JCTC Journal of Chemical Theory and Computation

Article

pubs.acs.org/JCTC

pyProCT: Automated Cluster Analysis for Structural Bioinformatics

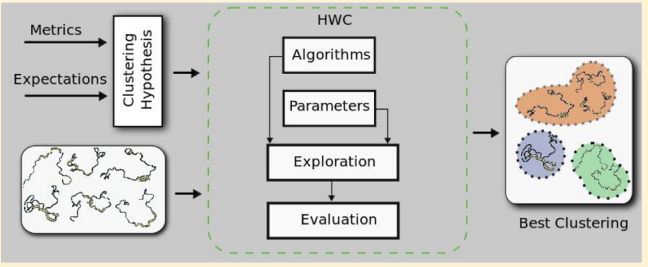
Víctor A. Gil[†] and Víctor Guallar*[,†,‡]

[†]Joint BSC-CRG-IRB Research Program in Computational Biology, Barcelona Supercomputing Center, Jordi Girona 29, 08034 Barcelona, Spain

[‡]Institució Catalana de Recerca i Estudis Avançats (ICREA), Passeig Lluís Companys 23, E-08010 Barcelona, Spain

Supporting Information

ABSTRACT: Cluster analysis is becoming a relevant tool in structural bioinformatics. It allows analyzing large conformational ensembles in order to extract features or diminish redundancy, or just as a first step for other methods. Unfortunately, the successfulness of this analysis strongly depends on the data set traits, the chosen algorithm, and its parameters, which can lead to poor or even erroneous results not easily detected. In order to overcome this problem, we have developed pyProCT, a Python open source cluster analysis toolkit specially designed to be used with ensembles of biomolecule conformations. pyProCT implements an automated protocol to choose the clustering algorithm and parameters that produce the best results for a particular data set. It offers different levels of customization according to users' expertise. Moreover, pyProCT has been designed as a collection of interchangeable libraries, making it easier to reuse it as part of other programs.



1. INTRODUCTION

In structural bioinformatics, data from computational experiments is produced and consumed at great speed. Improvements in software, together with the development of CPUs and accelerators have contributed to increase this data production. Conformational sampling software, such as molecular dynamics or Monte Carlo techniques, have specially enjoyed the benefits of this fast hardware evolution.[1,2] Since raw data is rarely useful per se, it must be processed to extract useful information. Thus, as data sets increase, so does the importance of having unsupervised and computationally cheap tools to analyze them.

Cluster analysis is one of the most successful and widespread tools to postprocess and analyze ensembles of conformations. It has been successfully used to discover near-native structures,[3] to study peptide folding,[4] or as a first step to extract kinetics information[5] (e.g., reaction pathways[6] or relative free energies[7]) among others. As a preprocessing step, it can be used to reduce data sets by extracting the most representative conformations, so that any subsequent computationally expensive postprocessing becomes more affordable.

There are many general use clustering algorithms, all having their own advantages and drawbacks. Some others, such as the ART-2′ algorithm[8] or the fuzzy clustering algorithm presented by Gordon et al.,[9] were specifically created to improve the geometrical analysis of conformational ensembles; others even account for the ensemble's kinetic properties.[5,10]

Implementations of the most traditional algorithms are included as tools into larger multipurpose suites, such as the well-known "ptraj",[11] which can be found in AMBER's suite or GROMACS' "g_cluster".[12] Others, such as "Wordom",[13] offer some clustering options along with tools to analyze MD trajectories. However, there are few software packages that are specifically written to perform cluster analysis over conformation ensembles.

Such diversification gives users plenty of options to choose the best suited algorithm for their problem types and data sets, in order to obtain the best quality clustering. Nevertheless, cluster analysis techniques are not bullet-proof and can be easily misused. This can be dangerous, as unexpected results will go unnoticed, contaminating any subsequent process. Indeed, the successfulness of a cluster analysis process depends mainly on two basic factors, which, if overlooked, can lead to poor or erroneous results:

- Used algorithm: Every clustering algorithm makes its own assumptions about the data set. It is well-known that k-means will perform better when applied to data sets composed of convex clusters and that hierarchical algorithms will obtain better results if the underlying structure is composed of elongated clusters. DBSCAN[14] will have problems clustering data sets with big density differences. The intrinsic characteristics of the data set in terms of cluster size, shape, and density can be a determinant factor to generate a good quality clustering. This reasoning also works the other way round: the clustering algorithm will, at the same time, impose some characteristics to the clusters it produces.
- Parametrization: Many algorithms need to be parametrized before using them. Clustering results can change if these parameters are not tuned carefully. Good examples are the cutoff in hierarchical algorithms

Received: April 10, 2014
Published: July 18, 2014

   dx.doi.org/10.1021/ct500306s | J. Chem. Theory Comput. 2014, 10, 3236−3243

or the 'Eps' and 'MinPts' parameters in DBSCAN (widely studied in the original article[14] and in more recent ones[15]). One parameter, the number of final clusters, has always deserved special attention. Algorithms that need this parameter will usually partition the data set in exactly the number of clusters they are told to use. This can result in a totally artificial partition that will not capture the underlying structure (if any).

