RCDPeaks: memory-efficient density peaks clustering of long molecular dynamics

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

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

OXFORD

Structural Bionformatics

RCDPeaks: Memory-Efficient Density Peaks Clustering of Long Molecular Dynamics

Daniel Platero-Rochart [1,*], Roy González-Alemán [1, 2,*], Erix W. Hernández-Rodríguez [3], Fabrice Leclerc [2], Julio Caballero [4] 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, Escuela de Química y Farmacia, Facultad de Medicina, Universidad Católica del Maule, 3460000 Talca, 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: Density Peaks is a widely spread clustering algorithm that has been previously applied to Molecular Dynamics simulations. Its conception of cluster centers as elements displaying both a high density of neighbors and a large distance to other elements of high density, particularly fits the nature of a geometrical converged Molecular Dynamics simulation. Despite its theoretical convenience, implementations of Density Peaks carry a quadratic memory complexity that only permits the analysis of relatively short trajectories.
Results: Here, we describe *DP+*, an exact novel implementation of Density Peaks that drastically reduces the RAM consumption in comparison to the scarcely available alternatives designed for Molecular Dynamics. Based on *DP+*, we developed *RCDPeaks*, a refined variant of the original Density Peaks algorithm. Through the use of *DP+*, *RCDPeaks* was able to cluster a one-million frames trajectory using less than 4.5 GB of RAM, a task that would have taken more than 2 TB and about 3X more time with the fastest and less memory-hunger alternative currently available. Other key features of *RCDPeaks* include the automatic selection of parameters, the merging of very similar center candidates, and the geometrical refining of returned clusters. The source code and documentation of *RCDPeaks* are free and publicly available on GitHub (https://github.com/LQCT/RCDPeaks.git).
Contact: roy_gonzalez@fq.uh.cu, daniel.platero@fq.uh.cu
Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Geometrical clustering of Molecular Dynamics (MD) simulations is a spread task in the Bioinformatics field. Formally conceptualized as an unsupervised machine learning technique (Sammut and Webb, 2010), clustering aims to classify elements according to their similarity into groups named clusters. Though a rich palette of these algorithms has been proposed and continuously optimized to deal with the growing size of MD trajectories (Shao *et al.*, 2007; Peng *et al.*, 2018), the popular Density Peaks alternative (Rodriguez and Laio, 2014) stands out for its simple yet powerful definitions.

In Density Peaks (DP), clusters centers are spotted as those elements displaying both a high density of neighbors and a relatively large distance to other elements of high density. As it has been already pointed out (Sylvain *et al.*, 2020), the previous statement exceptionally fits the nature of a converged MD simulation, where relevant biological states would lie in denser regions separated by lower-density zones of transitional basins.

Despite its theoretical convenience, DP has some practical limitations that have given rise to diverse enhancement proposals (see Seyedi *et al.*, 2019; Flores and Garza, 2020 for a review) typically addressed to one of the following aspects: i- the robust estimation of each element's density (Wang and Xu, 2017; Du *et al.*, 2016), ii- the selection of an adequate distance metric (Du *et al.*, 2017), iii- reducing the computational complexity of computing the local density of each component and their distance to neighbors of higher density (Majdara and Nooshabadi, 2020), iv- the automatic determination of clusters centers (Liang and Chen, 2016; Flores and Garza, 2020), and v- optimizing the process of assigning elements to clusters (Wang *et al.*, 2019; Seyedi *et al.*, 2019).

There are few implementations of DP specifically designed to treat MD trajectories. The *cpptraj* module of the AMBER suite (Roe and Cheatham, 2013) is equipped with an exact variant while a recent contribution has proposed *CLoNe* (Sylvain *et al.*, 2020), a robust improvement of the original algorithm. Even though these two options' quadratic time complexity is not critical for processing a relatively long MD trajectory, their also quadratic memory complexity renders this endeavor impractical.

Here we propose *DP+*, a methodology to derive the exact DP partitioning of elements but without constructing a square similarity matrix. Instead, a double-heap approach is used to produce an oriented tree where every node (trajectory frame) is connected to its nearest neighbor of higher density by a weighted edge (RMSD distance).

