X aiming at satisfying two conditions simultaneously: preserving a maximal amount of local information and outlining the overall sample features. Intuitively, we need at least one anchor point from each cluster to grasp their global distribution. However, this would disregard the local structure within the clusters. Therefore, in practice, up to four anchor points per cluster ($1 \leqslant n_{\text{anc}} \leqslant 4$) are chosen to both ensure the aforementioned preservation of local features and ease the MDS minimization problem. The benefits of this choice are further discussed on section 2.3. Hence, this part of the algorithm searches the high-dimensional representation of each cluster $\mathcal{C}_k$ for the vertices $\mathcal{A}_k = \{a_{k1}, \ldots, a_{kn_{\text{anc}}}\} \subset \mathcal{C}_k$ which define the tetrahedron of maximal volume, for $k = 1, \ldots, N_{\text{cl}}$. The detailed procedure is as follows:

3.1 Check if the cardinality $N_{\mathcal{C}_k}$ of cluster $\mathcal{C}_k$ satisfies $N_{\mathcal{C}_k} > N^{\max}_{\mathcal{C}_k}$. For the sake of computational efficiency, large clusters benefit from discriminating between points to reduce the list of candidate vertices before considering any tetrahedra. In that case, the p-th percentile of the distance to the medoid m_k is chosen as threshold, leaving only farther points in $\mathcal{C}_k$ as vertices. The percentile rank p is customized for different $N_{\mathcal{C}_k}$ with the parameter `param_anchor`. Our tests indicate that $N^{\max}_{\mathcal{C}_k} = 70$ is a robust choice, and this is hardcoded in the cl-MDS program.

3.2 Compute the volume of each possible tetrahedron whose vertices v are a subset of the (reduced) list of points in $\mathcal{C}_k$. Since a tetrahedron corresponds to a 3-simplex S_3, its volume V can be computed using the CayleyMenger determinant as follows [28, 29]:

$$V(S_3)^2 = \frac{1}{2^3(3!)^2} \begin{vmatrix} 0 & 1_4^\top \\ 1_4 & \mathbf{D}[v] \end{vmatrix}, \tag{3}$$

where $1_4^\top = (1,\ 1,\ 1,\ 1)$ and $\mathbf{D}[v]$ is the 4×4 submatrix of $\mathbf{D}$ corresponding to the vertices of the 3-simplex. We note here that the tetrahedron's volume is computed directly from the distances, and that these distances are not necessarily based on an Euclidean metric.

3.3 Keep the set of vertices $\mathcal{A}_k$ forming the tetrahedron with maximal volume.

Repeating those steps for each cluster, we obtain the complete set of anchor points $\mathcal{A} = \bigcup_{k=1}^{N_{\text{cl}}} \mathcal{A}_k$. We opted for the volume as a measure of the amount of preserved information, although one could argue that there exist better criteria, such as the number of data points enclosed by the tetrahedron. Those measures are nonetheless much more expensive computationally given their dependence on $N_{\mathcal{C}_k}$, whereas the volume criterion is independent of it.

Finally, step 4 is carried out. Applying the same weighted metric variation of the MDS method from step 2, a 2-dimensional representation of the anchor points $Y^{(g)}_{\text{anchor}} = \{y^{(g)}_{a_i} \mid a_i \in \mathcal{A}\}$ is computed. Notably, those new coordinates encode the global 2-dimensional distribution of the local structures previously clustered.

2.3. From local to global embedding

We have so far obtained a *collection* of low-dimensional representations of the original dataset X, characterized by preserving local and global structures *separately*. These embeddings are limited to certain subsets of points, i.e., anchor points in the case of the global one and clusters in the case of the local ones. The last part of the algorithm seeks to reconcile these two independent representations to achieve a complete representation Y of X, by mapping each cluster's local coordinates into the global map. That is, given a cluster $\mathcal{C}_k$ with $k = 1,\ldots,N_{\text{cl}}$, we can leverage its anchor points $\mathcal{A}_k$, whose coordinates are known in local and global representations, to obtain a suitable transformation for all of its members.

Two types of transformations are implemented: affine transformations $\mathbf{A}$, and projective transformations $\mathbf{H}$ (also known as perspective mappings or *homographies*) [30]. The simpler affine transformations preserve parallel lines, as opposed to the non-linear maps performed by the MDS algorithm. Hence, homographies were introduced for the sake of consistency, since they always preserve incidence (relations of containment between points, lines and planes) but not parallelism. There are other properties that are not preserved by the MDS algorithm, suggesting that there is room for future improvement in this direction. For instance, the relative ordering of distances between points is not always preserved, despite the MDS efforts of reproducing relative distances [see equation (2)]. This property will influence the selection process in step 5, as we explain below.

Before computing a transformation operator (in the form of a matrix), we need to ensure the absence of pathological configurations. Step 5 determines the most suitable transformation (affine or projective) for each cluster, depending on the number of anchor points n_{anc} and their relative distribution in both representations. The importance of this last property lies in its relation to convexity, which is a necessary (and sufficient) condition for homography. That is, the anchor points $\mathcal{A}_k$ ($k = 1,\ldots,N_{\text{cl}}$) form a quadrilateral in each representation whose convexity is not ensured in both maps (as anticipated above), potentially leading to an ill-conditioned transformation. Thus, the pathology check for a cluster $\mathcal{C}_k$ is performed as follows:

5.1 Check if $n_{\text{anc}} < 4$. In that case, an affine transformation $\mathbf{A}_k$ is chosen trivially and no further steps are needed.

5.2 Check if $\mathcal{A}_k$ does not coincide with its convex hull (i.e., the smallest convex subset in $\mathcal{A}_k$ containing $\mathcal{A}_k$) in the local representation $Y^{(l)}_{\mathcal{A}_k}$. This corresponds to one of the anchor points being enclosed within the triangle defined by the other three, or to (at least) three anchor points being collinear. In that case, a homography is ill-conditioned and we redefine $\mathcal{A}_k$ as its convex hull, implying $n_{\text{anc}} < 4$ and obtaining the result of step 5.1. The convex hull is computed using the Qhull library [31] through `scipy.spatial.ConvexHull` [32].

5.3 Repeat the previous check, now using the global representation $Y^{(g)}_{\mathcal{A}_k}$. If true, the four anchor points will be used to compute an affine transformation $\mathbf{A}_k$. Otherwise, the convexity is confirmed in both representations and a homography $\mathbf{H_k}$ is chosen.

Once each cluster has been checked, the algorithm can proceed with their mapping using the most appropriate transformation $\mathbf{T_k}$ for each $k = 1,\ldots,N_{\text{cl}}$, where $\mathbf{T_k}$ corresponds to $\mathbf{A}_k$ or $\mathbf{H_k}$. For simplicity, we use homogeneous coordinates to express the details of step 6. In this notation, any point $p^\top = (x, y)$ from $\mathbb{R}^2$ has an homogeneous representation $\tilde{p}^\top = (x, y, 1)$. On the other hand, any homogeneous point $\tilde{q}^\top = (u, v, w)$ is identified with $\tilde{r}^\top = \tilde{q}^\top/w = (u/w, v/w, 1)$, implying the existence of a bijection between Cartesian coordinates in the Euclidean plane and homogeneous coordinates in the *projective plane* ($\mathbb{P}^2$). Alternatively, equivalence classes can be used to understand the concept of homogeneous coordinates within a more rigorous theoretical framework [33]. This notation allows us to represent the transformations as simple matrix multiplications, whose details are explained below. Then, step 6 is organized as follows:

6.1 Compute the transformation matrix $\mathbf{T_k}$ that solves the equation

$$\begin{pmatrix} y^{(g)}_{a_i} \\ 1 \end{pmatrix} = \mathbf{T_k} \begin{pmatrix} y^{(l)}_{a_i} \\ 1 \end{pmatrix}, \tag{4}$$

for each $a_i \in \mathcal{A}_k$.

In particular, affine transformations (usually represented as the composition of a linear transformation $\mathbf{L}_k$ and a translation $\mathbf{b}_k$) correspond to the following matrix,

$$\mathbf{A}_k = \begin{pmatrix} \mathbf{L_k} & \mathbf{b_k} \\ 0_n^\top & 1 \end{pmatrix}, \tag{5}$$

where $0_2^\top = (0,\ 0)$ for transformations in $\mathbb{R}^2$, and, therefore, they are obtained as the least-squares solution of equation (4).

On the other hand, a homography matrix [34] has the form

$$\mathbf{H_k} = \left(\mathbf{A}_k^{(g)}\right)^{-1} \mathbf{F}\ \mathbf{A}_\mathbf{k}^{(\mathrm{l})}, \tag{6}$$

where $\mathbf{A}_\mathbf{k}^{(\mathrm{l})}$ is the affine transformation from the local $\mathcal{A}_k$ coordinates to the *canonical quadrilateral* $\{(1,0), (0,0), (0,1), (a,b)\}$, with $(a,\ b) = \mathbf{A}_\mathbf{k}^{(\mathrm{l})} y_{a_3}^{(\mathrm{l})}$. Likewise, $\mathbf{A}_k^{(g)}$ is the equivalent transformation from the global coordinates to a canonical quadrilateral whose fourth point is $(c,\ d) = \mathbf{A}_k^{(g)} y_{a_3}^{(\mathrm{g})}$. Both affine transformations can be computed using the least-squares method too. Finally, **F** is a *linear fractional transformation* from the first canonical quadrilateral into the second,

$$\mathbf{F} = \begin{pmatrix} bcs & 0 & 0 \\ 0 & ads & 0 \\ b(cs - at) & a(ds - bt) & abt \end{pmatrix}, \tag{7}$$

where $s = a + b - 1$ and $t = c + d - 1$ are positive for convex quadrilaterals. Since both pairs (a,b) and (c,d) are known from the previous affine transformations, **F** is known too.

6.2 Compute the global representation of cluster $\mathcal{C}_k$ using the previous mapping,

$$\begin{pmatrix} y_i' \\ w \end{pmatrix} = \mathbf{T_k} \begin{pmatrix} y_i^{(\mathrm{l})} \\ 1 \end{pmatrix}, \tag{8}$$

and applying the *perspective divide* $y_i^{(g)} = y_i'/w$ to each $i \in \mathcal{C}_k$.