Unfortunately, this selection and tuning step requires extensive knowledge of cluster analysis. Users, often experts in other problem domains, have tight time constraints that prevent them from acquiring this knowledge. Therefore, algorithm choice is typically based upon its appearance in other publications, the algorithm reputation, or simply the availability of its implementations. Parameters are, in turn, chosen either randomly or just using implementation default values, which are not always the best suited for the data set under study (an example of the consequences of a bad algorithm/parameters choice can be seen in Figure 4 of Supporting Information). Finally, once the target clustering has been produced, users do not have enough tools to asses its quality, thus reducing the chances of improving it by trial and error.

Here, we want to propose a novel cluster analysis methodology, the Hypothesis-driven Clustering Exploration (HCE), that will help to overcome the aforementioned problems. HCE's approach first converts the user's problem domain knowledge into a clustering hypothesis. Then, an exploration of the clustering space using up to five different algorithms is performed in order to obtain the best clustering fitting the hypothesis. By using HCE, users do not need to be familiar with cluster analysis techniques and its caveats but only to be able to define their problems in terms of a clustering hypothesis. The fact of not having to deal with each algorithm's distinctive features and parameters can potentially improve the quality of the cluster analysis results, which will also better fit the user's expectations and purpose.

HCE methodology has been implemented into pyProCT, an open source (MIT licensed) Python (http://www.python.org) toolkit. This software extends HCE by implementing two of the most common use cases: discovering hidden features of the data set (cluster analysis with prototype extraction) and trajectory compression. pyProCT can also be a powerful tool for more experienced clustering users. Thanks to its highly configurable input script, most of the software behavior can be customized, allowing expert users to better exploit their knowledge. Moreover, the object oriented implementation will allow any developer to easily add new modules or modify the provided ones, allowing them to interface with already written packages, which will be able to benefit from any of the implemented cluster analysis tools.

Finally, pyProCT offers a built-in graphical user interface (GUI) that greatly enhances its usability. Among other features, it allows users to visually review the cluster analysis results so that possible errors can be rapidly detected and the hypothesis improved.

2. METHODS AND PROTOCOL

pyProCT implements the HCE methodology by performing a four-step protocol: distance matrix calculation, clustering space exploration, clustering evaluation, and data extraction (see Figure 1).

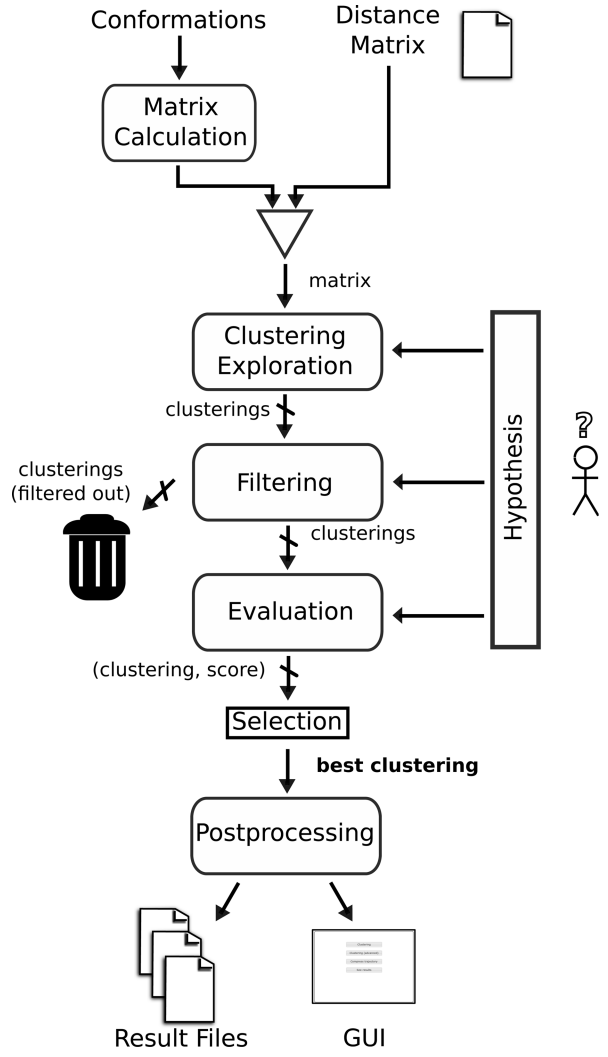


Figure 1. Implementation of HCE methodology in pyProCT.

2.1. Distance Matrix Calculation. Clustering algorithms require to evaluate several times a similarity function that is to be applied to all pairs of elements of the data set. Being one of the computational bottlenecks, its performance can speed up notably by precalculating a pairwise distance matrix. pyProCT offers three matrix generation options:

- Root mean square distance (RMSD) matrix: The RMSD matrix is generated by pairwise superposing all conformations to then calculate their coordinates RMSD. Different parts of the system can be selected to do the superposition and RMSD calculation step. This kind of matrix is the most used in conformational clustering projects (e.g., to cluster protein conformations based on their α carbon RMSD).
- Euclidean distance matrix: Its calculation also involves a two-step process. First, all conformations will be iteratively superimposed (all their coordinates or just a selected subset). Second, the geometric centers of the part of the system we want to analyze are calculated. The pairwise euclidean distances of these centers form the distance matrix. This method is preferred for scenarios where users want to extract information about the relative placement of one part of the system in motion

with respect to a point of reference (e.g., ligand−receptor systems).

