MDSCAN: RMSD-based HDBSCAN clustering of long molecular dynamics

Roy González-Alemán, Daniel Platero-Rochart, Alejandro Rodríguez-Serradet, Erix Hernández-Rodríguez, Julio Caballero, Fabrice Leclerc, Luis Montero-Cabrera

▶ **To cite this version:**

Roy González-Alemán, Daniel Platero-Rochart, Alejandro Rodríguez-Serradet, Erix Hernández-Rodríguez, Julio Caballero, et al.. MDSCAN: RMSD-based HDBSCAN clustering of long molecular dynamics. Bioinformatics, 2022, 38 (23), pp.5191-5198. 10.1093/bioinformatics/btac666 . hal-03938219

HAL Id: hal-03938219
https://hal.science/hal-03938219

Submitted on 13 Jan 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: Day Month Year
Original Paper

OXFORD

Structural Bioinformatics

MDSCAN: RMSD-Based HDBSCAN Clustering of Long Molecular Dynamics

Roy González-Alemán [1, 2,*], Daniel Platero-Rochart [1], Alejandro Rodríguez-Serradet [1], Erix W. Hernández-Rodríguez [3], Julio Caballero [4], Fabrice Leclerc [2,*] and Luis Montero-Cabrera [1]

[1]Laboratorio de Química Computacional y Teórica (LQCT), Facultad de Química, Universidad de La Habana, La Habana, 10400, Cuba
[2]Institute for Integrative Biology of the Cell (I2BC), CEA, CNRS, Université Paris Saclay,Gif-sur-Yvette, F-91198, France
[3]Laboratorio de Bioinformática y Química Computacional, Departamento de Medicina Traslacional, Facultad de Medicina, Universidad Católica del Maule, Talca 3480094, Chile
[4]Departamento de Bioinformática, Facultad de Ingeniería, Centro de Bioinformática, Simulación y Modelado (CBSM), Universidad de Talca, Talca, Chile

*To whom correspondence should be addressed.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Motivation: The term clustering designates a comprehensive family of unsupervised learning methods allowing to group similar elements into sets called clusters. Geometrical clustering of Molecular Dynamics (MD) trajectories is a well-established analysis to gain insights into the conformational behavior of simulated systems. However, popular variants collapse when processing relatively long trajectories because of their quadratic memory or time complexity. From the arsenal of clustering algorithms, HDBSCAN stands out as a hierarchical density-based alternative that provides robust differentiation of intimately related elements from noise data. Although a very efficient implementation of this algorithm is available for programming-skilled users (*HDBSCAN**), it cannot treat long trajectories under the *de facto* molecular similarity metric RMSD.

Results: Here, we propose *MDSCAN*, an HDBSCAN-inspired software specifically conceived for non-programmers users to perform memory-efficient RMSD-based clustering of long MD trajectories. Methodological improvements over the original version include the encoding of trajectories as a particular class of vantage-point tree (decreasing time complexity), and a dual-heap approach to construct a quasi-minimum spanning tree (reducing memory complexity). *MDSCAN* was able to process a trajectory of one-million frames using the RMSD metric in about 21 hours with less than 8 GB of RAM, a task that would have taken a similar time but more than 32 TB of RAM with the accelerated *HDBSCAN** implementation generally used.

Availability and implementation: The source code and documentation of *MDSCAN* are free and publicly available on GitHub (https://github.com/LQCT/MDScan.git) and as a PyPI package (https://pypi.org/project/mdscan/).

Contact: roy_gonzalez@fq.uh.cu, fabrice.leclerc@i2bc.paris-saclay.fr

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Molecular Dynamics (MD) is a physical-based computational technique aiming to describe the dynamic evolution of a system at the atomic level.

© The Author 2021. Published by Oxford University Press. All rights reserved. For permissions, please e-mail: journals.permissions@oup.com

Widely applied in the structural bioinformatics field, MDs are useful to understand, complement, or even predict experimental results. The primary outcome of an MD simulation is its trajectory, a compendium of the explored conformations sampled at specific periods. Nowadays, many snapshots can be contained in MD trajectories whose complexity often hides their inherent relationship. By grouping (clustering) similar conformations, it becomes easier to correlate MD results with experimental data as the relative importance of the sampled conformations becomes evident.

Geometrical clustering is one of the most frequently found post-processing analyses of MD. Although many options are available to the user (Shao *et al.*, 2007; Peng *et al.*, 2018), their results are not always analogous, as they assume different cluster definitions. Density-based variants represent clusters as regions of high density surrounded by noisy low-density zones. This notion, translated to the MD jargon, is the equivalent of defining a cluster as a temporary stable region of the conformational landscape.

From the density-based clustering alternatives (Campello *et al.*, 2020), HDBSCAN (Campello *et al.*, 2013) has proved one of the most robust currently accessible solutions. This method generates a complete hierarchy of the most significant and stable clusters through two intuitive parameters (which, when equally set, generate a pseudo-non-parametric algorithm): the minimum size of a cluster (m), and the k^{th} nearest neighbor to consider when determining an element's neighborhood (k).

Among the HDBSCAN's advantages described in the original paper, the following are of particular interest: (i) the ability to characterize datasets with nested clusters or clusters of different densities (a challenging task with other variants like DBSCAN (Schubert *et al.*, 2017) or DENCLUE (Hinneburg and Keim, 2003)), (ii) the straightforward simplification of the cluster hierarchy into an easily interpretable representation of the most significant clusters (as opposed to methods like gSkeletonClu (Sun *et al.*, 2010)), (iii) the fact of not being circumscribed to specific classes of problems (like gSkeletonClu) or element sets in the real coordinate space (like DECODE (Pei *et al.*, 2009) or Generalized Single-Linkage (Stuetzle and Nugent, 2010)), and (iv) the non-reliance on multiple (often critical) input parameters like the mentioned algorithms and many others.

From the previous list, DBSCAN (implemented in the *cpptraj* module of the AMBER suite) is an appealing choice for the analysis of MD trajectories. As stated by Schubert *et al.*, it is an algorithm proven to work in practical situations, that received the *SIGKDD test-of-time Award* in 2014 (Schubert *et al.*, 2017). Conceptually, HDBSCAN supersedes DBSCAN, being able to report clusters over all values of the DBSCAN's distance scale parameter ϵ and find those clusters that persist for many values of this magnitude.