Built upon *DP+*, we designed *RCDPeaks*, a refined variant of the original DP. Employing *DP+*, *RCDPeaks* processed a one-million frames trajectory using less than 4.5 GB of RAM, a task that would have taken more than 2 TB (and about 3X more time) or 7 TB (and about 30X more time) with *cpptraj* or *CLoNe*, respectively.

After computing each element's density and the optimal RMSD distance to its nearest neighbor of higher density through *DP+*, *RCDPeaks* can automatically set the necessary cutoffs and detect potential cluster centers. These centers are then selectively merged to guarantee a high distance between them and to avoid the unnecessary splitting of clusters. Then, the usual DP clustering of elements occurs, and it is refined to smaller groups of frames revealing a higher degree of collective similarity.

2 Computational Details

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

The computational performance of the AMBER *cpptraj* exact implementation of Density Peaks (Roe and Cheatham, 2013) and *CLoNe* (Sylvain *et al.*, 2020) were compared against *RCDPeaks*. The benchmark was conducted on a set of publicly available trajectories that are referred by their size 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 based on PDB 2RJY (Melvin *et al.*, 2016), iii- 50 kF, a 50000 frames MD of serotype 18C of *Streptococcus pneumoniae*, iv- 100 kF, a 100000 frames MD of Cyclophilin A based on PDB 2N0T, 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) (Álvarez-Ginarte et al., unpublished work), vi- 500 kF, a 500000 frames MD toy trajectory constructed from randomly selected conformations of 6 kF, and vii- 1 MF, a one-million frames MD of ubiquitin based on PDB 1UBQ . The details of MD simulations are available in the Supporting Information (S1: Details of the Molecular Dynamics Simulations). All trajectory and topology files used in this work can be found 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 unavailable at this moment.

RCDPeaks and *cpptraj*, used the same distance cutoff value (d_c) for every trajectory; 2.5 Å for 6 kF and 500 kF, 4 Å for 30 kF, 1 Å for 50 kF, 100 kF and 1M, and 2 Å for 250 kF. These values where set after a trial/error procedure aided by visual inspection of the number of possible centers in the decision graph. The only input parameter of CLoNe is a user-defined percentage p_{dc} of all pairwise similarity distances. This parameter was set to 0.4 for 6 kF (corresponding to $d_c = 2.6$) and to 4.4 for 30 kF (corresponding to $d_c = 4.0$). The other trajectories could not be analyzed with CLoNe due to excessive memory consumption.

All calculations were performed on an AMD Ryzen 5 Hexa-core Workstation with a processor speed of 3.6 GHz and 64GB 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 Density Peaks formalism

In DP formalism, cluster centers are surrounded by neighbors of lower local density, and they are distant from any point with high local density. This simple statement rules the algorithm, which can be described as follows when processing an MD trajectory.

Two magnitudes are computed for each frame i after setting a distance cutoff (d_c); its local density (ρ_i in equation 1) and its minimum distance to a neighbor of higher local density, (δ_i in equation 2). In equation 1 the term $\chi(x) = 1$ if $x < 0$ or zero otherwise so this is equivalent to define ρ_i as the number of i neighbors whose distance from i is under the d_c cutoff.

$$\rho_i = \sum_j \chi(d_{ij} - d_c) \tag{1}$$

In equation 2, an exception is made for the frame of maximum ρ_i, which is conventionally set to $max(d_{ij})$. Note that δ_i is significantly larger than the typical nearest neighbor distance only for those frames that are local or global maxima in ρ. Higher values of δ_i are then a distinctive hallmark of cluster centers. Previous information can be condensed and visually inspected in the *decision graph* of the trajectory; a 2D representation of ρ versus δ in which clusters centers are spotted at higher values of these two magnitudes.

$$\delta_i = min(d_{ij}) \ : \ \rho_j > \rho_i \tag{2}$$

After selecting cluster centers from the decision graph, each remaining frame is assigned to the same cluster as its nearest neighbor of higher ρ. Algorithm 1 contains the pseudocode corresponding to the previous steps.

To account for the notion of noise, DP defines a *boundary region* for each cluster C_i consisting of frames that were previously assigned to C_i but being within a distance d_c from frames belonging to other clusters. The maximum density value of the boundary region is designated as ρ_b and compared to the ρ_i of every frame in C_i. If $\rho_i > \rho_b$ the frame belongs to the core region (robust assignation), otherwise it can be considered in the halo zone (noisy assignation).