6.3 If $\mathbf{T_k} = \mathbf{H_k}$, an affine transformation is also computed for comparison. We retain the result with the lowest residue,

$$R(\mathbf{T_k}) = \sum_i (y_i^{(l)} - y_i^{(g)}(\mathbf{T_k}))^2, \tag{9}$$

in order to minimize the effect of outliers.

As a result of step 6, we obtain the output of the cl-MDS algorithm, $Y = \bigcup_{k=1}^{N_{\mathrm{cl}}} Y_{\mathcal{C}_k}^{(\mathrm{g})} = \{y_1,\ldots,y_N\}$. These steps constitute the core of the cl-MDS algorithm.

2.4. Cluster hierarchy and sparsification

Additional features are implemented to complement and improve the cl-MDS algorithm. Here we introduce the general ones, whereas appendix details those specific to visualization of databases of atomic structures. While cl-MDS strives to preserve local and global structures, its base algorithm sometimes lacks the sufficient flexibility for mapping complex databases where different layers of locality can be meaningful. To address this issue we introduce *cluster hierarchy*. In a hierarchical cluster setup, the number of clusters hyperparameter N_{cl} is replaced by a hierarchy hyperparameter,

$$h = [N_{\text{level 0}},\ N_{\text{level 1}},\ N_{\text{level 2}},\ldots,1], \tag{10}$$

where $N_{\text{level 0}}$ refers to the finest clustering level and 1 represents the final global MDS embedding (i.e., $N_{\text{level 0}} > \ldots > N_{\text{level m}} > \ldots > 1$). This list enables a hierarchical embedding on steps 3 to 6 that mimics the idea behind hierarchical clustering [35]. Each level m corresponds to a grouping $C^{(m)}$ of the dataset with a specific embedding $Y^{(m)}$ in the 2-dimensional Euclidean space, such that the next level $(m+1)$ merges several of those clusters and computes a new 2-dimensional embedding $Y^{(m+1)}$, using $Y^{(m)}$ as the local representation. In this way, hierarchical embedding improves the representation of samples with several levels of locality. The simplest hierarchy $[N_{\mathrm{cl}}, 1]$ is equivalent to N_{cl}, where a sole layer of local information is considered. The hierarchy approach is schematically depicted in figure 2 for a [5,2,1] hierarchy. See section 3 for examples.

To enable processing of large databases we have added sparsification support to cl-MDS. In our Python implementation this is selected through the keywords `sparsify` and `n_sparse`. Sparsification carries out the cl-MDS operations only on a subset of the data points, selected according to some predefined recipe (`sparsify="random"|"cur"|list`). `n_sparse` determines the (maximum) size of the resulting sparse set, denoted $\mathcal{I}_{sp}$. Three sparsification options are implemented. The first selection method, `"random"`, uses `numpy`'s random sampling routines [36]. The second option, `"cur"`, is based on CUR matrix decomposition [37, 38], a low-rank approximation procedure. This decomposition is characterized by retaining

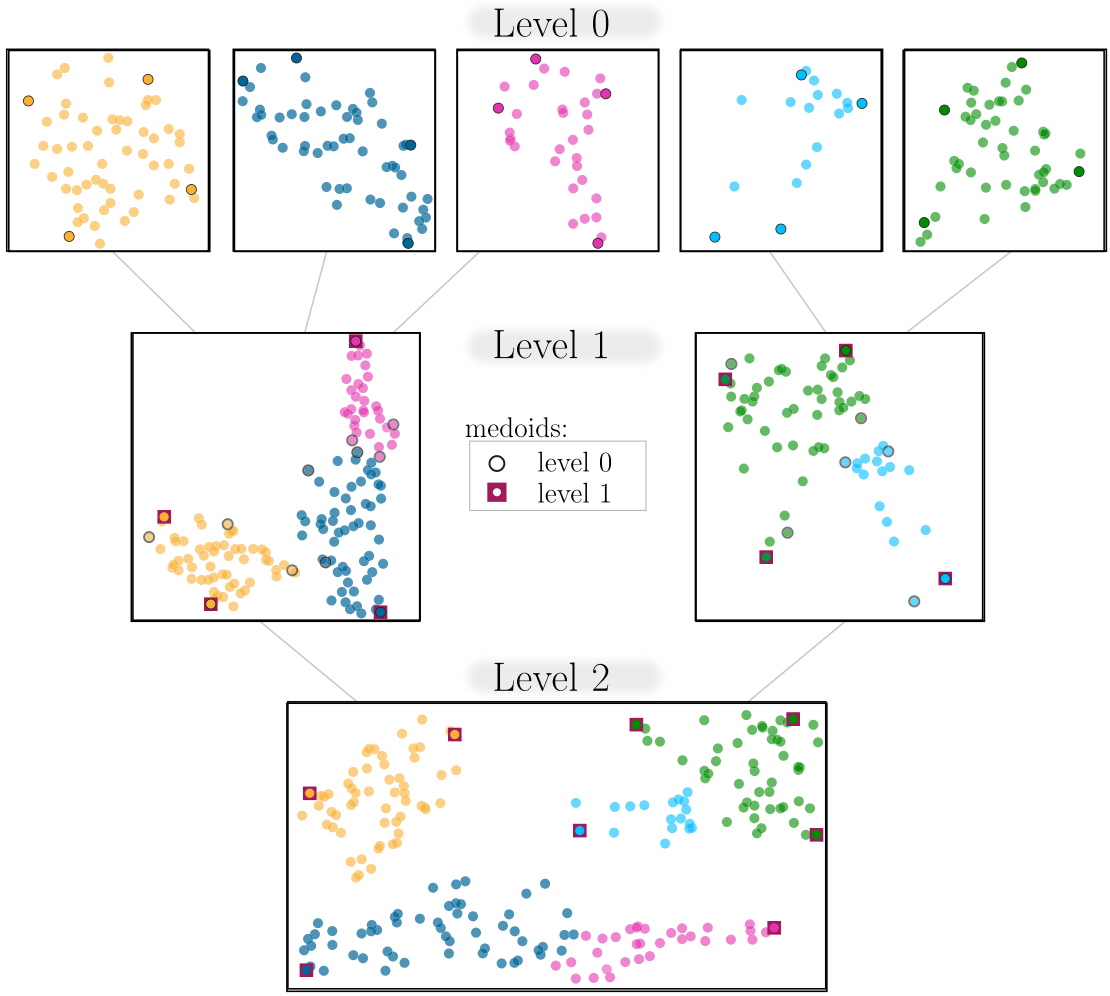

Figure 2. Illustration of the hierarchical cluster setup, using a [5, 2, 1] hierarchy. The figure includes the set of anchor points obtained per clustering level, following step 3. The last level does not require such set since it corresponds to the final embedding.

those rows (and columns) from the original matrix that allow its best low-rank fit, i.e., that capture the most representative part of it. The 'significance' of a row can be expressed in terms of its statistical influence [39] or its Frobenius norm [40], among other properties. The CUR implementation included in cl-MDS uses the latter. Hence, given a distance matrix **D** and a rank `n_sparse`, we choose the indices of those rows with higher Frobenius norm within the dataset. As a third sparsification option, the user can provide directly a custom list or array with the sparse indices to `sparsify`.

Bearing sparsification in mind, we developed a supplementary algorithm (step 7) that estimates the low-dimensional representation of those data points not included in the sparse set, i.e., $\overline{\mathcal{I}}_{sp} \coloneqq \mathcal{I} - \mathcal{I}_{sp}$. The embedding transformations inferred for the sparse set are reused for embedding $\overline{\mathcal{I}}_{sp}$, whereas the MDS mapping per cluster is replaced by an affine mapping. This is achieved through a three-step process:

7.1 Assign each data point i in $\overline{\mathcal{I}}_{sp}$ to the same cluster $\mathcal{C}_k$ as its nearest medoid m_k, with $k \in \{1,\ldots,N_{\mathrm{cl}}\}$. This approach extends the existing clustering to the complete database without requiring computing or storing its complete distance matrix. To avoid future confusion, we refer to the extended cluster k as $\mathcal{C}_k^*$, implying that $\mathcal{C}_k = \mathcal{C}_k^* \cap \mathcal{I}_{sp}$.

7.2 Compute an affine mapping from the original high-dimensional space $\mathbb{R}^n$ to $\mathbb{R}^2$, for each cluster $\mathcal{C}_k^*$ with $k = 1,\ldots,N_{\mathrm{cl}}$. Its matrix representation $\widetilde{\mathrm{A}}_k$ is the least-squares solution to the equation

$$\begin{pmatrix} y_j^{(l)} \\ 1 \end{pmatrix} = \widetilde{\mathrm{A}}_k \begin{pmatrix} x_j \\ 1 \end{pmatrix}, \tag{11}$$

for all sparse points $j \in \mathcal{C}_k$. The tilde is included to emphasize that the embedding is between distinct spaces, as opposed to the affine transformations from step 6.

7.3 Obtain the 2-dimensional representation of each cluster $\mathcal{C}_k^*$ via the composition $\widetilde{\mathbf{T}_k} \coloneqq \mathbf{T}_k \widetilde{\mathbf{A}}_k$, with $k = 1,\ldots,N_{\mathrm{cl}}$. Similarly to step 6.2, these coordinates are the result of the perspective divide $y_i^{(g)} = y'_i/w$, defined by the equation

$$\begin{pmatrix} y'_i \\ w \end{pmatrix} = \widetilde{\mathbf{T}}_k \begin{pmatrix} x_i \\ 1 \end{pmatrix}, \tag{12}$$

where $i \in \mathcal{C}_k^* \cap \overline{\mathcal{I}}_{sp}$. Note that we exclude the data points from $\mathcal{C}_k$ from these calculations because their 2-dimensional representation is already known.

Therefore, this algorithm completes the 2-dimensional representation of the database, $Y = \{y_1,\ldots,y_N\}$, previously restricted to the sparse set, $Y_{sp} \coloneqq \{y_j \mid j \in \mathcal{I}_{sp}\}$. Note that the estimation will be as good (or as bad) as the chosen sparse set.