Though not primarily conceived to deal with molecular ensembles, HDBSCAN has been used successfully in the conformational study of MD simulations (Melvin *et al.*, 2016, 2018) through a deeply optimized implementation referred as *HDBSCAN** from now on (McInnes and Healy, 2017). *HDBSCAN**'s authors creatively addressed each major step of the original version, reducing their complexity from $O(N^2)$ to near $O(N\ log\ N)$ in the average case (and even for the worst cases, a fast sub-quadratic complexity of the algorithm is expected).

Unfortunately, *HDBSCAN** excludes RMSD (the *de facto* metric in MD analysis despite its thoroughly described drawbacks (Sargsyan *et al.*, 2017)) or any other high dimensional metric to assess the pairwise similarity of elements. Although *HDBSCAN** can receive a pre-processed RMSD float square matrix, its explicitly fixed double-precision float data type bounds the range of applications to small MD trajectories.

Euclidean or Euclidean-like metrics are exploited in MD clustering under the assumption that they are faster to compute than the optimal RMSD (as less unitary operations are involved, and also because no alignment is performed between every pair of structures). However, we have previously shown that efficient approaches of Theobald's method for computing the optimal RMSD outperform most clustering software based on Euclidean or RMSD metrics (González-Alemán *et al.*, 2020a,b, 2021; Platero-Rochart *et al.*, 2022).

Here, we propose *MDSCAN*, a fast and memory-efficient RMSD-based implementation of HDBSCAN that is suitable to process long MD trajectories with no distance matrix involved. *MDSCAN*, similar to *HDBSCAN**, is an approximate approach to the reference implementation whose moderate deviations make a suitable compromise between the computational cost of the clustering job and the quality of returned clusters. The encoding of MD trajectories as a distinctive variant of vantage-point trees (decreasing the run times of RMSD computation up to a half) and a double-heap approach to calculate a quasi-minimum spanning tree of the MD trajectory's complete graph (significantly decreasing the RAM usage) are the major methodological contributions of this work.

MDSCAN clustered a 1 million frames MD trajectory using the RMSD metric in about 21 hours with less than 8 GB of RAM. The same job would have taken a similar time, but over 32 TB of RAM in the RMSD version of *HDBSCAN**. Our proposal is designed as a command-line interface for non-programmer users and provides helpful VMD visualization scripts.

2 Computational Details

MDSCAN has been coded in Python 3 programming language and is freely available at GitHub (https://github.com/LQCT/MDScan.git) and as a PyPi package (https://pypi.org/project/mdscan/). It depends on version 1.9.4 of MDTraj (McGibbon *et al.*, 2015) for the fast optimal RMSD calculations.

To contrast the computational performance of *MDSCAN* with other density-based clustering alternatives, we designed a benchmark against the following software: (i) the *cpptraj* (Roe and Cheatham, 2013) implementation of DBSCAN (using the RMSD metric), (ii) the *scikit-learn* (Pedregosa *et al.*, 2011) implementation of DBSCAN (using the RMSD metric), and (iii) the accelerated *HDBSCAN** implementation of McInnes and Healy (McInnes and Healy, 2017) (using the Euclidean and RMSD metrics).

The benchmark was conducted on a set of previously published trajectories that are referred by their size (kF = kilo Frame, MF = mega Frame) as follows: (i) 6 kF, a 6001 frames REMD simulation of the Tau peptide (Shea and Levine, 2016), (ii) 30 kF, a 30605 frames MD of villin headpiece (PDB 2RJY) (Melvin *et al.*, 2016), (iii) 50 kF, a 50500 frames MD of serotype 18C of *Streptococcus Pneumoniae* (González-Alemán *et al.*, 2021), (iv) 100 kF, a 100500 frames MD of Cyclophilin A (PDB 2N0T) (González-Alemán *et al.*, 2021), (v) 250 kF, a 250000 frames MD of four chains of the Tau peptide that corresponds to the MD simulation of an extended Tau peptide (PDB PHF8) during 1μs developed by Álvarez-Ginarte *et al.*(Laboratory of Computational and Theoretical Chemistry, University of Havana) in an unpublished work, (vi) 500 kF. a 500000 frames MD toy trajectory constructed from randomly selected snapshots of 6K (González-Alemán *et al.*, 2020a), and (vii) 1 MF, a one-million frames MD of ubiquitin (PDB 1UBQ) (Platero-Rochart *et al.*, 2022).

All trajectory and topology files are available online at the following addresses: 6 kF, 50 kF, 100 kF, 250 kF, 500 kF at https://doi.org/10.6084/m9.figshare.c.5403930.v1, 30 kF at https://doi.org/10.6084/m9.figshare.3983526.v1, and 1 MF at https://lbqc.ucm.cl/ubiquitin_1MF/.

Though all trajectories and topologies in this study are formatted as .dcd and .pdb respectively, *MDSCAN*'s users can choose from any of the available formats MDTraj provides, namely [arc, dcd, binpos, xtc, trr, hdf5, h5, ncdf, netcdf, nc, pdb.gz, pdb, lh5, crd, mdcrd, inpcrd, restrt, rst7, ncrst, lammpstrj, dtr, stk, gro, xyz.gz, xyz, tng, xml, mol2, hoomdxml,

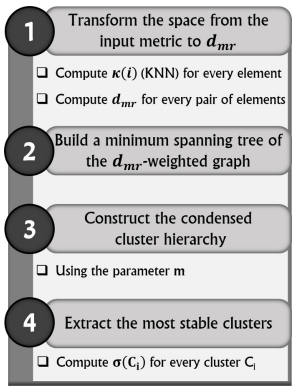

Fig. 1. Main steps of the HDBSCAN clustering algorithm.

and gsd] for trajectories, and [pdb, pdb.gz, h5, lh5, prmtop, parm7, prm7, psf, mol2, hoomdxml, gro, arc, hdf5, and gsd] for topologies.

Calculations were performed on an AMD Ryzen 5 Hexa-core Workstation with a processor speed of 3.6 GHz and 64 GB RAM under a 64-bit Xubuntu 18.04 operating system. Run times and RAM peaks were recorded with the */usr/bin/time* Linux command.