The typical workflow used in exact or modified DP variants saves the pairwise similarity of elements (frames in the particular case of an MD) into a square float matrix. This strategy may offer a fast determination of ρ_i and δ_i but inconveniently limits the algorithm's application to problems whose similarity matrix could fit in available RAM. Next, we describe *DP+*, an alternative approach to the exact DP that avoids the construction and storage of such a matrix and hence can be applied to treat much longer trajectories.

Algorithm 1: Density Peaks clustering algorithm

Require: $trajectory, d_c$
► **1. Compute the pairwise similarity matrix**
$rmsd\_matrix =$ ***calc_rmsd_matrix***$(trajectory)$
► **2. Compute ρ values for each node**
$elements = \{1, 2, 3, ..., trajectory.size\}$
$rho\_values = \{\}$
for $i \in elements$ **do**
  $i\_vector = rmsd\_matrix[i]$
  $rho\_values[i] =$ ***count_elements***$(i\_vector < d_c)$
► **3. Compute δ values for each node**
$delta\_values = \{\}$
for $i \in elements$ **do**
  $i\_vector = rmsd\_matrix[i]$
  $i\_rho = rho\_values[i]$
  $i\_sorted =$ ***sort_elements***(i_vector)
  for $j \in i\_sorted$ **do**
    $j\_rho = rho\_values[j]$
    if $j\_rho > i\_rho$ **then**
      $delta\_values[i] = rmsd\_matrix[i][j]$
  if $delta\_values[i] == None$ **then**
    $delta\_values[i] =$ ***get_max_value***$(rmsd\_matrix)$
► **4. Select cluster centers from the Decision Graph**
$decision\_graph =$ ***plot***$(rho\_values, delta\_values)$
$rho\_cut, delta\_cut =$ ***select_cutoffs***$(decision\_graph)$
$centers =$ ***select_centers***$(decision\_graph, rho\_cut, delta\_cut)$
► **5. Assign remaining elements**
$clusters =$ ***assign_elements***$(elements, centers)$

4 *DP+* Implementation

DP+ exploits the graph-theoretical view of an MD trajectory by considering it as a graph T in which all nodes are pairwise connected. In T, nodes represent frames, and their pairwise similarity distance weights undirected edges (Figure 1A). If ρ values are assigned as the weights of T nodes, then the goal of DP can be stated as transforming T into an oriented tree T' that contains only one outgoing edge per node pointing to its nearest neighbor of higher ρ. The weights of edges in T' correspond to δ values in equation 2 (Figure 1B).

For every frame i, *DP+* computes ρ_i from the i-versus-all RMSD vector ($RMSD_{ix}$), by counting the number of elements j whose $RMSD_{ij} < d_c$. As δ_i refers to the distance from i to its nearest neighbor of higher ρ, computing this magnitude requires iterative queries to the sorted $RMSD_{ix}$ vector. However, the complete sorting of $RMSD_{ix}$ is an expensive $O(n * log(n))$ operation. *DP+* makes a faster partial ordering ($O(n)$ time complexity) of $RMSD_{ix}$ at the k^{th} position and then a complete ordering of the much smaller k-neighborhood (denoted as η from now on). The value of k is internally defined as $0.02 * N$ (although users can modify it), where N is the total number of frames in the trajectory. *DP+* relies on the assumption that most frames will find their nearest neighbor of higher ρ inside this sorted η.

Figure 2 illustrates the previous procedure using the $RMSD_{0x}$ vector of a ten-frames trajectory where $d_c = 0.36$ nm and $k = 5$. In Figure 2A, ρ_0 (the number of frames j for which $RMSD_{0j} < d_c$) is set as 7 (bold entries). In 2B, the partial sorting of $RMSD_{0x}$ at $k = 5$ is exemplified. Note that this process returns the first unsorted k elements with lowest values. Figure 2C shows the last ordering stage in which only the first k elements of $RMSD_{0x}$ are completely sorted. This vector corresponds to η_i (see Algorithm 2).