3. Examples

In this section, we analyze several examples to show the potential of the cl-MDS method as well as its weaknesses. First, a variety of toy examples are introduced in section 3.1 to illustrate its main features and to compare its performance with other dimensionality reduction techniques. Instances of higher complexity are shown in section 3.2, regarding several extensive atomic databases. Additional details of our motivation and these datasets are given in their respective subsections. We decided to focus on atomic-structure samples because (i) they were our motivation to develop cl-MDS in the first place, and (ii) we wanted to apply all the functionalities available, including those related to atomic structure representations. However, the cl-MDS method is applicable to datasets from other fields too.

The following algorithms are considered for qualitative comparisons with cl-MDS: locally linear embedding (LLE) [41], modified LLE [42], Hessian eigenmaps (Hessian LLE) [43], local tangent space alignment (LTSA) [44], Laplacian eigenmaps (LE) [45], and the already mentioned Isomap [5], PCA [2], kPCA [46], MDS [6, 7], t-SNE [3, 4] and UMAP [11]. They are computed using `scikit-learn` [26] implementations except for UMAP, which has its own Python module (`umap`). On the other hand, detailed quantitative comparisons (the so-called goodness-of-fit tests) are not included, since a good choice of metric is highly dependent on the application domain, the user's expectations and the sample itself [9]. These metrics are (inevitably) tailored to diverse definitions of accuracy and visual quality, further adapted to account for specific shortcomings (e.g., the DEMaP metric [12], the 'local continuity' (LC) meta-criteria [47], or measures of *trustworthiness* and *continuity* [48]). Therefore, we opted for strong qualitative comparisons that provide a broader taste of cl-MDS, but we encourage the user to apply these metrics to their own samples after careful selection.

As a consequence of its algorithm, cl-MDS inherits several characteristics of MDS which are relevant for understanding the figures in this section. While their embedding dimensions lack interpretability (i.e., absence of meaningful axes), the relative Euclidean distances of the resulting 2-dimensional representation encode the original dissimilarity measure. Thus, the nearer two data points are in this visualization, the more similar they are and vice versa. In practice, this does not always hold for MDS, a problem that cl-MDS alleviates as we will discuss in section 3.2.1. Additionally, the embedded representation is invariant under affine transformations for both methods. However, the embedding codomain of cl-MDS is usually $[-1, 1] \times [-1, 1]$ without being strictly restricted to it, as opposed to MDS. Since dimensionality reduction techniques differ on their codomain, as well as on their interpretation (only PCA has a strong meaning associated to its axes), we omit the axis information in all the figures to avoid misleading comparisons.

3.1. Toy examples

Before diving into complex examples, let us discuss the main features of cl-MDS using simpler datasets. We start following a classic `scikit-learn` example of manifold learning methods [49], where an S-curve dataset with 1000 points and its corresponding Euclidean distances are used. We chose minimal parameters for all the techniques to compare their fastest computational speed. However, note that cl-MDS was not intended for improving time performance and has not been fully optimized accordingly. Also, this is the only example where the parameters are not fine tuned. Figure 3 illustrates the effect of the cl-MDS main hyperparameter, the number of clusters N_{cl} (see section 2.1). As expected, the algorithm output is sensitive to this choice, requiring a minimum clustering ($N_{\mathrm{cl}} = 15$) to preserve the distances in the embedded space consistently. Just like MDS, cl-

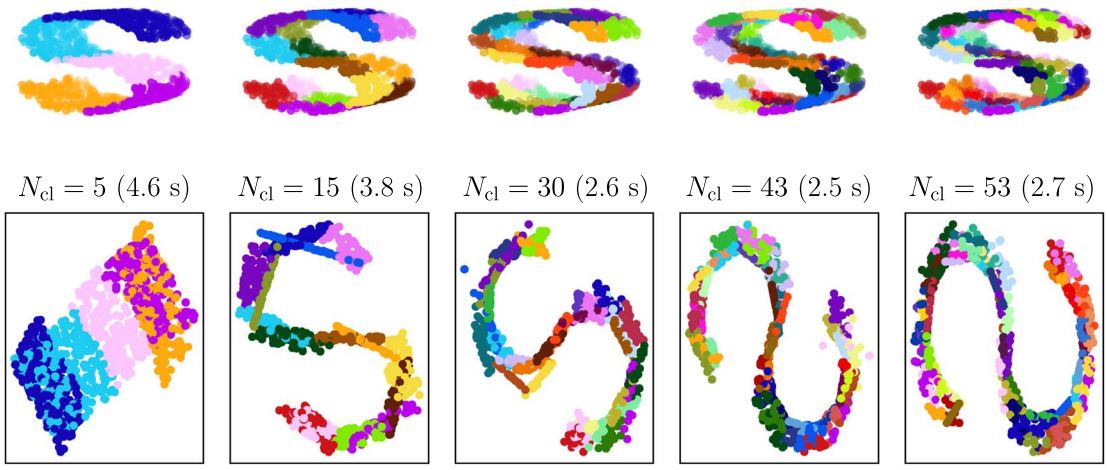

Figure 3. Effect of the number of clusters N_{cl} in cl-MDS embedding and performance for a simple example.

MDS focuses on revealing the metric structure, which does not guarantee uncovering the low-dimensional manifold. This is especially true when using Euclidean distances, since they do not characterize the global distribution of the manifold properly. Hence, this particular example requires feeding a larger amount of local information to the global MDS mapping (step 4) through the clustering, in order to counteract the ambiguity in large distances.

In general, a good choice of N_{cl} depends on the database and the provided metric, although a number between 5 and 20 is enough in our experience to capture local and global details properly. A higher number of clusters can be useful to retrieve finer details, especially in highly complex datasets. Nevertheless, overly increasing N_{cl} worsens the global embedding and dilutes the intrinsic value of medoids as representative data points. Therefore, we recommend finding a compromise between a finer embedding and a smaller clustering. In particular cases, a carefully devised cluster hierarchy can prove quite helpful; the added versatility of hierarchy and medoids information will be discussed in depth in sections 3.2.1 and 3.2.2. Although we use a heuristic approach for fine-tuning most of the parameters in this article, there are well-known criteria for choosing a suitable number of clusters [50, 51]. We have already explored some options, such as the elbow method and the silhouette statistic [52], but further analysis is needed in this direction.

On the other hand, the performance of cl-MDS improves speedwise with the number of clusters, being even faster than MDS for $N_{cl} > 15$ (see MDS performance in figure 4). While counter-intuitive at first, this result illustrates the usual trade-off between performance and sample size already mentioned in section 2.1. Although the cl-MDS algorithm uses several instances of MDS, the subset of data points processed per instance is much smaller than the complete sample. As figure 3 shows, increasing N_{cl} decreases average cluster size and speeds up individual cl-MDS calculations, until it reaches a threshold ($N_{cl} > 43$ in this example). This threshold is determined by the set of anchor points, whose size grows approximately as $4N_{cl}$, recovering the previous tradeoff. Thus, we conclude that our initial/previous compromise for choosing N_{cl} is also reasonable in terms of performance. In general, we expect that the computational complexity of cl-MDS scales similarly to those MDS instances, i.e., $\mathcal{O}(N^3)$ with N switching between the total number of anchor points and the size of the biggest cluster depending on N_{cl}. However, the hierarchical embedding has a non-trivial effect in cl-MDS performance. On the other hand, preliminary empirical analysis for samples up to $N_{sample} = 2000$ points showed an approximate scaling of $\mathcal{O}(N^2_{sample})$. Further testing is left for future work, since we can characterize the computational complexity more appropriately once we have optimized the code thoroughly.

Note that, once a suitable N_{cl} is reached, bigger clusterings may better incorporate the nuanced local details and improve its visual quality, but they do not change the overall embedding significantly. This is a fundamental distinction between cl-MDS and those methods whose main hyperparameter fixes (or guesses) the *number of close neighbors* for each point, i.e., the locality of the embedding. Figure 4 shows examples of the latter in its first two rows. Most of these methods aim to embed the original data uniformly, as opposed to the techniques included in the last row which seek to preserve the metric structure. That is, the former obtain an isotropic representation of the S-manifold but the metric information is partially lost.

Additional cl-MDS parameters, such as the MDS weights and the percentile ranks for anchor points, were minimized here for increased computational speed, reducing the accuracy too. In figure 3, we can appreciate the extreme linearity of certain cluster embeddings, which translates into the cl-MDS mapping being slightly less accurate than the MDS one in figure 4. This effect is a consequence of the MDS optimization process, which does

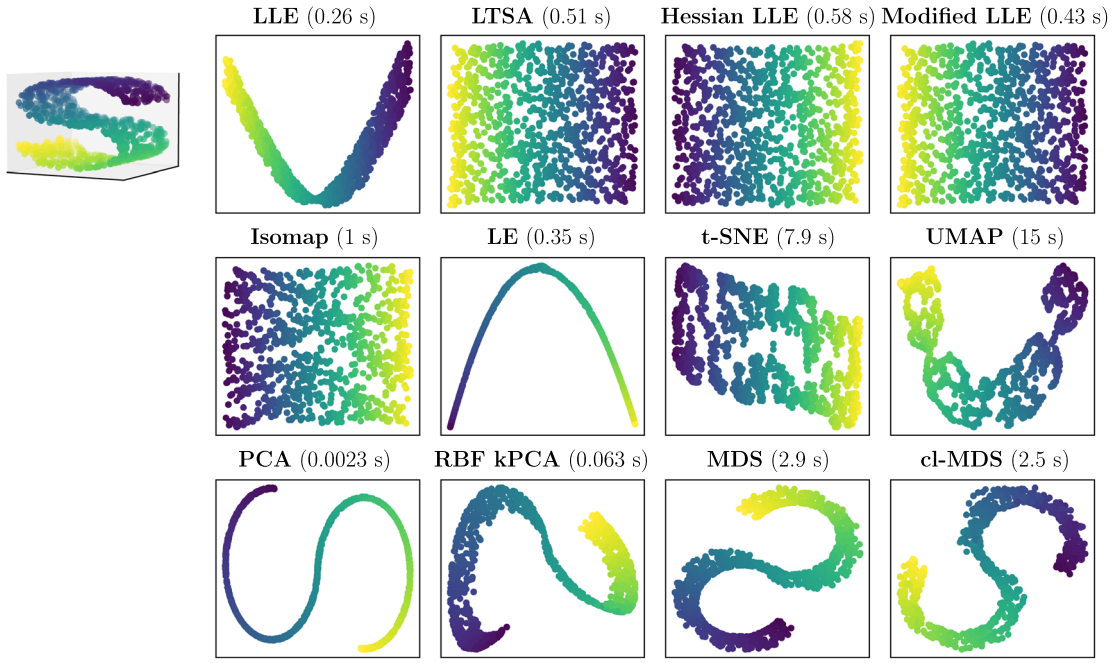

Figure 4. Comparison of several dimensionality reduction techniques applied to an S-curve manifold with 1000 points. Like the original example [49], we used minimal parameters whenever possible. The top two rows include those methods that require a fixed number of neighbors, in this case limited to 15.

not guarantee the preservation of relevant incidence relations (e.g., placing anchor points from the same cluster, originally convex, on a line). Moreover, this is accentuated by a high number of clusters, poor choices of anchor points and insufficient MDS iterations. To alleviate this issue, the cl-MDS implementation includes optimized values of all related parameters by default, reducing the emergence of these artifacts.