3 HDBSCAN Formalism

Like other density-based clustering algorithms, the main idea behind HDBSCAN is to isolate high-density regions from the lower-density or noisy zones. In the context of an MD trajectory, denser regions can be regarded as a highly-populated location of the system's conformational space, surrounded by transitional or noisy conformations. The main steps of this algorithm are detailed in Figure 1.

HDBSCAN formally defines the density of each frame i in terms of a *core distance* $\kappa(i)$; the distance from i to its k^{th} nearest neighbor. Note that the chosen metric for $\kappa(i)$ can be Euclidean, RMSD, or any other selected by the user. Computing $\kappa(i)$ for every frame of the trajectory permits to effectively spread apart denser frames from noise by defining a new similarity metric, the *mutual reachability distance* (d_{mr}, Equation 1), in which $d(i,j)$ is the distance between elements i and j in the input metric. Under d_{mr}, dense conformations (having low $\kappa(i)$) remain at the same original distance from each other while sparser frames are "pushed" to be at least their core distance away from any other point.

$$d_{mr}(i,j) = \begin{cases} max\{\ \kappa(i), \kappa(j), d(i,j)\ \}, & i \neq j \\ 0\ , & i = j \end{cases} \quad (1)$$

From a graph-theoretic point of view, a molecular trajectory can be seen as a complete graph (T) in which nodes represent frames, and pairwise edges hold the d_{mr} distance between nodes. In this scenario, creating a hierarchical divisive partition of T can proceed by setting a high threshold value ($dist$) at which to start erasing edges, in a way that T would pass from a complete graph to a completely disconnected one. As this naive approach is computationally prohibitive, HDBSCAN recurs to the construction of a *minimum spanning tree* (MST) whose progressive disconnection leads to the same hierarchy of components just described. An MST of T is a subset of T edges that connects all T nodes (without forming cycles) with the minimum total weight.

The MST inferred from T can be progressively disconnected to produce a hierarchy of clusters. HDBSCAN introduces a parameter m that represents the minimum number of points in a component to classify it as a cluster. m allows condensing the cluster hierarchy because now, cutting an edge that produces a component with less than m points is considered as a *"just-loosing-elements"* cluster and not as an independent one.

Concretely, the MST disconnection process (Figure 2) proceeds in this fashion: A new magnitude λ is defined as the inverse of the d_{mr} distance ($\lambda = 1/dist$). MST' edges are sorted in increasing order of their λ value (high distances edges come first). Successive edge cutting produces two child sub-trees at each cleavage, giving rise to one of the following situations: (i) one of the child sub-trees contains m or fewer points, (ii) both child sub-trees includes m or fewer points, and (iii) both child sub-trees carries more than m points. In the first situation, a component without the lost members is retained, and the split is considered spurious. No child is returned, only a shrink component. The second situation marks the MST disconnection endpoint, as no further valid components will be produced. The third case corresponds to a "true split", and effectively separates the parent component into two new independent ones.

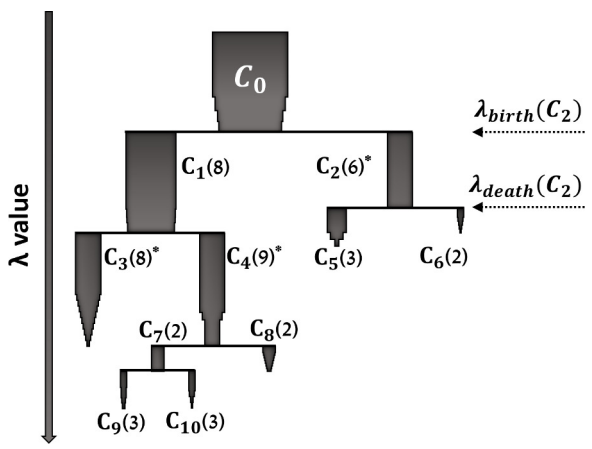

Fig. 2. Condensed hierarchy of clusters produced by the HDBSCAN's disconnection process. The stability of cluster C_x is inside parentheses. Final selected clusters have an asterisk. Components' size is scaled by their relative width.

The extraction of final clusters from the condensed hierarchy takes place according to the definition of *cluster stability* ($\sigma(C_i)$, Equation 2). First, for a given cluster C_i (see C_2 in Figure 2) let's define its λ_{birth} and λ_{death} as the λ values at which C_i becomes a cluster and disappears respectively. Inside C_i, for each element e, the λ_e value denotes when e "abandons" C_i, either as a spurious or a true split (note that $\lambda_{birth} < \lambda_e < \lambda_{death}$). Then the stability of C_i is calculated through the Equation 2.

$$\sigma(C_i) = \Sigma_{e \in C_i}(\lambda_e - \lambda_{birth}) \quad (2)$$

Once $\sigma(C_i)$ of all hierarchical clusters are computed, the final step is to find a flat (non-hierarchical) set of disjoint clusters with maximum stability. To that end, the cluster tree is processed from the leaves ($C_3, C_9, C_{10}, C_8, C_5$, and C_6 in Figure 2) upwards. Initially, all leaves are declared as clusters. Then, the stabilities of sibling leaves i, j (sharing the same parent k) are summed and the result is compared to the stability of their parent . If $\sigma(i) + \sigma(j) > \sigma(k)$, $\sigma(k)$ is set to $\sigma(i) + \sigma(j)$, but i and j (not k) are still considered the selected clusters. On the contrary, if $\sigma(i) + \sigma(j) \leq \sigma(k)$, $\sigma(k)$ conserve its value, k is marked as selected cluster, and all descendants of k are unselected.

In Figure 2 (cluster stability values are inside parentheses) we can start the previously described process from leaves C_9 and C_{10}. As $3 + 3 > 2$, $\sigma(C_7)$ is set to 6, but C_7 is not selected as cluster. Repeating with C_7 and C_8 results in selecting C_4 as cluster and unselecting C_9, C_{10}, and

C_8 $(6 + 2 < 9)$. Continuing with C_3 and C_4 excludes the possibility to select C_1 as a final cluster because $8 + 9 > 8$. Note that no comparison is ever made against C_0. Similarly for the right section of the tree, C_2 is selected as cluster (excluding C_5 and C_6) giving that $3 + 2 < 6$. In this manner, final clusters with maximal stability are C_3, C_4, and C_2.