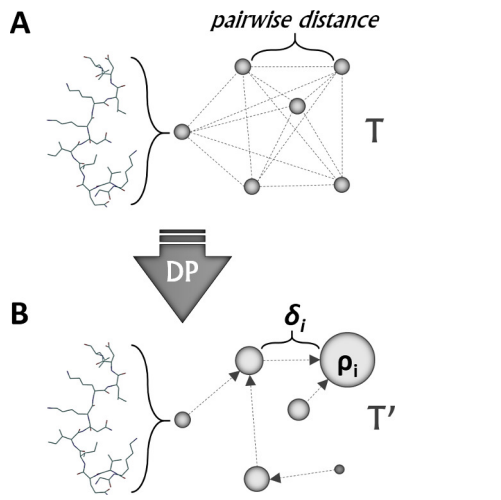

Fig. 1: Graph-theoretical view of an MD trajectory before and after applying DP. A-) Complete graph T in which nodes correspond to frames and undirected edges denote pairwise similarity B-) Oriented tree T' obtained after applying DP to T. Each node (weighted by its ρ value) contain a single outgoing edge pointing to its nearest neighbor of higher density.

Algorithm 2: Get ρ and η for a particular node i

function get_node_info($i, k, d_c, trajectory$)
  $i\_vector =$ ***calc_rmsd_vector***$(i, trajectory)$
  $i\_rho =$ ***count_elements***$(i\_vector < d_c)$
  $i\_partition =$ ***partial_sort_elements***(i_vector, k)
  $i\_eta\_elements =$ ***sort_elements***$(i\_partition[0:k])$
  $i\_eta\_rmsd = i\_vector[i\_eta\_elements]$
  $i\_eta =$ ***join***$(i\_eta\_elements, i\_eta\_rmsd)$
  return (i_rho, i, i_eta)

To avoid the storage of T information as a square matrix, *DP+* gradually constructs T' using data distributed in two separate heaps. The main heap will contain the ρ_i, i, and η_i for a subset of frames (Figure 2D), while an auxiliary heap will store those frames whose nearest neighbor

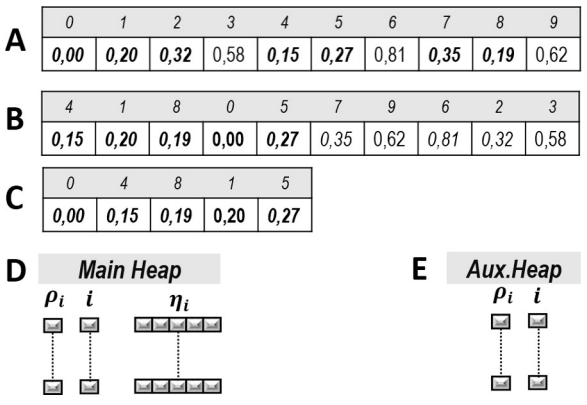

Fig. 2: *DP+* main objects and operations involved in the computation of ρ_i and η_i for a ten-frames trajectory ($d_c = 0.36\ nm$ and $k = 5$). A-) $RMSD_{0x}$ vector. Bold entries correspond to frames closer than d_c from frame 0. B-) $RMSD_{0x}$ partially sorted at $k = 5$. C-) Complete ordering of first k values of $RMSD_{0x}$ (η_0). D-) Main heap. E-) Auxiliary heap.

of higher density could not be found inside their η_i (Figure 2E). The importance of using a heap data structure lies in its ability to quickly retrieve an extreme value (minimum in our case) of the collections it contains. If we introduce several tuples containing ρ_i and η_i, a so-called *"min heap"* can return the minimum weighted frame and its corresponding η_i in logarithmic time. Through the use of heaps, *DP+* speeds up the construction of T', exploiting the observation that frames with lower ρ are more likely to find their nearest neighbor of higher density inside η.