The next example corresponds to a dataset of 1000 random points distributed over the unit square, avoiding N_h randomly placed circular regions (*holes*). Rather than characterizing each point i by its 2-dimensional coordinates, we use the vector $v_i := (d_{i\,h_1},\ldots,d_{i\,h_{N_h}}) \in \mathbb{R}^{N_h}$ where $d_{i\,h_j}$ denotes its Euclidean distance to the center of the hole h_j, for $j = 1,\ldots,N_h$. This is a straightforward approach for building a high-dimensional dataset with a custom dimension N_h. Figure 5 illustrates this for an N_h=12 example, with the pairwise distances in $\mathbb{R}^{12}$ fed to cl-MDS. Additionally, this figure includes an application of Voronoi diagrams as a qualitative measure of accuracy for cl-MDS. The Voronoi partition of a set of points corresponds to those regions of space, *Voronoi cells*, containing the closest generator point. That is, the Voronoi partition associated to the medoids in $\mathbb{R}^{12}$ is equivalent, by definition, to clustering these medoids in this Euclidean space. Hence, the metric topology on this partition is preserved in the 2-dimensional embedding only when the Voronoi cells in $\mathbb{R}^2$ contain the clusters perfectly. Figure 5 shows how close cl-MDS is to achieving this objective.

Now that we have built some intuition regarding the advantages, disadvantages and hyperparameters of cl-MDS, we can apply it to more complex datasets. Also, advanced features such as sparsification and the corresponding estimation of the complete 2-dimensional representation are used in the following examples.

3.2. Visualizing atomic environments

Analyzing atomic databases, which can comprise thousands (even millions) of structures, has become an increasingly difficult task. During the last decade, new mathematical descriptions of atomic structures have been developed as an alternative to simpler approaches, such as straightforward visualization of the structures, which are prone to information overload from such databases. As a result, we can compare their members (either atoms or molecules) using abstract mathematical representations known as *atomic descriptors* [22, 53, 54], where each member is characterized numerically. This representation ranges from simple scalar distances and angles to complicated many-body atomic descriptors which are mapped to high-dimensional vectors. These can be used to compute a similarity measure between atomic configurations based on kernel functions, paving the way to study these databases and their underlying properties. Still, the dimensionality of these descriptions can obscure the results and their communication, as we discussed in section 1.

The high-dimensional nature of many-body atomic representations makes them a representative example of the usefulness of dimensionality reduction techniques, in general, and cl-MDS, in particular. These methods can be very effective for visually comparing atomic structures and studying patterns within a database, fostering the

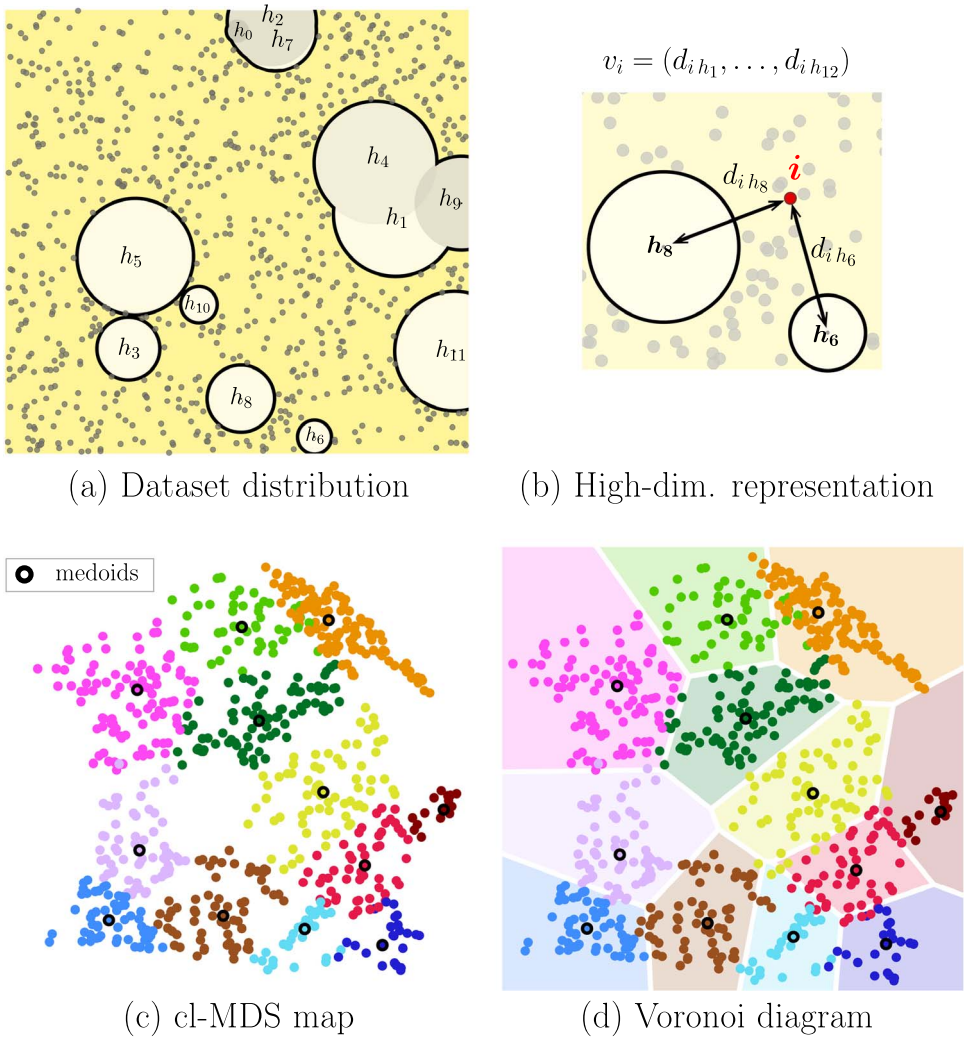

Figure 5. Simple example of cl-MDS applied to a high-dimensional dataset. The sample consists of 1000 points distributed in $\mathbb{R}^2$ as shown on panel (a), where $N_h = 12$. Their high-dimensional representation is obtained from the pairwise distances to each hole, illustrated on panel (b). Panels (c) and (d) show the cl-MDS embedding and the Voronoi diagram of the medoids, respectively.

development of related software [55–57] in recent years. Some of the most common techniques adopted by materials scientists are the linear algorithm PCA, its non-linear counterpart kernel PCA (kPCA) [46], metric MDS, sketch-map [58], and diffusion maps [59, 60]. While these methods are well established and extremely powerful for visualization purposes, atomic-structure samples tend to be highly uneven (i.e., crucial data can be both over- and underrepresented) with multiple structural and chemical features at different scales. Even in the absence of noise, they can be challenging to embed and to display in a clear, intuitive way.

cl-MDS is designed to address those common pitfalls, especially for kernel similarities based on smooth overlap of atomic positions (SOAP) descriptors [22]. cl-MDS aims to enhance the visualization quality across several layers of information, i.e., dominant features at different atomic length scales. On the other hand, atomic descriptors are increasingly used as input for other ML models, either for analysis (e.g., classification tasks) or prediction purposes (e.g., regression models for the potential energy). Adding their results on top of these visualizations is straightforward, with the 2-dimensional embedding working as a 'white canvas'. We argue that the parallelism between visual distances (i.e., Euclidean distances on a 2-dimensional plot) and kernel (dis) similarities provides a transparent and intuitive baseline to navigate these complex samples.

All the examples in this section use the same framework. Following the specifics described in appendix, each data point corresponds to an atom represented by a SOAP descriptor computed with the option `"quippy_soap_turbo"`. This way, the dataset has an associated kernel distance based on the similarities between atomic environments defined within the chosen cutoff radii [see equation (14)]. Note the difference between using a cutoff sphere and restricting the number of neighbors as main criterion for defining the idea of environment. The former establishes how far the neighborhood of an atom stretches without assuming the