4 *MDSCAN* Approach

4.1 Encoding an MD trajectory as a Vantage Point Tree

As discussed in Section 3, HDBSCAN must compute the core distance of every element under consideration. By organizing the input data into some well-defined manner, data structures like *kd-trees* (Baskett and Shustek, 1975) are commonly employed to look up for the k nearest neighbors of query elements without having to explore the whole database each time.

A kd-tree is constructed through iterative bisections of the input data along a single coordinate. These cuts are made at points producing a maximum spread in the selected coordinate's distribution. Unfortunately, efficient usage cases of kd-trees are restricted to Euclidean metric spaces of low dimensionality. Such limitation automatically prevents their utilization in the high-dimensional spaces that characterize molecular conformations.

Vantage point trees (vp-trees, Yianilos, 1993) are an alternative to kd-trees conceived to work with general metrics in high-dimensional spaces. Rather than performing cuts among the coordinates values, nodes of the vp-tree split the database into smaller subspaces employing distinctive elements known as vantage points (vp). By convention, near-to-vp instances constitute the left subspace while far points are grouped into the right subspace. The recursive partition of the input database then leads to a binary tree. In a vp-tree, every frame has a "perspective" on the entire T via their distance to all other frames. This notion of "perspective" is a direct consequence of the triangle inequality represented in Equation 3 that holds for every pair of frames $(a, b) \in T$. In Equation 3, $d(a, b)$ denotes the distance between two points a and b, which will always be greater or equal than the absolute value of the difference between distances from p to a and b respectively.

$$d(a, b) \geq |d(p, a) - d(p, b)| \tag{3}$$

Given a metric space (S, d), and some finite subset $(T \in S)$ representing the trajectory of an MD simulation, the goal of a vp-tree encoding is to organize T in a way that the k nearest neighbors of every query frame q, may be located faster than the naive approach of visiting all frames in T for each query. Suppose that for some frame $p \in T$, the median (μ) of the p-versus-all distances is determined. Then T can be split into two subspaces; the left subspace (or inside sphere S_{pl}), containing frames closer than μ to p, and the right subspace (or outside sphere S_{pr}), containing frames at μ or larger distance values from p (see Figure 3). S_{pl} and S_{pr} will have roughly the same size if there are relatively few frames lying exactly at μ.

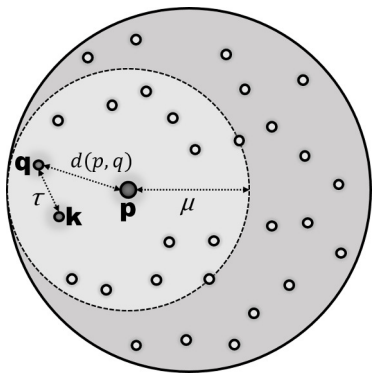

Fig. 3. Partition of a database via the vantage point p. Elements closer than μ to p conforms the left subspace (inside sphere in light gray) while the rest conforms the right subspace (outside sphere in dark gray). q and k denote a query point and its k^{th} nearest neighbor respectively. τ and $d(p, q)$ represent the distances from q to k and to p respectively.

Now suppose that for a query frame $q \in S_{pl}$ the k^{th} nearest neighbors is solicited. By only visiting frames inside S_{pl}, it is possible to define a variable τ storing the distance from q to the k^{th} neighbor found so far. The relevance of vp-trees comes from the fact that if $d(p, q) \geq \mu + \tau$, then the S_{pr} subspace can be safely removed from consideration as searching their elements would not lead to a $\tau \leq d(p, q)$. Similarly, if $q \in S_{pr}$ and $d(p, q) \leq \mu - \tau$, searching the S_{pl} subspace is unnecessary. In both cases, a single point's "perspective" sufficed to prune significantly the search. However, if $\mu - \tau < d(p, q) < \mu + \tau$ no such reduction is possible and the whole T must be explored. Details on the mathematical validity of described notions are available in the fundamental publication of Yianilos on vp-trees (Yianilos, 1993).

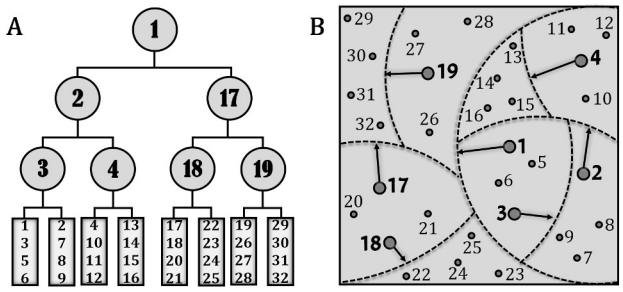

Fig. 4. Two representations of the binary splitting of a 32-element trajectory into a set of 8 sub-trajectories or buckets. A-) Binary vp-tree. Circles represent vantage points and rectangles the sub-trajectories. B-) Spherical cuts applied to the trajectory. Each vantage point has an arrow whose size equals the corresponding μ value.

In this work, we implemented a variation of vp-trees that join leaves into buckets of frames or sub-trajectories. Thus we benefited from both; the efficient pruning offered by the vp-tree and the fast optimal RMSD computation provided by the MDTraj package. In Figure 4A, a trajectory of 32 frames is partitioned as a bucket-type vp-tree (vp^b-tree). The same partitioning is represented in Figure 4B, but using a spherical view of the cuts. Note that, in both cases, the binary splitting of the input trajectory propagates until the formation of buckets.

4.2 Dual-Heap construction of a quasi-Minimum Spanning Tree

HDBSCAN is a general algorithm that can be useful in multiple fields. We will focus on how simple observations lead to substantial memory

savings of optimal RMSD-based implementations for processing long MD trajectories. Even when other alternatives are possible, we believe that the optimal RMSD is a well-studied popular metric whose performance and limitations are thoroughly described and accepted by the community.