- External matrix loading: As the distance metrics included may not be the best for all systems[16,17] and users may want to implement their own similarity functions, the third option allows them to load an external pregenerated matrix.

2.2. Clustering Space Exploration. The number of ways of grouping the elements of a data set, the clustering space, is huge (superexponential in n[18]). Within this space, it is accepted that only few clusterings have enough quality to attain valuable information; the existence of a unique clustering is a subject of debate.[19,20] Obtaining these representative clusterings might involve an exhaustive search of the clustering space assessing their quality. This approach is, in most cases, computationally unaffordable due to the combinatorial explosion.

pyProCT's approach uses a heterogeneous set of clustering algorithms as exploration agents that perform a guided search of the clustering space restricted to the area containing meaningful (not random) clusterings. The more different the used algorithms are, the wider the exploration is, thus increasing the probability of visiting the best quality clustering. Currently, pyProCT implements one representative of each of the most important clustering algorithm families, namely, hierarchical clustering,[21] k-medoids (partitive algorithms), DBSCAN,[14] gromos[4] (both density-based algorithms), and spectral clustering[19] representing connectivity/spectra-based clustering family.

Since the choice of the algorithm's parameters affects notably the result, pyProCT exploits it as another source of variability to yet widen the explored area of the clustering space. In our implementation, the working hypothesis is also involved in the strategies to obtain the parameters for each algorithm.

A brief description of the algorithms and the parametrization methods implemented are further explained in Supporting Information section 1.

2.3. Clustering Evaluation. Some authors warn that 'good quality' clusterings are those that allow users to accomplish their goals.[22] Therefore, it is not possible to asses the quality of a clustering without a previous definition of user's expectations. In pyProCT, this is achieved through the definition of a working hypothesis that is composed of a mixture of subjective and objective elements. Subjective elements (e.g., an estimation of the final number of clusters or the noise the user thinks the data set has) add contextual and domain-based information, directly derived from the users' purpose and expectations as well as their expertise in the system under study.

In pyProCT, the best solution is the one that best fits the clustering hypothesis. This hypothesis is used twice in the evaluation step: to eliminate clusterings too different from user's expectations and to score the remaining ones.

- Filtering: All unsuitable clusterings generated during the exploration are rejected. In this step, the expected number of clusters, the amount of produced noise, and the cluster size (number of conformations inside that cluster) are checked. The purpose of this filtering is twofold: it enforces the clustering hypothesis and improves the overall efficiency of the process, as it reduces the number of clusterings to be evaluated in the next step.
- Clustering evaluation: The specification of metric-based quality criteria adds an objective dimension to the hypothesis by describing measurable traits of each solution (e.g., cluster separation). Clustering quality is usually measured using two approaches. External cluster validation indices compare clusterings with an already known correct answer.[23−25] Instead, internal cluster validation indices (ICVs) only use the internal information on the clustering to evaluate it (e.g., distance between elements) to evaluate it. Given that the correct answer is not known a priori, pyProCT only uses ICVs. Unfortunately, these indices tend to impose their assumptions about the data set layout and characteristics, penalizing those that do not fit them. It has been observed, however, that combining some ICVs can help to overcome this problem.[26] One or more criteria (i.e., a scoring function based on a linear combination of normalized ICVs) must be defined. The clustering with the highest score for any of them is chosen to be the best clustering. Users are also ultimately responsible for choosing the ICVs that compose the criteria. Similar strategies have been used previously in other works, using the silhouette coefficient,[27] or Pal et al.[28] approach, which is closer to the method discussed here.

pyProCT's default behavior defines a simple criteria that fosters both cluster compactness and separation. More information about the implemented ICVs and other query types can be found in Supporting Information section 2.

The exploration and evaluation steps are the core of the HCE strategy. It prevents users from having to learn about clustering algorithms and the meaning and use of their parameters, thus making the whole cluster analysis process less prone to generate unexpected/undesired results. However, the success of the process will still depend on the ability of users to express their hypothesis using the currently implemented tools. We do not recommend to start with too specific hypotheses unless the user has good knowledge of the data set.

2.4. Postprocessing. Finally, the best clustering will be used to extract information from the data sets. Currently, pyProCT implements two use cases.

- Cluster analysis: A set containing all the cluster prototypes (the prototype is the central conformation of a cluster, i.e., the conformation with the minimum distance from all other conformations in the cluster) of the best clustering will be generated. It will also record which conformations were assigned to each of the clusters, so that users can know which conformation belongs to each cluster.
- Compression: In this case, the user will define a target number of conformations (lower than the size of the input ensemble). pyProCT will use the best clustering in order to eliminate the per-cluster redundancy so that the reduced input ensemble contains the chosen target number of conformations. The compression protocol first calculates the target number of elements of each cluster so that it is proportional to the initial population. Then, it applies a k-medoids algorithm to each cluster to obtain a number of 'local clusters' equal to the target number of elements. The prototypes of each of the new 'local clusters' are stored as elements of the compressed set. This way the space is fairly partitioned and we ensure that all new elements are good representatives of the input data.