Concretely, after defining a local density cutoff d_c, *DP+* follows the next steps to construct T' (see Algorithm 3): A still not analyzed frame i is chosen from the trajectory. This action will occur whenever the main heap is empty. $RMSD_{ix}$ is then calculated and ρ_i computed counting the number of elements j with $RMSD_{ij} < d_c$. Through the already mentioned sorting strategy, η_i is obtained and *DP+* proceeds to search the first frame $X_j \in \eta_i$ having $\rho_j > \rho_i$. If such a frame is found, a directed edge from i to j is created, and δ_i is set to d_{ij}. During this process, all inspected j for which $\rho_j \leq \rho_i$ are transferred to the main heap as a tuple containing ρ_j, j index and η_j. If the opposite situation happens, i.e., a frame j whose $\rho_j > \rho_i$ is not found in η_i, then a tuple containing ρ_i and i index is passed to a secondary heap for future processing. The previous process goes on until all frames have been considered.

At that point, the frames i that did not found their nearest neighbor inside η_i are already stored in the auxiliary heap. For each one of them, *DP+* recalculates $RMSD_{ix}$ and finds the frame j with $\rho_j > \rho_i$ to set δ_i. In the special case where i has the maximum value of ρ (so it is impossible to find $\rho_j > \rho_i$), δ_i is set to $max(RMSDix)$. Experiments show that the average size of the auxiliary heap is a small percent of N.

Algorithm 3: Compute the Oriented Tree of an MD trajectory

function compute_oriented_tree($k, d_c, trajectory$)
  ► **1. Initialize containers**
  $elements = \{1, 2, 3, ..., trajectory.size\}$
  $main\_heap =$ ***create_heap***$()$
  $auxiliary\_heap =$ ***create_heap***$()$
  $rho\_info = \{\}$
  $delta\_info = \{\}$
  $nearest\_neighbors = \{\}$
  ► **2. Find node i whose neighborhood will be analyzed**
  while $True$ **do**
    if $main\_heap \neq \emptyset$ **then**
      $i, i\_rho, i\_eta =$ ***pop_first_from***$(main\_heap)$
    else if $elements \neq \emptyset$ **then**
      $i =$ ***pop_any_from***$(elements)$
      $i\_rho, i, i\_eta =$ ***get_node_info***$(i, k, d_c, trajectory)$
    else
      break
    ► 3. **Try to find j inside η_i**
    while $True$ **do**
      if $i\_eta \neq \emptyset$ **then**
        $j, rmsd\_ij =$ ***next***(i_eta)
      else
        send$((i\_rho, i), auxiliary\_heap)$
        break
      if $j \in elements$ **then**
        $j\_rho, j, j\_eta =$ ***get_node_info***$(j, k, d_c, trajectory)$
        send$((j\_rho, j, j\_eta), main\_heap)$
        $rho\_info[j] = j\_rho$
        remove_from$(elements, j)$
      else
        $j\_rho = rho\_info[j]$
      if $j\_rho > i\_rho$ **then**
        $nearest\_neighbors[i] = j$
        $delta\_info[i] = rmsd\_ij$
        break
  ► **4. Processing the auxiliary heap**
  while $True$ **do**
    if $auxiliary\_heap \neq \emptyset$ **then**
      $i\_rho, i =$ ***pop_first_from***$(auxiliary\_heap)$
      $i\_vector =$ ***calc_rmsd_vector***$(i, trajectory)$
      $denser\_j =$ ***get_elements***$(rho\_info > i\_rho)$
      $j\_vector = i\_vector[denser\_j]$
      if $j\_vector \neq \emptyset$ **then**
        $j =$ ***get_min_element***(j_vector)
        $delta\_info[i] = i\_vector[j]$
        $nearest\_neighbors[i] = j$
      else
        $delta\_info[i] =$ ***get_max_value***$[i\_vector]$
        $nearest\_neighbors[i] = i$
    else
      break
  return $(delta\_info, rho\_info, edges)$

5 *RCDPeaks* Refinements

As explained in Section 4, *DP+* is an exact implementation of the original DP. *DP+* avoids the quadratic memory complexity by using heap-based data structures. Having equivalent results, both approaches share the same shortcomings, among which are: i- the consideration of very similar center candidates as independent cluster seeds, inducing the unnecessary splitting of final clusters. This occurs because, in the user-selected region of the decision graph, no checking is performed on centers to ensure their pairwise geometrical separation. ii- the impossibility to run an automatic job given that ρ and δ must be manually selected from the decision graph, and iii-the excessive flexibility of core and halo definitions for MD applications (see Figure 4). In this section, we propose *RCDPeaks* (Refined-Core Density Peaks), which is built upon *DP+* and addresses the aforementioned limitations.