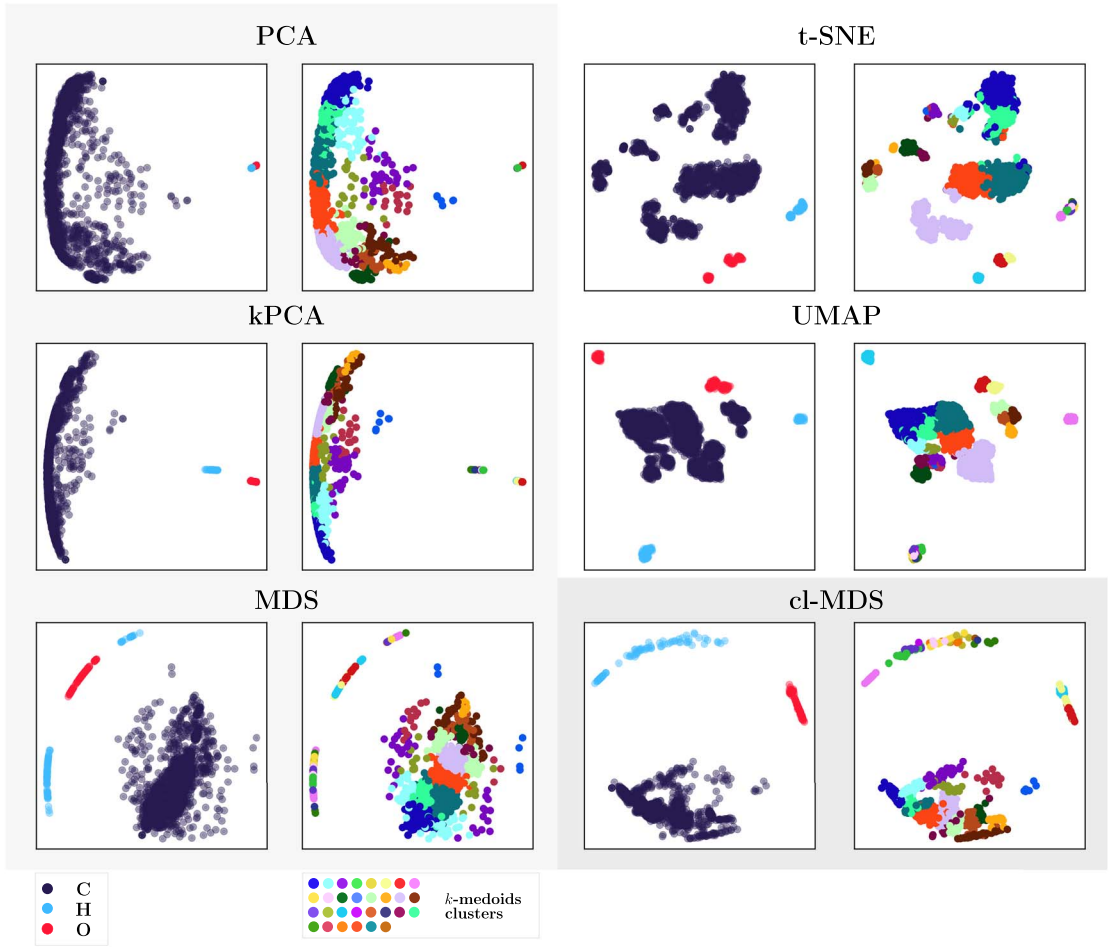

Figure 6. Embedding of 2000 randomly chosen data points performed using several methods, where each point corresponds to an atom from the CHO database. Two plots are included per method/embedding: one color-coded according to chemical species (C, H, O) and another color-coded according to *k*-medoids clustering.

number of nearest neighbors that populate that region. A cutoff sphere-based representation allows us to systematically compare atoms with differing numbers of neighbors, more suitable for this application.

The scripts used to generate the data in the examples are publicly available in the cl-MDS GitHub repository [13], except for the PtAu nanoclusters example. The latter is part of a longer TurboGAP tutorial [61], where more visualizations are included. Other applications of cl-MDS can be already found in the literature [62, 63].

3.2.1. CHO structural database

The CHO database is a subset of the original dataset from [64], corresponding to a wide variety of CHO-containing materials. It contains 675 computer-generated models of CHO materials, for a total of 151,556 unique atomic environments. We consider two approaches to dimensionality reduction: (i) an overall embedding of a multispecies sample and (ii) separated embeddings per chemical species (C, H, O). The former is shown on figure 6, with several dimensionality reduction techniques applied to a random sample of 2000 data points. Here, the high-dimensional representation was computed using cutoff radii $r_{\text{soft}} = 3.75$ Å, $r_{\text{hard}} = 4.25$ Å, with dimension $N_{\text{SOAP}} = 2700$. These radii ensured the convergence of the ML models trained in [64] and, therefore, retain enough relevant structural knowledge. Only those techniques that did not fail were included, illustrating some of the common visualizations available for an atomic sample. The main reasons behind the poor visualizations obtained by Isomap, LE, LTSA, LLE and its variations are: (1) their assumption of a non-disconnected underlying manifold, which is very unlikely when different classes are present in a dataset (e.g., different atomic species); and (2) their tendency to collapse most of the data points together, due to poorly constrained cost functions. A more detailed analysis of the specific weaknesses of those methods can be found on [65].

First, it is noteworthy that the most elementary chemical intuition is captured visually by PCA, kPCA and cl-MDS solely. A multispecies sample is characterized by higher dissimilarities among atoms from different

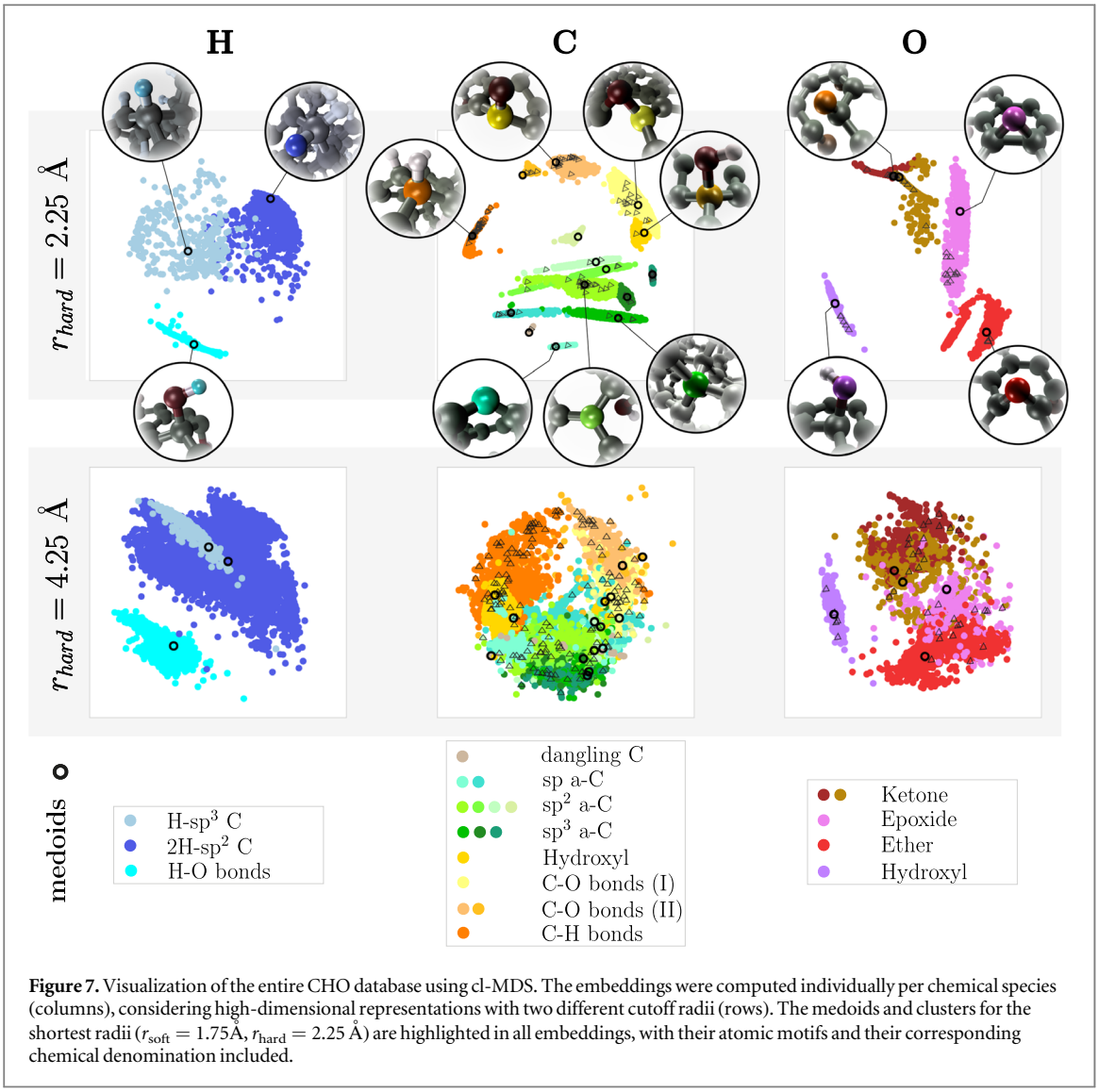

Figure 7. Visualization of the entire CHO database using cl-MDS. The embeddings were computed individually per chemical species (columns), considering high-dimensional representations with two different cutoff radii (rows). The medoids and clusters for the shortest radii (r_{soft} = 1.75Å, r_{hard} = 2.25 Å) are highlighted in all embeddings, with their atomic motifs and their corresponding chemical denomination included.

chemical species, unless the central atom is explicitly neglected. That is, we would expect that a 2-dimensional representation of such sample reflects those dissimilarities with a proper separation of atomic species, particularly for metric-based methods. However, all MDS attempts failed to encode global H and O dissimilarities in terms of the pairwise distances in the embedded space, despite MDS being the quintessential method for pairwise distance preservation. On the other hand, manifold-based methods such as t-SNE and UMAP are misleading in this regard, since neither the distances nor the size of the clusters in their embeddings are meaningful. Even if they retain some global structures, figure 6 shows how relative positions between H clusters were lost.

Second, we incorporate cluster information from *k*-medoids into figure 6. Given that data clustering is independent of the embedding, we can compare how consistently each method preserves it. As expected, cl-MDS outperforms other methods since its algorithm is based on clustering preservation, overcoming the tendency of regular MDS to mix different clusters. Conversely, the usual 'clustered-appearance' of t-SNE maps is only partially consistent with *k*-medoids clustering. Meanwhile, PCA, kPCA and UMAP (to a lesser extent) tend to collapse smaller clusters irrespective of their relevance, obscuring their visualization. In particular, this issue happened with UMAP despite increasing its hyperparameter `min_dist`, which adjusts the minimum distance between embedded points. Moreover, UMAP and t-SNE maps were obtained after performing hyperparameter estimation based on the silhouette score [66] for that very same clustering. We considered other two classifications, but they did not balance as well the local and global characteristics of the sample. The full details of these tests are included in the Supporting Information.

Motivated by the clear separation between C, H and O atoms in figure 6, we computed the cl-MDS maps per chemical species, as shown in figure 7. In this second approach, each embedding includes all atoms from the same species within the CHO database, i.e., 135,618 C atoms, 7669 H atoms and 8269 O atoms. Sparsification support proved itself very handy, especially for carbon, combined with the estimation of the low-dimensional

representation for the whole dataset (see section 2.4). We found that a sparse set containing as few as 2 percent of the whole carbon dataset was representative enough when carefully selected, e.g., using a combination of random and CUR-based data points as well as precomputed medoids. Oxygen and hydrogen datasets, albeit smaller in size, also benefited from sparsification due to its improvement of MDS performance within cl-MDS computations.