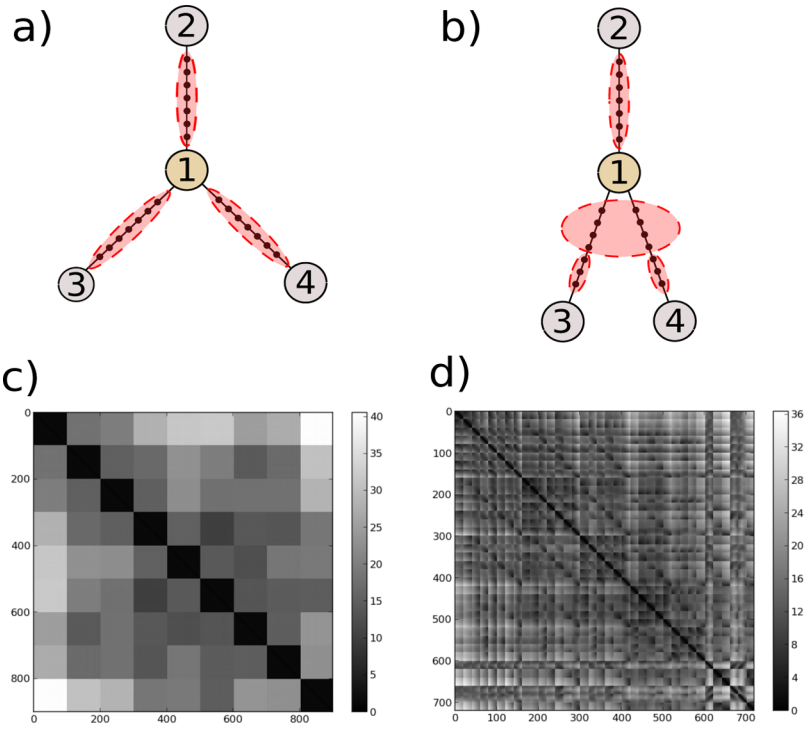


Figure 2. (a and b): New conformations (black dots) are created by interpolating from a base conformation (light-brown colored) to a target conformation (in gray), forming the interpolation sets (in red). All pairs of initial conformations without repetition are used. (a) If all target conformations are equally separated, interpolation clusters are easily separable and can be organized in multiple ways to form interpolation metaclusters. (b) The existence of distance asymmetries will difficult the partitioning into metaclusters. (c) RMSD matrices of the 900 conformations ensemble and (d) the 720 one. Clusters in matrix c are well-defined due to the small magnitude of the added noise. The small visible structures in d correspond to the interpolation clusters.

2.5. Code and Interface. pyProCT is a complete clustering toolkit written mainly in Python. It can be used as a standalone program or, thanks to its high modularity, as part of other software packages.

pyProCT offers different degrees of control according to the method chosen to run the clustering job. The easiest way to use it is through the included browser-based GUI, organized as a wizard. It includes visualization tools, allowing the user to preview the selections needed to superimpose and calculate distances. Finally, it can generate a complete graphical representation of the clustering results. Among other features, it allows visualizing the most representative structures of each of the generated clusters and includes tools to visually validate the results. It also allows plotting the ICV values of all accepted clusterings, being this a tool of utmost importance for users who want to better understand the behavior of applied ICVs and criteria in order to improve the working hypothesis. When used locally, the GUI will connect to an installed pyProCT instance to run the analyses and show the results. It also allows users to download the generated script, so that they can further modify it or use it in remote machines. The GUI has been written in Python (the Web server) and HTML with Javascript (the front end). A detailed list of all the Javascript libraries used can be found in the documentation.

pyProCT can also be used as a command line program that receives a JSON (JavaScript Object Notation) string as its input script. Inside the script the different actions and their parameters are defined; algorithms can be manually selected and their parameters specified at will. The use of this script represents the finest type of control that a user can have over pyProCT (without modifying the code).

Complete specifications of the control script are detailed in the Supporting Information section 3.

2.6. Performance. PyProCT uses the pyRMSD[29] package in order to efficiently calculate pairwise RMSD matrices. Furthermore, the use of the C-written condensed matrix module included in the same package has also helped us to considerably speed up the code thanks to its faster access times and its implementation of some simple neighboring queries. Cython (http://cython.org) generated code has been used to enhance the calculation speed of some quality functions. A Python+C implementation of the hierarchical algorithm has been also used.[30]

Finally, both the clustering search and the clustering evaluation steps have been parallelized by using a parallel task scheduler. One of the two available scheduling types uses Python's 'multiprocessing' module to allow to fully use the cores of a single machine. The second scheduling type is based on MPI (through the mpi4py interface[31]) and is better suited for distributed environments.

2.7. Testing and Validation. The whole framework has been tested using the unit testing technique, with a good testing coverage. The tests of some of the most consolidated parts have been changed to regression tests. A simple validation script of the HCE method over 2D data sets is also available into the code repository. A short discussion about the validation can be found in Supporting Information section 6.