5.1 Automatic Detection and Merging of Cluster Centers

In the original DP, users must select the cluster centers from the decision graph before the DP algorithm could assign the remaining frames to each cluster (Figure 3A). This selection introduces a potentially biased, user-dependent step that also prevents automatic runs. Several authors have used statistical mechanisms to bypass this step (see Flores and Garza, 2020 for a review) by detecting clusters centers as ρ, δ or γ outliers (equation 3).

$$\gamma_i = \rho_i * \delta_i \tag{3}$$

The *gap-based* centers selection method proposed by Flores and Garza (Flores and Garza, 2020) proceeds as follows: First, a subset P_1 containing elements whose ρ and δ values are higher than the average is defined (discontinue lines in Figure 3B). P_1 is subsequently sorted in descending order of each γ_i score. The consecutive point distance (equation 4) between all candidates, as well as the average point distance (equation 5) are then computed. In this context, a gap is formally defined as a $d_i \geq \bar{d}_i$. The last gap in P_1 (formed by elements i and $i+1$) is considered a threshold and all elements before i are marked as cluster centers.

$$d_i = abs(\gamma_i - \gamma_{i+1}) \tag{4}$$

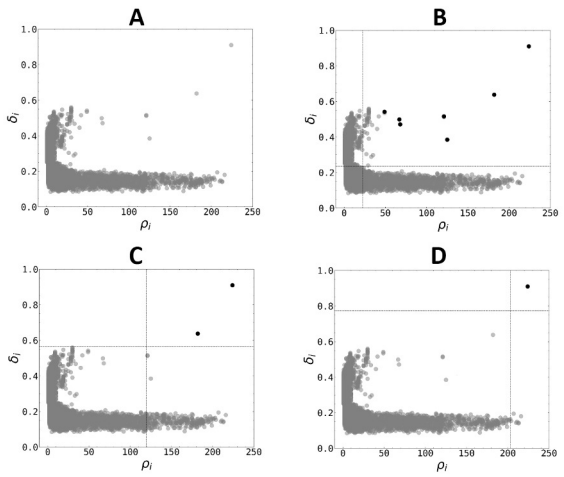

Fig. 3: Iterative gap-based method of Flores and Garza implemented in *RCDPeaks* for the automatic detection of cluster centers. A-) Decision graph. B-D-) Consecutive iterations of the method produce several automatic guesses of cluster centers.

$$\bar{d}_i = \sum_{a \in P_i} \frac{d_a}{|P_i|} \tag{5}$$

The described methodology produced a high number of cluster centers for the trajectories analyzed in this work. Instead of stopping the algorithm after the first loop, *RCDPeaks* makes another iteration on a new subset P_2, containing only elements whose ρ and δ values are higher than the average in P_1 (Figure 3C). This procedure effectively reduces the number of candidate clusters, which are intuitively a subset of the original P_1. Iteration continues until the one-member set P_n is found (Figure 3D). All sets from P_1 to P_n may be considered as valid automatic guesses of cluster centers. Each one of the P_n guesses made by *RCDPeaks* will be further processed in n distinct clustering jobs of the same oriented tree represented by the decision graph in Figure 3A. In the analyzed trajectories, n varies from 2 to 3.

Although *RCDPeaks* implements the Flores and Garza method, users still have the choice to manually set ρ and δ values. Also, as the most time-consuming part of *RCDPeaks* consist of computing those two magnitudes for each frame, the software conveniently saves the decision graph, allowing users to experiment on their own the result of different ρ and δ cutoffs for cases where the automatic guesses do not perform as expected in an inexpensive way.

Centers retrieved by either an automatic or a manual selection may lie within a d_c radius. Those cases correspond to regions with multiple density peaks. The original DP unsuccessfully handles these cases by dividing the region into analogous clusters. *RCDPeaks* avoids this worthless splitting through a merging process of nearby centers. This process iteratively takes the center of highest γ_i from P_i as a reference and removes other centers within a d_c distance from further consideration.