In Section 3, $\kappa(i)$ was defined as the distance from i to its k^{th} nearest neighbor. Consequently, the *k-neighborhood* of i (denoted by $\eta(i)$) can be defined as the set of nodes j for which $d(i,j) \leq \kappa(i)$. As a derivation of Equation 1 it can be stated that: *Any node j belonging to $\eta(i)$ and having a core distance $\kappa(j) \leq \kappa(i)$, leads to a minimum value of $d_{mr}(i,j) = \kappa(i)$* (Equation 4).

$$d_{mr}(i,j) = \{\kappa(i) \mid \forall j \in \eta(i) : \kappa(j) \leq \kappa(i)\} \tag{4}$$

This means that if we would like to construct an MST from T nodes, joining i it to any node $j \in \eta(i)$ with $\kappa(j) \leq \kappa(i)$ would create a minimum weighted edge of that MST. In the hypothetical case in which we could connect all nodes of T as a tree following the special case in Equation 4, we would end up with an MST from which the final steps of HDBSCAN may continue (see Section 3).

Although possible, the described scenario is unlikely to occur for a real MD trajectory. The principal assumption of *MDSCAN* is that it happens for most nodes, generating not an MST but a minimum spanning forest (MSF); a collection of disconnected minimum spanning trees of T nodes. The number of MST inside the MSF equals the number of nodes that could not find another node j satisfying Equation 4. For those disconnected i, it is still possible to find a node j in one of the $MST \in MSF$ whose $d_{mr}(i,j)$ though not minimal, would be small enough. Joining all disconnected nodes in the MSF without creating cycles gives a quasi-MST (instead of an exact one). This quasi-MST could be used later to perform subsequent steps of the original formulation of HDBSCAN. The capital importance of this workflow to get a quasi-MST is that no similarity matrix must be stored in RAM.

Nodes i more likely to find a neighbor j satisfying Equation 4 are those with higher values of $\kappa(i)$. As *MDSCAN* continuously checks for Equation 4 to hold, we used a heap data structure that retrieves the node with maximum $\kappa(i)$ found so far in logarithmic time. *MDSCAN* uses two heaps; the first one (main heap) will contain some next-to-analyze nodes, while an auxiliary heap involves already analyzed nodes failing Equation 4 that will be re-processed after exhaustion of the main heap.

The algorithm starts by randomly choosing a not-analyzed node i from the trajectory. This will occur whenever the main heap is empty, so choosing its first element (highest core distance found so far) is not possible. To retrieve the $\eta(i)$ and $\kappa(i)$ of every node, *MDSCAN* queries the vp^b-tree data structure described in Section 4.1. Then, the software searches nodes $j \in \eta(i)$ having $\kappa(j) \leq \kappa(i)$. If such a node is found, a directed edge from i to j is created, and its weight set as $d_{mr}(i,j) = \kappa(i)$.

During this process, all inspected j for which $\kappa(j) \geq \kappa(i)$ are transferred to the main heap as a tuple containing $\kappa(j)$, j index, and $\eta(j)$. If the opposite situation happens, i.e. a node j whose $\kappa(j) \leq \kappa(i)$ is not found in $\eta(i)$, then a tuple containing $\kappa(i)$ and i index is passed to an auxiliary heap for future processing. The previous process goes on until consideration of all nodes. At the end of this stage, an MSF is achieved. The deviation from an exact MST will thus arise from the connection of nodes in the auxiliary heap. The less populated this heap is, the smaller the deviation.

To join the remaining nodes at the auxiliary heap, *MDSCAN* runs the following steps for each one. First, the $RMSD_{ix}$ vector, containing distances from i to all other nodes in T is calculated. Then, the $d_{mr}(i,j)$ is calculated for all nodes j that are not in the same tree that i. The smallest value of these d_{mr} is taken as the weight of a directed connection from i to j. Once the quasi-MST is constructed, *MDSCAN* continues with the building of a cluster hierarchy and the extraction of the most stable clusters exactly as it was described in Section 3 (see Figure 1).

5 *MDSCAN* Benchmark

5.1 Time and memory consumption

Several density-based clustering options were compared against *MDSCAN* in terms of run time and memory consumption: (i) the *HDBSCAN**'s generic options (using the RMSD or Euclidean metrics), (ii) the *HDBSCAN**'s Prim option (using Euclidean metric), (iii) the DBSCAN implementation included in the AMBER package *cpptraj* (using RMSD metric), and (iv) the DBSCAN algorithm available in the scikit-learn Python library (using RMSD metric). The Prim and generic labels of *HDBSCAN** refer to the approach followed for constructing the quasi-MST (see *Section S3: Approaches for computing the quasi-MST in the* HDBSCAN* *software* of Supporting Information).

Although HDBSCAN can be recognized as a replacement for DBSCAN, we included the latter in our benchmark as it is still a popular choice for clustering MD (possibly because of the absence of accessible implementations of HDBSCAN).

For benchmark purposes, we used $k = m = 5$ for all HDBSCAN variants, and an equivalent value of ϵ for DBSCAN alternatives (see Supplementary Information *S1: Search of equivalent parameters between DBSCAN and HDBSCAN**. Both m and k parameters can impact the performance and results of the algorithm. Choosing m too high outputs few clusters, as at every cut of the spanning tree, the number of lost elements will be less than m, and thus, no novel cluster is produced (the parent is considered a just-loosing-elements cluster and does not split). This marginally accelerates the algorithm because the hierarchy would contain fewer clusters to process. The most conservative and natural choice for m in the MD context is 2, as this is the minimum number of elements a cluster can have. One-element clusters or singletons are not considered real clusters because they are not similar to any other element.

On the other hand, the value of k is an estimate of each element's local density (used to calculate the mutual reachability distance). Taking too high values of k would lead to the homogenization of density estimates, and consequently, fewer points in the dataset will be marked as clusters. Increasing k could potentially affect the performance of the data structure selected to retrieve the k^{th} nearest neighbor (as more elements must be inspected from the total.) For benchmark purposes, we used $k = m = 5$ for all HDBSCAN variants, and an equivalent value of ϵ for DBSCAN alternatives (see Supplementary Information *S1: Search of equivalent parameters between DBSCAN and HDBSCAN**).

From the analyzed variants, *MDSCAN* is the fastest option in all cases except for the smallest 6 kF trajectory, where the effort of constructing a vp^b-tree is significant compared to the time taken by the computation of pairwise similarities. *MDSCAN*'s time efficiency comes mainly from the accelerated RMSD computations offered by the MDTraj suite and from our vp^b-tree encoding (discussed in Section 4.1). Indeed, results showed that while the quasi-MST's weight computed by *MDSCAN* with and without using vp^b-tree is equivalent, the run time decreases up to a half when the vp-tree encoding is exploited (see *Section S2: Impact of the vp-tree encoding on MDSCAN performance* of Supporting Information).