3. APPLICATIONS AND RESULTS

3.1. RMSD-Based Cluster Analysis over Synthetic Conformational Ensembles. *3.1.1. Ensemble Generation.* From an initial Ubiquitin conformation, we have performed an

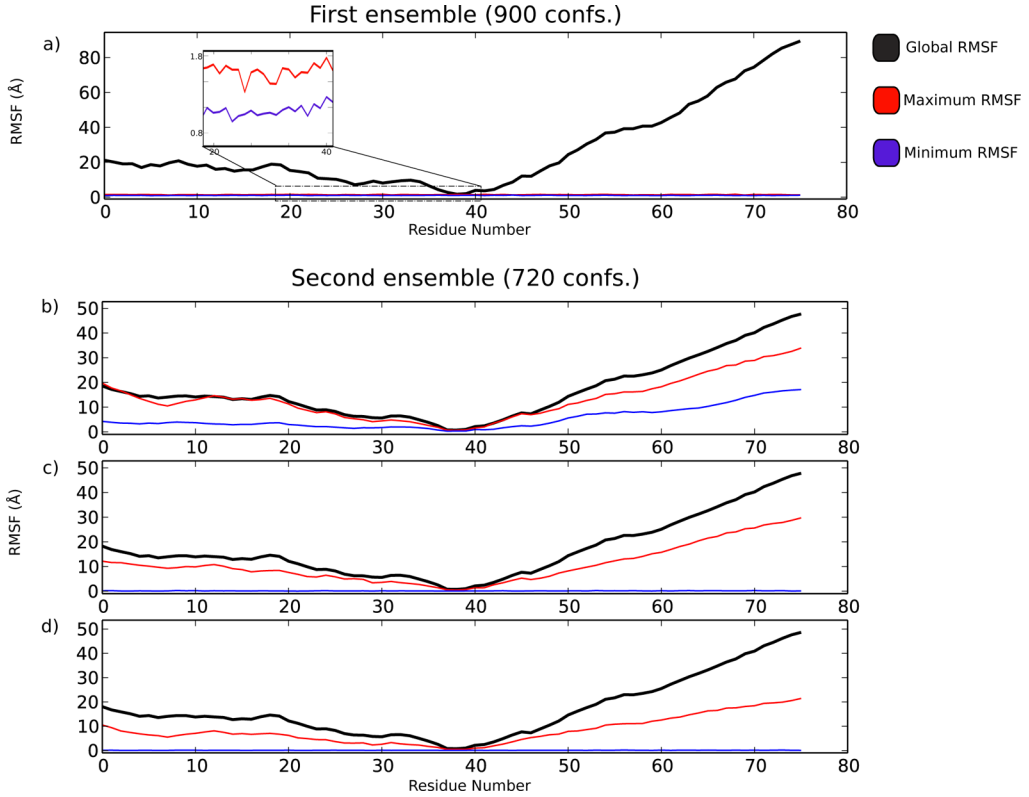


Figure 3. Global, maximum, and minimum RMSF plots. In all cases, the global RMSF plot high values (especially in the head and tail of the protein) are a result of the big global residue movements induced by the torsional changes. For the first ensemble (a), the resulting clustering is able to successfully separate the conformations, which is reflected by low values of $RMSF_{min}$ and $RMSF_{max}$, that only capture the added noise. For the second ensemble (b,c,d), $RMSF_{min}$ and $RMSF_{max}$ show a decrease in its values as the software is able to generate better clusterings in terms of the separation of residue movement.

iterative process in which a torsion angle is randomly chosen and modified at each step. We have produced a set of nine conformations (including the initial nonmodified one) that have great pairwise RMSD differences (17.75 Å mean, 8.85 Å std. dev.).

Using these nine initial conformations, we have generated two ensembles:

- A 900 conformations ensemble was produced by generating 100 random noise structures, using a $[-1, 1]$ Å range for each coordinate, applied to each initial conformation. The resulting ensemble contains nine well-defined clusters with sharp boundaries that are easily identifiable in its RMSD matrix plot (Figure 2c).
- A 720 conformations ensemble was created by interpolating 20 new structures for each of the possible 36 pairs (without repetition) of the initial set, without including any of the original conformations. Each intermediate conformation is calculated using the formula $b_i = b + ((0.2 + (0.6/20)i)(t - b))$, where b and t are the base and target conformations, respectively. The resulting ensemble contains up to 36 small clusters of 20 conformations each: the interpolation sets (Figure 2a and b). The resulting RMSD matrix is shown in Figure 2d.

3.1.2. First Ensemble Cluster Analysis. For the cluster analysis on the first ensemble, we pretend to have superficial knowledge of its generation process so that the exact data layout is unknown, although we know that some well-defined clusters may exist. The first step was to choose which algorithms to use as well as their parameters. In this case, we will use the default options, that is, pyProCT will try all the algorithms and calculate different parameter sets for each of them. The second step is to define the clustering hypothesis. Given the limited information we have about the ensemble, it has to be loose enough to allow pyProCT to effectively explore the clustering space. The clustering hypothesis we have chosen instructs pyProCT to limit its search to clusterings with at least 3 clusters and at most 50 with a minimum size of 20 conformations. The default evaluation criteria has been used, which means that clustering score is calculated summing the 40% of its cohesion index value plus the 60% of its silhouette index value (that gives separation and compactness information at the same time), stressing in this way the contribution of compactness to the score. In terms of the clustering hypothesis, this means that we prefer clusterings where clusters are compact over clusterings where clusters are very separated.