5.2 Clusters Core Refining

MD clusters generated by the original version of DP usually contain structurally unrelated frames. Definitions of the core and halo zones (see Section 3) contribute to some extent to the separation of highly similar elements (core) from more loosely related ones (halo). However, the original cores obtained by the DP clustering may still display a high level of dissimilarity as can be appreciated in Figure 4.

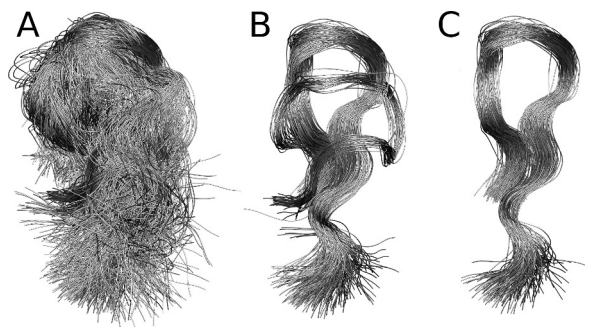

Fig. 4: Second cluster of trajectory 6 kF. A-) The raw cluster obtained by the original DP approach. B-) Cluster core obtained by the original DP approach. C-) refined cluster core obtained by *RCDPeaks*.

Since cluster centers have a preponderant significance in DP, it is reasonable to expect their geometrical resemblance to frames in their respective cores. *RCDPeaks* follows a simple procedure to extract a set of exemplar frames (a *refined core*), evincing a higher degree of collective similarity than what can be obtained from the original definition of core zones in DP. For each cluster C_i, its refined core will consist of those frames within a d_c distance from its cluster center. As it can be appreciated in Figure 4C, this restrained set does exhibit a considerable level of uniformity.

6 Performance Comparison

The run time and RAM consumption of *RCDPeaks*, *cpptraj*, and *CLoNe* when processing different MD trajectories are compared in Table 1. To the best of our knowledge, these three software are the only DP implementations publicly available and specifically designed to deal with MD simulations. While *cpptraj* implements the original algorithm, *CLoNe* was inspired on DP to overcome several of its limitations.

Table 1. Run time and RAM consumption of analyzed DP implementations.[1]

		RCDPeaks		***cpptraj***		***CLoNe***		
Trajectory	**No. of atoms** (selection)	Run time *h:mm:ss*	RAM peak *GB*	Run time *h:mm:ss*	RAM peak *GB*	Run time *h:mm:ss*	RAM peak *GB*	Disk space GB
6 kF	217 (all)	0:00:05	0.14	0:00:10	0.09	0:00:40	2.35	0.21
30 kF	64 (CA)	0:00:42	0.16	0:01:46	1.78	0:23:22	39.72	6.30
50 kF	78 (no H)	0:02:00	0.19	0:05:59	4.71	***0:11:29***	***>64.00***	***15.00***
100 kF	660 (backbone)	0:41:59	0.92	2:10:23	19.38	***NR***	***≫ 74.51***	***> 57.22***
250 kF	160 (backbone)	1:14:04	0.87	***0:00:04***	***>125.50***	***NR***	***≫ 465.66***	***> 359.06***
500 kF	217 (all)	6:47:12	2.03	***0:00:09***	***>499.99***	***NR***	***≫ 1862.65***	***> 1430.51***
1 MF	304 (backbone)	33:21:10	4.16	***0:00:26***	***>2048***	***NR***	***≫ 7452.07***	***> 5723.20***

[1] *Bold entries denote a memory crash (jobs taking more than 64GB of RAM). NR means Not Ran Job.*

As it is shown in Table 1, *CLoNe* has the highest RAM consumption, which only permitted to process the small trajectories 6 kF and 30 kF. This variant also uses substantial disk space resources if the similarity metric is not euclidean (RMSD in our case), as the user must provide a text file with the pairwise similarity information. Although *CLoNe* also has the slowest run time (about 30X slower than *RCDPeaks* for the 30 kF trajectory), this is not a critical aspect when dealing with the short trajectories it can manage.