While the generic implementations of *HDBSCAN** (RMSD and Euclidean-based) have run times comparable to *MDSCAN*, their high RAM consumption only permitted them to process the two smallest trajectories. *HDBSCAN**'s generic implementations are fast primarily because they use a single-linkage approach to get a tree and do not provide an exact MST at all .

The *HDBSCAN**'s Prim-Euclidean alternative could analyze all trajectories (except the 1 MF job that was stopped after running for 72h),

Table 1. Run time and memory consumption of analyzed software on different trajectories.[1]

			MDSCAN [RMSD]		HDBSCAN* [generic-RMSD]		HDBSCAN* [generic-Euclidean]		HDBSCAN* [Prim-Euclidean]		DBSCAN [scikit-RMSD]		DBSCAN [cpptraj-RMSD]	
Traj. Name	Traj. Size (GB)	# Atoms (selection)[a]	Run time (hh:mm:ss)	RAM peak (GB)	Run time (hh:mm:ss)	RAM peak (GB)	Run time (hh:mm:ss)	RAM peak (GB)	Run time (hh:mm:ss)	RAM peak (GB)	Run time (hh:mm:ss)	RAM peak (GB)	Run time (hh:mm:ss)	RAM peak (GB)
6 kF	0.02	217 (all)	0:00:06	0.18	0:00:04	1.49	0:00:02	1.29	0:00:30	0.15	0:00:03	0.78	0:00:10	0.09
30 kF	0.02	64 (CA)	0:00:19	0.21	0:00:53	35.90	0:00:25	29.01	0:02:14	0.18	0:00:41	17.58	0:01:39	1.78
50 kF	0.05	78 (no H)	0:01:37	0.26	—	**83.82**	—	**74.60**	0:04:28	0.26	0:02:03	46.73	0:05:46	4.71
100 kF	0.75	660 (back)	0:36:42	1.78	—	**335.28**	—	**299.49**	2:31:14	2.58	—	**150.51**	2:10:39	19.38
250 kF	0.47	160 (back)	0:37:46	1.20	—	**2103.87**	—	**1863.54**	5:52:37	1.64	—	**931.32**	—	**125.50**
500 kF	1.25	217 (all)	6:42:18	2.83	—	**8381.90**	—	**7453.01**	21:00:04	4.41	—	**3725.29**	—	**499.99**
1 MF	3.47	304 (back)	21:01:06	7.67	—	**33534.32**	—	**29809.12**	**> 72:00:00**	**14.23**	—	**14901.16**	—	**2048**

[1] *Bold entries denote either a memory crash (jobs taking over 64 GB) or a time crash (jobs taking more than 72h). For each memory-crashed job, an estimation of its memory peak is provided.* [a] *all: all atoms, CA: alpha carbon atoms, no H: non-hydrogen atoms, back: backbone atoms.*

but taking up to nine more times than *MDSCAN* (see Traj. 250 kF in Table 1). This option neither constructs an exact MST from the input data, though a less simplistic approach than a single-linkage is followed to construct a tree (see *Section S3: Approaches for computing the quasi-MST in the* HDBSCAN* *software* of Supporting Information).

DBSCAN alternative available in the scikit-learn Python library has a run time performance that compares well to that of the *HDBSCAN** 's RMSD generic implementation, but could only process the three smallest trajectories. The *cpptraj*'s implementation of DBSCAN is twice slower than the former and, though less memory-hunger, it could not either process trajectories bigger than 100 kF due to its excessive RAM consumption.

Regarding the RAM management, the only efficient alternatives are *MDSCAN* and *HDBSCAN**'s Prim-Euclidean, which do not employ square pairwise similarity matrices to derive the required tree. They both have a similar consumption for the smallest trajectories (from 6 to 50 kF, Table 1), with *MDSCAN* displaying the best behavior for the longest cases (from 100 kF to 1 MF, Table 1). With *MDSCAN*, the RAM is consumed mainly by the MD trajectory file. One copy of this object is needed to instantiate the vp^b-tree data structure and to make the similarity recalculations to complete the final quasi-MST. Another copy is created when producing the vantage tree's buckets. As it is shown in *Section S2: Impact of the vp-tree encoding on* MDSCAN *performance* of Supporting Information, using *MDSCAN* without a vp^b-tree encoding is more memory-friendly (as the second copy never gets created). However, we are persuaded that this small memory trade-off is justified given the speed up reached with the vantage point-based alternative.

The generic-RMSD and generic-Euclidean options of *HDBSCAN** carry the highest memory consumption because, besides the input, these implementations produce another four double-precision float matrices living simultaneously on memory. The estimated RAM consumption of these five objects (Equation 5) is reported in Table 1 for those trajectories that produced a memory crash.

$$V_{RAM} = \frac{(M * m_1 + 4m_2) * N}{2^{30}} \tag{5}$$

In Equation 5, m_1 is the size of the float data type employed in the input matrix ($m_1 = 4$), m_2 is the size of the float data type employed internally by *HDBSCAN** ($m_2 = 8$), N denotes the number of conformations in the trajectory, and M represents the number of columns in the input matrix ($M = N$ for the generic-RMSD, while $M = 3 * number\_of\_atoms$ in the generic-Euclidean variant).

The memory consumption of DBSCAN alternatives is not as huge as in the generic implementations of *HDBSCAN**, but it is still considerable. The *cpptraj* choice consumes the equivalent of an upper triangle in a square matrix of single-precision floats ($m = 4$), while the scikit-learn option depletes the equivalent of four such square matrices (8X more RAM than *cpptraj*).