The definition of a clustering size range in the hypothesis prevents the user from having to specify a concrete value for the desired number of clusters, a mandatory parameter in some algorithms, helping to avoid the creation of artificial partitions. Indeed, after applying pyProCT, the resulting clustering contains nine equally sized clusters, which is the expected clustering size. To check whether every cluster contains the expected elements, we can inspect the results file (The file format is described in Supporting Information section 4), where all generated clusterings have been saved in human-readable form. Another option is to use the results viewer included in the GUI: in some cases, especially when a geometrical similarity metric is used and a good balance of

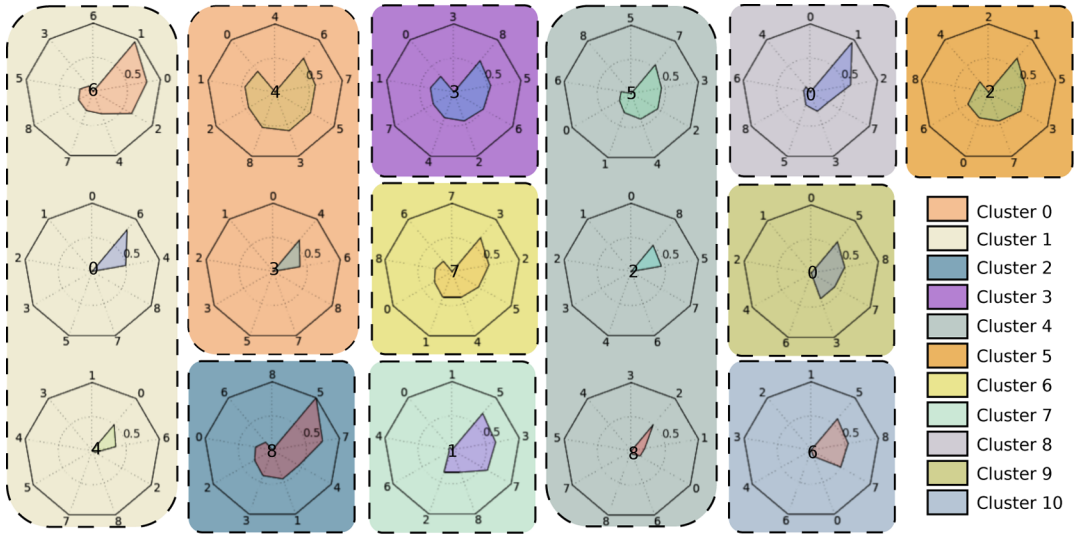


Figure 4. Each shadowed rectangle shows the contents of each of the 11 clusters for the first clustering. Each radial plot shows the percentages of all interpolation sets generated from the base conformation (which id is written in its center) to each of the other target conformations. All clusters contain elements from more than one interpolation set. In particular, clusters 1, 2, and 8 contain an almost complete interpolation set (from conformation 1 to 6, from 5 to 8, and from 0 to 1, respectively).

compactness/separation is needed, cluster contents can be indirectly checked by inspecting the global and per-cluster root-mean-square fluctuation (RMSF) plots. In order to calculate the RMSF of a set of m conformations of n_{res} residues, all conformations are iteratively superimposed. Then RMSF for each residue is calculated following eq 1, where r_{ref} is the mean conformation, $i \in [1,n_{res}]$ is the number of residue and j is the number of conformation. The residue position is represented by its α carbon position.

$$\mathrm{RMSF}^i = \sqrt{\frac{1}{m}\sum_{j=1}^{m}(r_j^i - r_{\mathrm{ref}}^i)^2} \quad (1)$$

When applied to the ensemble, the RMSF plot gives us an intuition of the global per-residue mobility. If applied to the conformations inside a cluster it shows how much of this global movement was captured by it. In this work, we will also use RMSF_{max} and RMSF_{min}, two derived measures that account for the maximum and minimum RMSF values of all clusters (see eqs 2 and 3). These two measures have been used to compare the collective per-cluster RMSF with the global one. If a clustering does not succeed to correctly capture structural variance, the values of local RMSF will be closer to the global RMSF and so will be the value of RMSF_{max}. Lower values of local RMSF will be reflected as lower values of RMSF_{min}. Both RMSF_{max} and RMSF_{min} give us a fast way of comparing per-cluster and global RMSF, even in situations when the clustering size (the number of clusters) is big. In Figure 3a, we can see how the global RMSF plot captures wide displacements in α carbon positions, induced by the torsional changes in the initial conformation. However, in this easy case, RMSF_{max} and RMSF_{min} plots show fluctuations between 0.9 and 1.1 Å, meaning that clusters are just capturing the noise added to each of the conformations of the initial ensemble, and indirectly confirming that cluster contents are correct (elements were separated correctly).

$$\mathrm{RMSF}^i_{min} = \min(\mathrm{RMSF}^i_1, \mathrm{RMSF}^i_2, ..., \mathrm{RMSF}^i_{n_{clusters}}) \quad (2)$$

$$\mathrm{RMSF}^i_{max} = \max(\mathrm{RMSF}^i_1, \mathrm{RMSF}^i_2, ..., \mathrm{RMSF}^i_{n_{clusters}}) \quad (3)$$

3.1.3. Second Ensemble Cluster Analysis. Again, we want to recreate a scenario in which the user does not have full information about how the data set was generated. For our first cluster analysis attempt, we will use pyProCT's default hypothesis so that we can learn more about the data set.