Since carbon has the richest structural landscape in this database, we eased the clustering and embedding process for carbon by applying cluster hierarchy, with $h = [15, 7, 3, 1]$ (see section 2.4). We used a simpler clustering for hydrogen and oxygen, with $N_{\mathrm{cl}} = 3$ and $N_{\mathrm{cl}} = 5$, respectively. Besides the usual advantages of k-medoids versus k-means [67] (e.g., allowing for different sizes of clusters, being less sensitive to outliers), the knowledge of the medoids is itself valuable for visualization purposes. That is, we do not only obtain a 2-dimensional plot of the dataset, but also a representative per cluster that we can track back to the sample, as figure 7 illustrates. We have added labels that identify the clusters/medoids with classical chemical configurations (i.e., simple hybridizations and functional groups) to simplify the visualization, associating several clusters with the same label. However, the clustering was thorough enough to further distinguish between groups in terms of other structural features, such as angles or bond distances.

Additionally, figure 7 contains two separate embeddings for each chemical species, computed from SOAP representations of the database with different cutoff radii: the radii already used in figure 6, and a shorter version with $r_{\mathrm{soft}} = 1.75$ Å, $r_{\mathrm{hard}} = 2.25$ Å. These visualizations illustrate the importance of hyperparameter selection depending on the ML application. While the SOAP representation does not change here its original dimensionality (which depends exclusively on the number of basis functions used), the region of SOAP space spanned by the sample increases, effectively becoming more sparsely populated with increased averaged distances between data points, hindering data clustering. Moreover, a small cutoff radius emphasizes first and second neighbors in the representation, allowing an straightforward connection between the embeddings and classical chemical motifs or functional groups in this example.

On the other hand, large radii retain more structural information, despite increasing the complexity of the atomic environments and their overall dissimilarities. That is, the larger cutoff radii are better suited for training ML potentials and evaluating their performance, as opposed to the shorter radii which ease data visualization and classification. For instance, the gray triangles in figure 7 indicate the sparse subset of atomic environments (for C and O atoms) used in [64] to build a ML model that involved expensive calculations. Their visualization in the first row proves that their sampling process preserved the diversity of the database, since all motifs are represented. The sparse set is not homogeneously distributed in those maps though, suggesting that a larger cutoff radii is needed for the selection process, i.e., intuitive motif classification is not informative enough to train a predictive ML model. As a final remark, note that the same radii could be too small for molecular datasets for instance (see section 3.2.2); in practice, the choice of radii depends heavily on the dataset and the purpose of the visualization.

3.2.2. QM9 database

The QM9 database [68] is a subset of a much larger database (the GDB-17 database[69], with 166 billion molecules) carefully selected for a detailed sampling of the chemical space of small organic compounds. In particular, it contains 133,885 neutral organic molecules composed of carbon, hydrogen, oxygen, nitrogen and fluorine, up to nine 'heavy' atoms (C, O, N, F). To represent QM9 we use SOAP vectors of dimension $N_{\mathrm{SOAP}} = 7380$, and cutoff radii $r_{\mathrm{soft}} = 3$ Å, $r_{\mathrm{hard}} = 3.5$ Å. As discussed in section 3.2.1, the difference between chemical species outweighs any other dissimilarity in a combined embedding; consequently, we performed a separate cl-MDS embedding per chemical species. A sparse set of 1000-2000 atoms depending on the species was used, carefully selected by combining k-medoids, random picking and a consistent clustering. The visualization of each atomic species was obtained through the estimation of the low-dimensional coordinates (see section 2.4). Figure 8 shows the resulting cl-MDS embeddings, which help us visualize the composition of the QM9 database. Here we can appreciate once again how cl-MDS performs worse when separating clusters for chemical species with richer variety of atomic environments, e.g., carbon. As opposed to the CHO example in section 3.2.1 (see figure 7), C atoms were embedded with a simple hierarchy, $h = [15, 1]$, reducing the capability of cl-MDS to effectively differentiate additional nuances. Its combination with medoids information nonetheless highlights those subtleties via the corresponding molecular motifs, proving again the value of cluster information. Here, all the medoids were included, revealing the existence of other relevant properties, apart from the chemical species, such as the geometrical arrangement of the molecules. For instance, this is suggested by the absence of fluorine-related clusters in H, N and O embeddings despite its presence in several structures. Independently of the number of clusters, fluorine is not representative enough to weigh in the dissimilarities unless it is a first-nearest neighbor, i.e., with carbon.

Additionally, we can easily identify those clusters whose members have extremely similar descriptors. While all clusters include at least 4 atomic configurations, the visualizations in figure 8 show single-point clusters. This

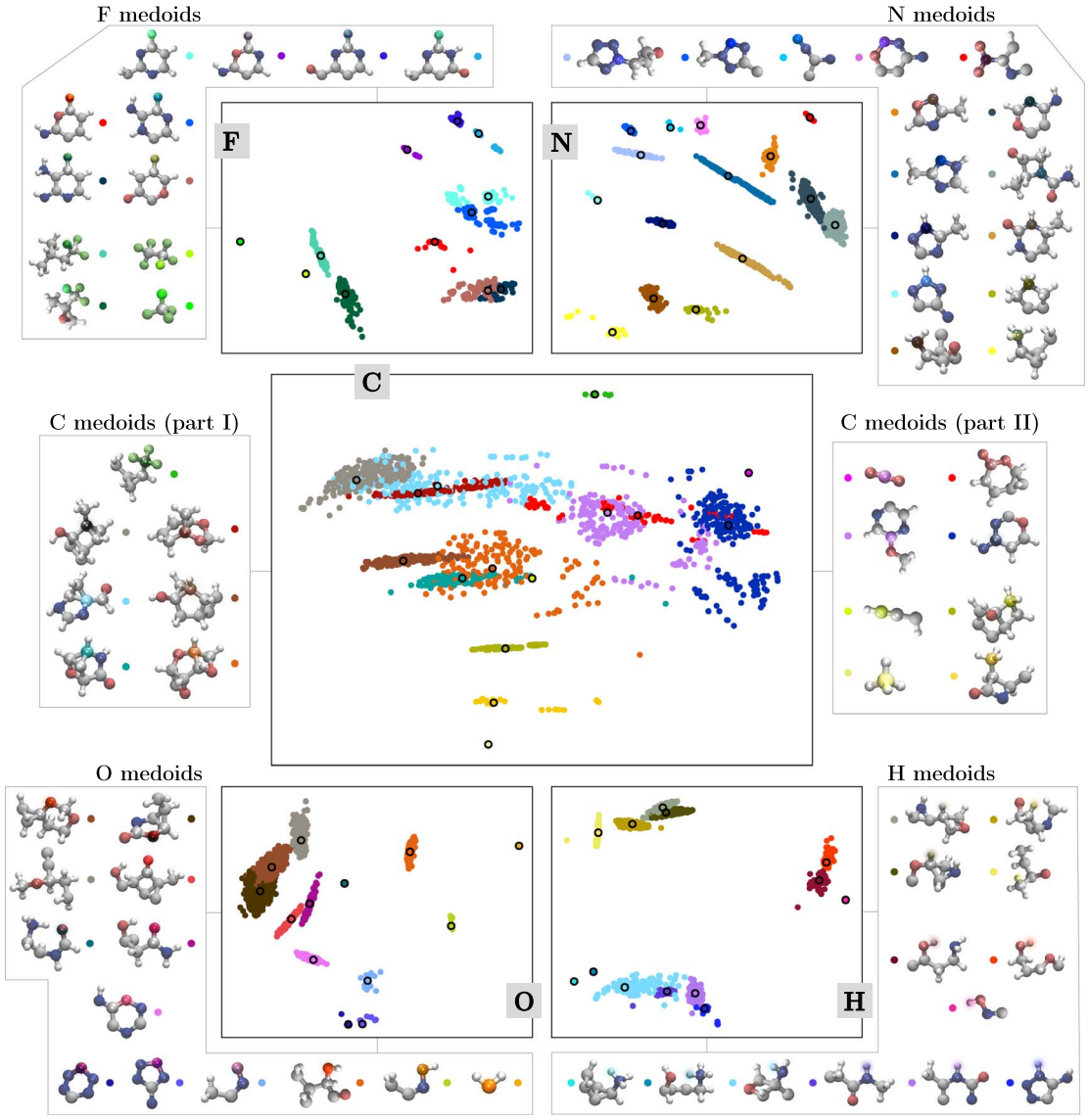

Figure 8. Visualization of the QM9 database through five cl-MDS mappings, one per atomic species (C, H, O, N, F). We show all the medoids and their atomic environments to illustrate the distribution of chemical and structural properties in each embedding, highlighting them in each molecular motif with the following colors: gray (C), white (H), red (O), blue (N) and green (F).

is a consequence of the high similarity between these atomic environments, which is reflected by their high-dimensional representation. That is, the kernel distances between those environments are effectively zero, representing them with the same point in both high- and low-dimensional representations.

3.2.3. Database of PtAu nanoclusters

In previous examples, we used visualizations to navigate the structural and chemical complexity of material and molecular databases. Now, we would like to introduce the potential of cl-MDS for the analysis and interpretation of scientific results obtained from other models, such as density-functional theory (DFT) or ML-driven atomistic simulations. For instance, we could assess the performance of a ML-based Gaussian approximation potential (GAP) [70] by generating a database and checking some physical and/or chemical properties on top of cl-MDS. We invite the reader to check the literature for further ideas about similar applications of dimensionality reduction methods, e.g., see [57] for other examples in material science and chemistry.