5.2 Approximations and uncertainties in the Minimum Spanning Tree computation

In the HDBSCAN's formalism, an MST in the d_{mr} space is required (see Section 3, Figure 1). It must be stressed that a graph may contain more than one MST, so two exact implementations of HDBSCAN that used distinct algorithms to retrieve the MST may produce different results in terms of cluster composition. Producing the exact MST is a time-consuming task, usually approached from a heuristic perspective to speed up the algorithm run time. For instance, *HDBSCAN** generic variants do not produce an exact MST, as they use a simplistic single-linkage union of elements to quickly create a tree and then proceed with the subsequent steps of HDBSCAN. Though a more robust approach is followed in the Prim-Euclidean case of *HDBSCAN**, no MST gets ever created with this approach either (see *Section S3: Approaches for computing the quasi-MST in the* HDBSCAN* *software* of Supporting Information).

MDSCAN does not attempt to construct the exact MST, but it provides a quasi-MST with minor deviation from the exact one, as it is presented in Table 2. The exact MST in the d_{mr} metric for trajectories 6, 30, 50, and 100 kF was calculated using a Prim-based *in-house* script (this analysis was not conducted for the biggest trajectories because the available RAM was insufficient). The exact MST's weight was then compared to that of the quasi-MST computed with *MDSCAN*, and the deviation never exceeds 0.8% in any trajectory. For comparison, the exact MST obtained with the Boruvka algorithm in *HDBSCAN** (not presented in the benchmark because of the very long run time of this option) deviates up to 5.8% regarding the Prim-Euclidean approximate tree provided by this software.

Table 2. Deviations from the exact MST in MDSCAN and HDBSCAN*

	MDSCAN (RMSD)			HDBSCAN (Euclidean-boruvka)		
Traj. Name	MST	quasi-MST	Deviation (%)	MST	quasi-MST	Deviation (%)
6 kF	1553.8121	1565.6521	0.76	22505.5140	23812.1929	5.81
30 kF	5247.5244	5286.8418	0.75	44017.5060	46044.3410	4.60
50 kF	2699.6461	2713.9600	0.53	22092.1187	23103.7774	4.58
100 kF	5477.1217	5516.5962	0.72	132284.1054	136900.7708	3.49

5.3 Comparing clusters composition

As stated in Section 5.2, the HDBSCAN clustering alternatives analyzed in this work are expected to produce distinct partitions for each trajectory, mainly because the algorithm for the MST construction drastically varies among them. However, it is worth assessing the impact of this methodological divergence on the outcome clusters produced by each variant. The Adjusted Rand Index (ARI) serves this purpose by comparing two partitions and ranking their similarity from non-related (unbound negative values) to maximal correspondence (positive values bound to 1.00).

The upper triangle of each matrix in Table 3 shows the global ARI between the *MDSCAN* (A), the generic RMSD-based *HDBSCAN** (B),

the generic Euclidean-based *HDBSCAN** (C), and the Prim Euclidean-based *HDBSCAN** (D) implementations. As envisioned, the global similarity of clusterings is far from 1.00 in every case. However, there is an appreciable resemblance in clusterings produced by generic implementations of *HDBSCAN** using RMSD or Euclidean metric in the 6 and 30 kF trajectories. In the particular case of the 30 kF trajectory, all clusterings coming from *HDBSCAN** are correlated but not analogous to *MDSCAN* outcomes. For trajectories bigger than 30 kF, only *MDSCAN* and Euclidean-based *HDBSCAN** produced results, and they were also divergent.

Table 3. Adjusted Rand Index (ARI) of clustering outputs obtained with different HDBSCAN implementations for each analyzed trajectory. The upper triangle of each matrix corresponds to the global ARI, while in the lower triangle only the ARI of those clusters whose population is higher than 1% of the trajectory size is depicted.[a]

	6 kF				**30 kF**				**50 kF**			
	A	*B*	*C*	*D*	*A*	*B*	*C*	*D*	*A*	*B*	*C*	*D*
A	**1.00**	0.34	0.35	0.34	**1.00**	0.24	0.44	0.48	**1.00**	—	—	0.00
B	0.74	**1.00**	0.66	0.32	0.34	**1.00**	0.61	0.64	—	**1.00**	—	—
C	0.74	1.00	**1.00**	0.38	0.46	0.49	**1.00**	0.76	—	—	**1.00**	—
D	0.74	0.99	0.99	**1.00**	0.50	0.55	0.76	**1.00**	0.00	—	—	**1.00**
	100 kF				**250 kF**				**500 kF**			
	A	*B*	*C*	*D*	*A*	*B*	*C*	*D*	*A*	*B*	*C*	*D*
A	**1.00**	—	—	-0.02	**1.00**	—	—	-3.66	**1.00**	—	—	-27.61
B	—	**1.00**	—	—	—	**1.00**	—	—	—	**1.00**	—	—
C	—	—	**1.00**	—	—	—	**1.00**	—	—	—	**1.00**	—
D	0.80	—	—	**1.00**	-4.8	—	—	**1.00**	1.00	—	—	**1.00**

[a] *Each different HDBSCAN variant is represented by a letter: A-* MDSCAN*, B- the generic RMSD-based* HDBSCAN**, C- the generic Euclidean-based* HDBSCAN**, and D- the Prim Euclidean-based* HDBSCAN*.

When clustering MD simulations, users are often more interested in the representative bigger clusters found in trajectories. Although there is no universal or well-established size limit for what a 'representative cluster' can be, we selected as representative those clusters whose sizes are greater or equal than the 1% of the trajectory size. This choice seems reasonable and even conservative, considering that a common practice is to select only the first few biggest N clusters for analyses. In the lower-triangle of each matrix in Table 3, it is shown the pairwise ARI of previous clusterings when considering these bigger clusters.

The ARI between *MDSCAN* and the other alternatives gets doubled in 6 kF. For this trajectory, the most populated clusters reported by all software are indeed similar (clusterings from all variants of *HDBSCAN** are equivalent). This fortuitous agreement is trajectory-dependent, as appreciated in the 30 kF case for which no such prominent improvement of the clustering analogy is attained.

While the ARI can globally inform on partitions similarity, it yields no clues on the equivalence of individual clusters fetched by every software. In *Section S4: Equivalence of representative clusters* of Supporting Information, we designed a more detailed analysis on the correlation of individual clusters across *MDSCAN* and *HDBSCAN** implementations for trajectories 6 and 30 kF (the only trajectories all software could analyze). As a general trend, clusters from all implementations are interconnected. *MDSCAN* often produces smaller groups of frames with the advantage of them having a higher collective similarity (shorter average diameter). The smaller size of some clusters produced by *MDSCAN* (together with the fact that they are tighter than those seized with Euclidean-based options of *HDBSCAN**), is a direct consequence of the pairwise superposition followed when using the RMSD metric. Having tight clusters is the desired behavior when clustering MD as more related frames get included in the same group, facilitating the visual analysis or quantitative averages calculated from clustered structures.