The resulting clustering contains 11 clusters. The fact that their sizes are all bigger than a single interpolation set, indicates that we are obtaining interpolation metaclusters (clusters containing elements from more than one interpolation set). The percentage of each interpolation set in each cluster (Figure 4) shows that only 3 out of the 11 clusters contain a complete interpolation set; the 8 remaining clusters contain partial interpolation sets that, in most cases, share the same base conformation. These can be explained by the distance asymmetries present in the original 9 conformations ensemble (see Figure 2b). The RMSF plot in Figure 3b shows large RMSF_{max} and RMSF_{min} values, meaning that clusters are capturing wide residue fluctuations. These values warn us about the quality of the clustering, pointing us to reevaluate the validity of the default hypothesis. In this second attempt, we will loosen the hypothesis by allowing more and smaller clusters. The number of clusters range has been maintained, but the minimum cluster size has been lowered to 15. Furthermore, we increase the noise to 5% (unclustered elements), giving pyProCT more freedom to explore. Finally, we change the default evaluation criteria so that the final score is calculated using a 66% of silhouette index value and a 33% of cohesion value. This gives a slightly higher weight to the separation/compactness ratio than the default criteria. After performing the exploration, we have found a clustering composed of 35 clusters with 20 conformations each, and one with 15. If we look for them in the results file, we can see that each of these clusters corresponds to one of the interpolation sets (the missing five elements being interpreted as noise). The RMSF plot in Figure 3c reflects a remarkable decrease of the RMSF_{min} value, meaning that one or more clusters were able to capture the local fluctuations. The RMSF_{max} value, however, only decreases

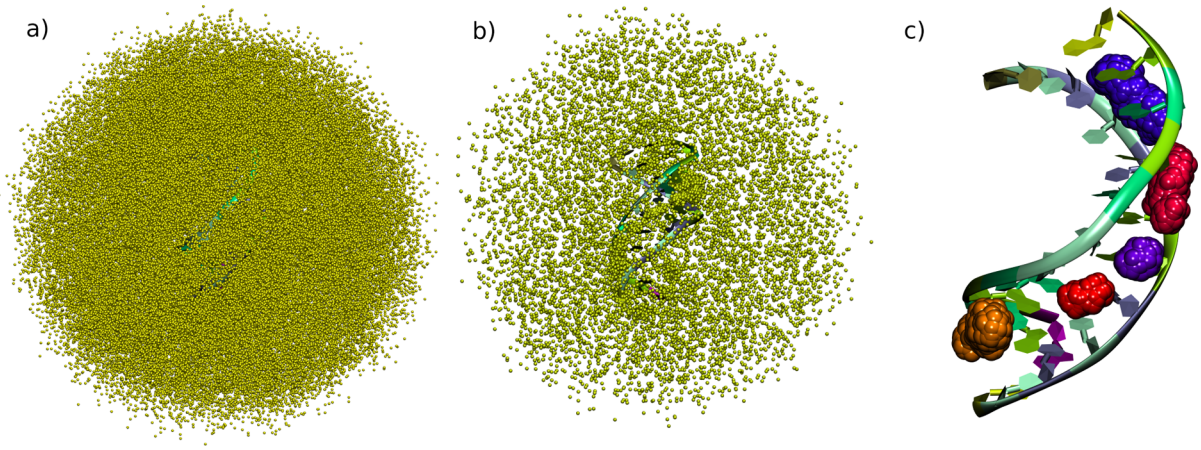


Figure 5. Atomic level representations of the MD trajectory data set. In part a, the whole data set is shown. It is worth noting that the cluster formed by the bulk solvent covers the clusters of interest (around the binding sites). This data set's redundancy was reduced, being part b, the resulting compressed data set. Finally, in part c, the five retrieved clusters are shown along with the DNA strand.

to some extent, suggesting that many clusters are still capturing wide fluctuations: the clustering may still be improved.

The last attempt of improving our cluster is to further loosen the working hypothesis. This time, we will let pyProCT produce even more clusters (in the range 3 to 100). Larger number of clusters involves lowering the minimum cluster size, set here to only 10 elements. Moreover, generating a larger number of clusters (of smaller size) makes the clustering more prone to producing noise. To address this issue, we raised the allowed clustering noise to 10% of the elements.

The resulting clustering contains up to 49 clusters with an average size of 14 elements. With a layout beyond our intuition, this last clustering is the one that best separates the ensemble conformations. As a consequence, its $RMSF_{max}$and $RMSF_{min}$values are the lowest of the three clusterings we have generated (see Figure 3d).