For this example, we chose a general-purpose ML potential for modelling Pt and Au systems and computed a database of PtAu alloy nanoparticles (NPs). We want to better understand the site distribution of this database, emphasizing any differences between 'bulk' and surface atoms, which might give useful insight in applications in catalysis. Note, however, that there is no proper bulk in these NPs due to their reduced size. That said, the analysis workflow should be also applicable to larger structures without modification. The database was

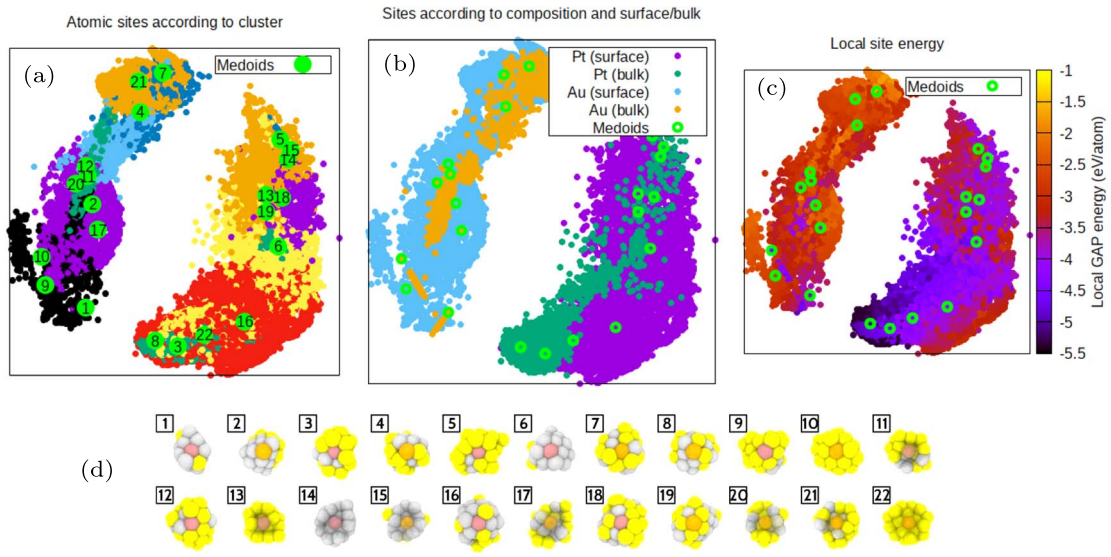

Figure 9. Visualization of a GAP-generated database of PtAu nanoclusters, using cl-MDS to illustrate (a) *k*-medoids clustering, (b) atomic sites in terms of composition and bulk/surface character, and (c) local energy predicted by the PtAu GAP. Those sites labeled as medoids are shown in panel (d), highlighting the central atom. Note that whenever a medoid in (d) in in the NP interior, some of the surface atoms are made transparent to allow for visualization of the medoid.

generated using the TurboGAP code [61], with 10 NPs per size and composition. It has 660 unique NPs, including pure Pt and Au ones. Each atomic site is represented by a SOAP descriptor of size $N_{\mathrm{SOAP}} = 279$, using cutoff radii $r_{\mathrm{soft}} = 5.2$ Å, $r_{\mathrm{hard}} = 5.7$ Å. In this case, we computed the representation using SOAP compression [71], which already reduces the dimension of the SOAP vector.

The resulting cl-MDS embedding is shown in figure 9, with hierarchy $h = [22, 1]$. The entire set of atomic sites is 24 750, with 500 sparse data points selected using `sparsify="cur"`. When computing the full distance matrix is computationally affordable (memory-wise), CUR matrix decomposition is a suitable sparsification option for evenly distributed samples, where relevant members are equally represented. Those data points outside the sparse set were assigned estimated coordinates following section 2.4 recipe. As expected, the clustering is influenced by the chemical species (Pt/Au) and its relative position in the NP (surface/'bulk'). Comparing figures 9 (b) and (c), we get a precise quantitative assessment of properties that we know to be true qualitatively, i.e., we can quantify and visualize chemical intuitive concepts. Here, the visualization illustrates how the cohesive energy of atoms naturally increases with atomic coordination, i.e., the latter produces a stabilizing effect moving from the interior sites to the surface sites.

4. Conclusions and outlook

We have introduced a novel technique for data visualization called cluster MDS, which aims to capture high-dimensional local and global features adequately in a single 2-dimensional representation. This issue is inadequately addressed by older methods due to limitations of their dimensionality reduction algorithms. More recent techniques still experience important limitations in this regard, such as the need for a 'balancing' parameter that may crucially impact the structures preserved (e.g., UMAP, GLSPP [72]), or the imposition of specific metrics that limit their application to other fields (e.g., PHATE, DGL[73]).

The cl-MDS algorithm is based on a combination of data clustering and data embedding through *k*-medoids and metric MDS, respectively, with any distance matrix (or dissimilarity measure) as accepted input. We have illustrated the effect of its main hyperparameter, the number of clusters, which can capture the local nuances in the visualization without affecting the overall structure preservation, once a minimum value is reached. This value depends on the dataset and impacts the quality of cl-MDS mappings, rendering our heuristic selection in this paper unsatisfactory. We will explore available criteria to automate this choice and increase its reliability. In its more advanced form, this hyperparameter accepts a hierarchy of clusters that eases the embedding of highly complex data and aids the visualization of big amounts of unlabelled data.

Additionally, cl-MDS can estimate the 2-dimensional coordinates of other points once a first embedding has been obtained. Given that our method partially inherits the decreasing performance of MDS with dataset size, this estimation is extremely useful when combined with the included sparsification support. However, its quality

strongly depends on the chosen sparse set and needs to be carefully assessed. We will update the options for automated sparse selection available in the code as more robust alternatives arise. Other future improvements of the code include the optimization of its computational speed and memory requirements.

Its comparison with well-known methods such as PCA, kPCA, t-SNE, MDS and UMAP showed that cl-MDS improved visualization of different layers of locality, most notably compared to MDS performance. We applied it to datasets of sizes 10^3 to 10^6 and dimensionality up to 7380. In particular, we focused on atomic-structure examples to showcase all the functionality and advantages of this embedding tool, which includes specific recipes for atomic databases. Despite the value of manifold unfolding in other contexts, metric preservation is arguably the best approach for atomic databases. The comparison of atomic structures, as well as the study of their properties, usually involves some sort of similarity measure; therefore, its retention is invaluable in any visualization. Beyond the application to atomic structure datasets highlighted in this paper, we remark that cl-MDS is a general visualization tool. Any dataset with relevant local and global structures can benefit from its use, whenever the data accepts the definition of a meaningful metric.

Acknowledgments

The authors are grateful for financial support from the Academy of Finland under projects #321713, #329483, #330488 and #347252, as well as computational resources provided by CSC—IT Center for Science and Aalto University's Science-IT Project.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://github.com/mcaroba/cl-MDS.

Appendix Embedding atomic structure representations

We have added functionality to the cl-MDS code specifically tailored for processing atomic structure information. First, the distance matrix **D** can be substituted by a file as input parameter, given in a format compatible with the Atomic Simulation Environment (`ASE`) Python library [74, 75]. This file may include a single atomic structure, a concatenation of them, or a trajectory. Given this input, the user has three options: (1) choosing a cutoff radius, relying on the code defaults; (2) selecting a representation for the atomic descriptors, being automatically computed by the cl-MDS code; or (3) providing an array of precomputed descriptors, e.g., obtained with an external tool such as `DScribe` [76].

Option (2) has currently three supported representations (`descriptor="quippy_soap"|"quippy_soap_turbo"|"quippy_soap_turbo_compress"`) available via `quippy`, a Python interface to the Fortran code `QUIP` [77] generated by `f90wrap` [78]. The first option, `"quippy_soap"`, describes atomic environments using the smooth overlap of atomic positions (SOAP) vectors [22]. The second option, `"quippy_soap_turbo"`, uses an optimized version [79] of the previous SOAP atomic descriptors through the `soap_turbo` library [80]. Additionally, a string of hyperparameter definitions (`descriptor_string`) can be given for both descriptor options. The third option, `"quippy_soap_turbo_compress"`, enables SOAP compression [71] while using the default hyperparameters, computing similarly accurate SOAP vectors with lower dimensionality. Other implemented arguments include `average_kernel`, which extends the previous mathematical representations to whole structures (only available with `"quippy_soap"`); and `do_species`, which allows the pre-selection of certain species within a database to speed up the calculations.

Once we have a representation of the atomic structures, we need to obtain its associated distance matrix. A suitably constructed atomic descriptor provides the basis for defining a metric that can be fed into the cl-MDS algorithm. That is, these vectors are used to build a (usually) positive-definite kernel with an induced Gram matrix **K** of size $N \times N$, where N is the length of the database. For SOAP this kernel function is defined as [22]

$$K_{ij} = \left(\mathbf{q}_i^{\mathrm{SOAP}} \cdot \mathbf{q}_j^{\mathrm{SOAP}}\right)^{\zeta}, \tag{13}$$

where $\mathbf{q}_i^{\mathrm{SOAP}}$ denotes the SOAP descriptor of atom i, ζ can be any positive scalar and $0 \leqslant K_{ij} \leqslant 1$. The importance of defining a kernel resides in its connection with a dissimilarity measure. As shown in [81], it is always possible to find a dissimilarity measure **D** associated to a positive definite kernel **K**, given by

$$(D_{ij})^2 = \frac{1}{2}(K_{ii} + K_{jj}) - K_{ij}. \tag{14}$$

Considering polynomial kernels such as equation (13) are always positive definite, the resulting SOAP dissimilarity matrix is

$$D_{ij} = \sqrt{1 - K_{ij}}. \tag{15}$$

This distance measure is automatically computed by the cl-MDS code from the set of atomic descriptors prior to performing the clustering and embedding steps.

In addition to the embedding features described in section 2.4, we have also added the possibility of fine-tuning the minimization process of the MDS stress from step 4 (see section 2) by means of a weighted distance matrix. While our MDS implementation can generally accommodate weights, this is a straightforward extension for atomic structure visualization based on the SOAP kernel, which is evaluated for the set of medoids $\mathcal{M}$. That is, given two atoms $i \in \mathcal{C}_k$, $j \in \mathcal{C}_s$ and a positive integer η, the distance is redefined as

$$D_{ij}^{(w)} = \sqrt{1 - K_{ij} \cdot (K_{m_k,m_s})^{\eta}}. \tag{16}$$

Thus,

$$\mathbf{D}^{(w)} = [1 - \mathbf{K} \odot (\mathbf{K}_{\mathcal{M}})^{\odot\eta}\,]^{1/2}, \tag{17}$$

where $\odot$ denotes the element-wise product (also known as Hadamard or Schur product) and element-wise exponentiation, and

$$\begin{aligned} (K_{\mathcal{M}})_{ij} &= K_{m_k,m_s}, && \text{if } i \in \mathcal{A}_k \text{ and } j \in \mathcal{A}_s, \\ (K_{\mathcal{M}})_{ij} &= 1, && \text{if and only if } i, j \in \mathcal{A}_k. \end{aligned}$$

These weighted distances afford us greater control in decoupling the representation of individual clusters on the global map, effectively allowing us to continuously increase the emphasis on the local vs global structure of the data.