6 Conclusions

HDBSCAN is a solid, popular hierarchical density-based clustering alternative with a broad range of applications. Though already tested for MD simulations, current implementations are oriented to programming-skilled users. Besides, they are impractical to analyze long trajectories because of their quadratic memory complexity (if using RMSD as similarity metric) or due to slow run times (when the Euclidean metric is preferred).

Here we presented *MDSCAN*, an efficient approach to HDBSCAN designed to treat long MD simulations. *MDSCAN* employs the RMSD metric, but it does not rely on a pairwise similarity matrix as most related software. Instead, it uses a double heap method that constructs a quasi-Minimum Spanning Tree, a neuralgic step in HDBSCAN's workflow. *MDSCAN*'s run time efficiency is accomplished by encoding the MD trajectory as a vantage point tree data structure that avoids futile recalculations of the similarity information.

MDSCAN is a publicly available and intuitive command-line interface whose results can be visualized through VMD scripts.

Funding

This work was supported by the Cuban Oficina de Gestión de Fondos y Proyectos Internacionales, CITMA [PN223LH010-02 to R.G.A and L.M.C], the Eiffel Scholarship Program of Excellence of Campus France [P104786Z to R.G.A]; the Project Hubert Curien-Carlos J. Finlay [41814TM to R.G.A, F.L, and L.M.C]; and the Fondo Nacional de Desarrollo Científico y Tecnológico [CONICYT/FONDECYT/INACH/POSTDOCTORADO/No. 3170107 to E.W.H.R].

Data availability

The source code and documentation of *MDSCAN* are free and publicly available on GitHub (https://github.com/LQCT/MDScan.git) and as a PyPI package (https://pypi.org/project/mdscan/).

References

Baskett, F. and Shustek, L. J. (1975). An Algorithm for Finding Nearest Neighbors. *IEEE Transactions on Computers*, **C-24**(10), 1000–1006.

Campello, R. J. G. B. *et al.* (2013). Density-Based Clustering Based on Hierarchical Density Estimates. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7819 LNAI, pages 160–172.

Campello, R. J. G. B. *et al.* (2020). Density-based clustering. *WIREs Data Mining and Knowledge Discovery*, **10**(2).

González-Alemán, R. *et al.* (2020a). BitClust: Fast Geometrical Clustering of Long Molecular Dynamics Simulations. *Journal of Chemical Information and Modeling*, **60**(2), 444–448.

González-Alemán, R. *et al.* (2020b). Quality Threshold Clustering of Molecular Dynamics: A Word of Caution. *Journal of Chemical Information and Modeling*, **60**(2), 467–472.

González-Alemán, R. *et al.* (2021). BitQT: a graph-based approach to the quality threshold clustering of molecular dynamics. *Bioinformatics*, **38**(1), 73–79.

Hinneburg, A. and Keim, D. A. (2003). A General Approach to Clustering in Large Databases with Noise. *Knowledge and Information Systems*, **5**(4), 387–415.

McGibbon, R. T. *et al.* (2015). MDTraj: A Modern Open Library for the Analysis of Molecular Dynamics Trajectories. *Biophysical Journal*, **109**(8), 1528–1532.

McInnes, L. and Healy, J. (2017). Accelerated Hierarchical Density Based Clustering. In *IEEE International Conference on Data Mining Workshops, ICDMW*, volume 2017-November, pages 33–42. IEEE.

Melvin, R. L. *et al.* (2016). Uncovering Large-Scale Conformational Change in Molecular Dynamics without Prior Knowledge. *Journal of Chemical Theory and Computation*, **12**(12), 6130–6146.

Melvin, R. L. *et al.* (2018). Visualizing correlated motion with HDBSCAN clustering. *Protein Science*, **27**(1), 62–75.

Pedregosa, F. *et al.* (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, **12**, 2825–2830.

Pei, T. *et al.* (2009). DECODE: A new method for discovering clusters of different densities in spatial data. *Data Mining and Knowledge Discovery*, **18**(3), 337–369.

Peng, J. H. *et al.* (2018). Clustering algorithms to analyze molecular dynamics simulation trajectories for complex chemical and biological systems. *Chinese Journal of Chemical Physics*, **31**(4), 404–420.

Platero-Rochart, D. *et al.* (2022). RCDPeaks: Memory-efficient density peaks clustering of long molecular dynamics. *Bioinformatics*, **38**(7), 1863–1869.

Roe, D. R. and Cheatham, T. E. (2013). PTRAJ and CPPTRAJ: Software for processing and analysis of molecular dynamics trajectory data. *Journal of Chemical Theory and Computation*, **9**(7), 3084–3095.

Sargsyan, K. *et al.* (2017). How Molecular Size Impacts RMSD Applications in Molecular Dynamics Simulations. *Journal of Chemical Theory and Computation*, **13**(4), 1518–1524.

Schubert, E. *et al.* (2017). DBSCAN revisited, revisited: Why and how you should (still) use DBSCAN. *ACM Transactions on Database Systems*, **42**(3), 1–21.

Shao, J. *et al.* (2007). Clustering molecular dynamics trajectories: 1. Characterizing the performance of different clustering algorithms. *Journal of Chemical Theory and Computation*, **3**(6), 2312–2334.

Shea, J. E. and Levine, Z. A. (2016). Studying the early stages of protein aggregation using replica exchange molecular dynamics simulations. In *Methods in Molecular Biology*, volume 1345, pages 225–250.

Stuetzle, W. and Nugent, R. (2010). A generalized single linkage method for estimating the cluster tree of a density. *Journal of Computational and Graphical Statistics*, **19**(2), 397–418.

Sun, H. *et al.* (2010). gSkeletonClu: Density-based network clustering via structure-connected tree division or agglomeration. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 481–490. IEEE.

Yianilos, P. N. (1993). Data structures and algorithms for nearest neighbor search in general metric spaces. *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321.