The *cpptraj* alternative is considerably less RAM consuming than *CLoNe*. The memory peak for each analyzed trajectory roughly corresponds to the storage of a half-precision float square matrix (pairwise RMSD information). For short and medium-sized MD trajectories (see

100 kF in Table 1), *cpptraj* has an affordable memory cost. However, if relatively long trajectories must be processed, the quadratic RAM complexity of *cpptraj* becomes a major limitation. In terms of run time, *cpptraj* is also faster than *CLoNe* but still about 3X slower than *RCDPeaks*. It is worth noting that developers of *cpptraj* have marked their implementation as experimental. This software will produce neither the core nor the boundary regions of the calculated clusters.

The fastest and the most memory-efficient software is *RCDPeaks*. The key factors contributing to the speed up of this variant are the use of MDTraj for computing the optimal RMSD distances and, to a lesser extent, the sorting procedure to get η_i (see Section 4). On the other hand, the RAM consumption of *RCDPeaks* is remarkably low, mainly due to the small size of the *main heap* (see Section 4).

7 Conclusion

In this work, we have proposed *DP+*, an exact implementation of the popular Density Peaks clustering algorithm. The main contribution of *DP+* lies in its ability to reduce the quadratic memory complexity of the original DP. Instead of storing the pairwise similarity of frames into a square matrix, a double-heap approach is employed to construct an oriented tree from the MD trajectory. Besides being faster than other similar MD-oriented software, our approach produces massive savings of RAM resources. Built on top of *DP+*, we conceived *RCDPeaks*, a refinement of the original DP algorithm including convenient features like the ability to automatically produce multiple guesses of cluster centers, the merging of very similar cluster center candidates, and the refinement of retrieved clusters.

Acknowledgements

D.P.R thanks Matteo Dal Peraro and Sylvain Träger for their help on setting up *CLoNe*.

Funding

This work was supported by the Eiffel Scholarship Program of Excellence of Campus France [P744468L 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].

References

Du, M. *et al.* (2016). Study on density peaks clustering based on k-nearest neighbors and principal component analysis. *Knowledge-Based Syst.*, **99**, 135–145.

Du, M. *et al.* (2017). A novel density peaks clustering algorithm for mixed data. *Pattern Recognit. Lett.*, **97**, 46–53.

Flores, K. G. and Garza, S. E. (2020). Density peaks clustering with gap-based automatic center detection. *Knowledge-Based Syst.*, **206**, 106350.

Liang, Z. and Chen, P. (2016). Delta-density based clustering with a divide-and-conquer strategy: 3DC clustering. *Pattern Recognit. Lett.*, **73**, 52–59.

Majdara, A. and Nooshabadi, S. (2020). Accelerated Density-Based Clustering using Bayesian Sequential Partitioning. In *2020 IEEE Int. Symp. Circuits Syst.*, pages 1–5. IEEE.

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

Melvin, R. L. *et al.* (2016). Uncovering Large-Scale Conformational Change in Molecular Dynamics without Prior Knowledge. *J. Chem. Theory Comput.*, **12**(12), 6130–6146.

Peng, J.-h. H. *et al.* (2018). Clustering algorithms to analyze molecular dynamics simulation trajectories for complex chemical and biological systems. *Chinese J. Chem. Phys.*, **31**(4), 404–420.

Rodriguez, A. and Laio, A. (2014). Clustering by fast search and find of density peaks. *Science (80-.).*, **344**(6191), 1492–1496.

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

Sammut, C. and Webb, G. I. (2010). Encyclopedia of Machine Learning.

Seyedi, S. A. *et al.* (2019). Dynamic graph-based label propagation for density peaks clustering. *Expert Syst. Appl.*, **115**, 314–328.

Shao, J. *et al.* (2007). Clustering molecular dynamics trajectories: 1. Characterizing the performance of different clustering algorithms. *J. Chem. Theory Comput.*, **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 Mol. Biol.*, volume 1345, pages 225–250.

Sylvain, T. *et al.* (2020). CLoNe: Automated clustering based on local density neighborhoods for application to biomolecular structural ensembles. *Bioinformatics*.

Wang, G. *et al.* (2019). Modified FDP cluster algorithm and its application in protein conformation clustering analysis. *Digit. Signal Process. A Rev. J.*, **92**, 97–108.

Wang, X. F. and Xu, Y. (2017). Fast clustering using adaptive density peak detection. *Stat. Methods Med. Res.*, **26**(6), 2800–2811.