ORCID iDs

Patricia Hernández-León https://orcid.org/0000-0002-3402-2553
Miguel A Caro https://orcid.org/0000-0001-9304-4261

References

[1] Lui K, Ding G W, Huang R and McCann R 2018 *Advances in Neural Information Processing Systems NeurIPS* 31, 8453
[2] Hotelling H 1933 *Journal of Educational Psychology* **24** 417–41
[3] van der Maaten L and Hinton G 2008 *Journal of Machine Learning Research* **9** 2579–605
[4] van der Maaten L 2014 *Journal of Machine Learning Research* **15** 3221–45
[5] Tenenbaum J, Silva V and Langford J 2000 *Science* **290** 2319–23
[6] Kruskal J B 1964 *Psychometrika* **29** 1–27
[7] Kruskal J B 1964 *Psychometrika* **29** 115–29
[8] Borg I and Groenen P J F 2005 *Modern Multidimensional Scaling* (Springer) (https://doi.org/10.1007/0-387-28981-X)
[9] Bertini E, Tatu A and Keim D 2011 *IEEE Trans. Vis. Comput. Graph.* **17** 2203–12
[10] Tsai F S 2012 *Expert Syst. Appl.* **39** 1747–52
[11] McInnes L, Healy J and Melville J 2018 Umap: Uniform manifold approximation and projection for dimension reduction arXiv:1802.03426 accessed on May 2 2024
[12] Moon K R *et al* 2019 *Nat. Biotechnol.* **37** 1482–92
[13] Caro M A and Hernández-León P (2018) *cl-MDS repository* accessed on May 2 2024: https://github.com/mcaroba/cl-MDS
[14] Yu G-J and Wang S-C 2008 *22nd International Conference on Advanced Information Networking and Applications* 748–54
[15] Shon M, Choi W and Choo H 2010 *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery* 42–7
[16] Saeed N and Nam H 2016 *IEEE Trans. Signal Process.* **64** 2649–59
[17] Kaufman L and Rousseeuw P 1987 *Statistical Data Analysis Based on the L1 Norm and Related Methods* 405–16
[18] Bauckhage C 2015 *Numpy/scipy recipes for data science: k-medoids clustering* (https://doi.org/10.13140/2.1.4453.2009)
[19] Macqueen J 1967 *Proceedings of the V Berkeley Symposium on Mathematical Statistics and Probability* 1 281
[20] Hartigan J A and Wong M A 1979 *Journal of the Royal Statistical Society. Series C (Applied Statistics)* **28** 100–8
[21] Ng A Y, Jordan M I and Weiss Y 2001 *Advances in Neural Information Processing Systems 14 (NIPS 2001)* (MIT Press) 849–56
[22] Bartók A P, Kondor R and Csányi G 2013 *Phys. Rev.* B **87** 184115
[23] Caro M A and Hernández-León P (2021) *fast-kmedoids repository* accessed on May 2 2024: https://github.com/mcaroba/fast-kmedoids
[24] Peterson P 2009 *Int. J. Computational Science and Engineering* **4** 296–305
[25] Caro M A, Aarva A, Deringer V L, Csányi G and Laurila T 2018 *Chem. Mater.* **30** 7446–55
[26] Pedregosa F *et al* 2011 *Journal of Machine Learning Research* **12** 2825–30
[27] Groenen P J F and van de Velden M 2016 *Journal of Statistical Software* **73** 1–26
[28] Sommerville D M Y 1929 *An Introduction to the Geometry of N Dimensions* (Methuen & Co.)

[29] Gritzmann P and Klee V 1994 *Discrete Math.* **136** 129–74
[30] Schneider P J and Eberly D H 2003 *Geometric Tools for Computer Graphics* (Morgan Kaufmann)
[31] Barber C B, Dobkin D P and Huhdanpaa H T 1996 *ACM Trans. on Mathematical Software* **22** 469–83
[32] Virtanen P *et al* 2020 *Nat. Methods* **17** 261–72
[33] Richter-Gebert J 2011 *Perspectives on Projective Geometry: A Guided Tour Through Real and Complex Geometry* (Springer Berlin) (https://doi.org/10.1007/978-3-642-17286-1)
[34] Eberly D H 2011 accessed on May 2 2024: https://www.geometrictools.com/Documentation/PerspectiveMappings.pdf
[35] Hastie T, Tibshirani R and Friedman J 2009 *The Elements of Statistical Learning* (Springer Nueva York) (https://doi.org/10.1007/978-0-387-84858-7)
[36] Harris C R *et al* 2020 *Nature* **585** 357–62
[37] Drineas P, Kannan R and Mahoney M W 2006 *SIAM J. Comput.* **36** 184–206
[38] Mahoney M W, Maggioni M and Drineas P 2008 *SIAM J. Matrix Anal. Appl.* **30** 957–87
[39] Mahoney M W and Drineas P 2009 *Proceedings of the National Academy of Sciences* **106** 697–702
[40] Leskovec J, Rajaraman A and Ullman J D 2020 *Mining of Massive Datasets* (Cambridge University Press)
[41] Roweis S T and Saul L K 2000 *Science* **290** 2323–6
[42] Zhang Z and Wang J 2006 *Advances in Neural Information Processing Systems 19 (NIPS 2006)* (MIT Press) 1593–600
[43] Donoho D L and Grimes C 2003 *Proceedings of the National Academy of Sciences* **100** 5591–6
[44] Zhang Z and Zha H 2004 *SIAM J. Sci. Comput.* **26** 313–38
[45] Belkin M and Niyogi P 2003 *Neural Comput.* **15** 1373–96
[46] Schölkopf B, Smola A and Müller K-R 1998 *Neural Comput.* **10** 1299–319
[47] Chen L and Buja A 2009 *J. Am. Stat. Assoc.* **104** 209–19
[48] Venna J and Kaski S 2006 *Neural Netw.* **19** 889–99
[49] Vanderplas J 2024 Comparison of Manifold Learning methods, accessed on May 2 2024: https://scikit-learn.org/stable/auto_examples/manifold/plot_compare_methods.html
[50] Liu Y, Li Z, Xiong H, Gao X and Wu J 2010 *IEEE International Conference on Data MiningICDM* 911–6
[51] Halkidi M, Batistakis Y and Vazirgiannis M 2001 *Journal of Intelligent Information Systems* **17** 107–45
[52] Kaufman L and Rousseeuw P J 1990 *Finding Groups in Data: An Introduction to Cluster Analysis* (Wiley) (https://doi.org/10.1002/9780470316801)
[53] Willartt M J, Musli F and Ceriotti M 2019 *J. Chem. Phys.* **150** 154110
[54] De S, Bartók A P, Csányi G and Ceriotti M 2016 *Phys. Chem. Chem. Phys.* **18** 13754
[55] De S and Ceriotti M (2019) *Interactive Sketchmap Visualizer* Zenodo (https://doi.org/10.5281/zenodo.3541831)
[56] Fraux G, Cersonsky R and Ceriotti M 2020 *Journal of Open Source Software* **5** 2117
[57] Cheng B *et al* 2020 *Accounts Chem. Res.* **53** 1981–91
[58] Ceriotti M, Tribello G A and Parrinello M 2011 *Proceedings of the National Academy of Sciences* **108** 13023–8
[59] Coifman R R, Lafon S, Lee A B, Maggioni M, Nadler B, Warner F and Zucker S W 2005 *Proceedings of the National Academy of Sciences* **102** 7426–31
[60] Coifman R R and Lafon S 2006 *Appl. Comput. Harmon. Anal.* **21** 5–30
[61] Caro M A 2024 TurboGAP, accessed on May 2 2024 : turbogap.fi
[62] Muhli H *et al* 2021 *Phys. Rev.* B **104** 054106
[63] Jana R and Caro M A 2023 *Phys. Rev.* B **107** 245421
[64] Golze D *et al* 2022 *Chem. Mater.* **34** 6240–54
[65] van der Maaten L, Postma E O and van den Herik J 2009 *Tilburg University Technical Report* TiCC-TR 2009-005
[66] Rousseeuw P J 1987 *J. Comput. Appl. Math.* **20** 53–65
[67] Arora P, Deepali D and Varshney S 2016 *Procedia Computer Science* **78** 507–12
[68] Ramakrishnan R, Dral P O, Rupp M and von Lilienfeld O A 2014 *Scientific Data* **1**
[69] Ruddigkeit L, van Deursen R, Blum L C and Reymond J L 2012 *J. Chem. Inf. Model.* **52** 2864–75
[70] Bartók A P, Payne M C, Kondor R and Csányi G 2010 *Phys. Rev. Lett.* **104** 136403
[71] Klawohn S, Darby J P, Kermode J R, Csányi G, Caro M A and Bartók A P 2023 *J. Chem. Phys.* **159**
[72] Cai W 2017 *Knowl.-Based Syst.* **118** 191–203
[73] Song Y, Wang J, Qi L, Yuan W, Zhensu M Yu and Qu J 2017 *7th IEEE International Conference on Electronics Information and Emergency Communication (ICEIEC)* 244–7
[74] Bahn S R and Jacobsen K W 2002 *Comput. Sci. Eng.* **4** 56–66
[75] Larsen A H *et al* 2017 *J. Phys.: Condens. Matter* **29** 273002
[76] Himanen L *et al* 2020 *Comput. Phys. Commun.* **247** 106949
[77] Csányi G *et al* 2007 *IoP Comput. Phys. Newsletter* **Spring** 2007
[78] Kermode J R 2020 *J. Phys. Condens. Matter* **32** 305901
[79] Caro M A 2019 *Phys. Rev.* B **100** 024112
[80] Caro M A 2019 soap_turbo repository, accessed on May 2 2024: https://github.com/libAtoms/soap_turbo
[81] Schölkopf B 2000 *Advances in Neural Information Processing Systems 13 (NIPS 2000)* (MIT Press)