This illustrates how pyProCT can find good quality clusterings if enough freedom is given to it. What is more, the iterative scheme shown here (Figure 4) helped us to gain more insight about the data and its layout by creating three hypotheses adapted to our increasing knowledge of the data set.

3.2. Clustering of an MD Trajectory. In our last application example, we want to cluster analyze the cisplatin−DNA interaction data set used by Lucas et al.[32] This data set holds a 3.7 s molecular dynamics MD simulation stored in 126 857 frames (Figure 5a). It shows the electrostatic preassociation of cisplatin, a clinically relevant drug used to treat several types of cancer, with a DNA strand.

When performing a cluster analysis on this data set the first obstacle we find is its huge size. To handle its pairwise RMSD matrix, at least 29 GB of RAM would be needed. This makes the analysis computationally unfeasible in common workstations and nonspecialized hardware architectures. To overcome this issue, we have first divided the trajectory into 12 chunks of 10 572 frames (the 12th part is 7 frames smaller). Then, we have used the compression feature of pyRMSD to reduce the size of each piece below 1500 frames. Finally, all compressed partial trajectories have been merged. Since the addition of local compressions could have added redundancy, we have compressed the resulting trajectory again in order to have about 15 000 frames (Figure 5b).

A quick visualization of the data set reveals a second difficulty: relevant clusters are hidden by a large convex cluster that corresponds to random posotions of the drug away in bulk solvent space (Figure 5a). This bulk cluster has different density, size, and shape than the clusters we are interested in, thus complicating the analysis. This data set represents a good example of how a clustering analysis that is trivial for the human brain can be difficult to automatize.

The clustering hypothesis we have used reflects what we have learned from this visual inspection. We have allowed a huge noise generation (maximum noise of 80%) in order to permit the classification of the bulk cluster as noise. In addition, we have decided to let pyProCT choose clusterings containing from 3 to 20 clusters, with at least 50 elements inside each cluster. All other parameters were left with their default values.

The resulting clustering (Figure 5c) contains five clusters in which positions and relative populations correlate well with the ones shown in the kinetic analysis performed by Lucas et al.[32] To obtain this final clustering, pyProCT generated 314 clusterings, from which 164 were directly rejected. The whole exploration process took 90 min using five working threads plus a nonproductive control thread in a 4-cores (8 threads) Intel Xeon W3530 @ 2.80 GHz workstation with 12 GB of RAM. Thanks to the use of pyRMSD,[29] the matrix calculation step, one of the bottlenecks of cluster analysis software, took only 19 s (0.35% of the total time).

4. CONCLUSIONS

pyProCT is an open source Python cluster analysis toolkit that can work as a standalone program or as part of other projects. Its implementation of the HCE method can help users with little or no previous experience in cluster analysis to produce more reliable results. In addition, the high level input customization makes it a powerful tool when used by experts.

We have shown how it can be applied to common use cases of cluster analysis: feature extraction and redundancy elimination. In the first conformational clustering case presented, pyProCT's default hypothesis was enough to guess the algorithm and parametrization (including the number of clusters) that produced the best clustering. In the second proposed example, we showed an intuitive iterative scheme that allows us to refine the default hypothesis in order to improve the results and to gain insight into our data set structure. Finally, we have shown how to use it in a spatial cluster analysis

scenario, where a large and noisy data set was compressed to an easy to handle ensemble.

pyProCT is available at https://pele.bsc.es/pele.wt/tools.

■ ASSOCIATED CONTENT

Supporting Information

Implemented clustering algorithms; clustering properties and quality functions; input and output file formats. This material is available free of charge via the Internet at http://pubs.acs.org.

■ AUTHOR INFORMATION

Corresponding Author

*Email: victor.guallar@bsc.es.

Notes

The authors declare no competing financial interest.

■ REFERENCES

(1) Shaw, D. E.; Chao, J. C.; Eastwood, M. P.; Gagliardo, J.; Grossman, J. P.; Ho, C. R.; Lerardi, D. J.; Kolossváry, I.; Klepeis, J. L.; Layman, T.; McLeavey, C.; Deneroff, M. M.; Moraes, M. A.; Mueller, R.; Priest, E. C.; Shan, Y.; Spengler, J.; Theobald, M.; Towles, B.; Wang, S. C.; Dror, R. O.; Kuskin, J. S.; Larson, R. H.; Salmon, J. K.; Young, C.; Batson, B.; Bowers, K. J. *Commun. ACM* **2008**, *51*, 91.

(2) Stone, J. E.; Hardy, D. J.; Ufimtsev, I. S.; Schulten, K. *J. Mol. Graphics Modell.* **2010**, *29*, 116−125.

(3) Zhang, Y.; Skolnik, J. *J. Comput. Chem.* **2004**, *25*, 865−871.

(4) Daura, X.; Gademann, K.; Jaun, B.; Seebach, D.; van Gunsteren, W. F.; Mark, A. E. *Angew. Chem., Int. Ed. Engl.* **1999**, *38*, 236−240.

(5) Prinz, J.-H.; Wu, H.; Sarich, M.; Keller, B.; Senne, M.; Held, M.; Chodera, J. D.; SchÃOEtte, C.; NoÃ©, F. *J. Chem. Phys.* **2011**, *134*, 174105.

(6) Noe, F.; Schutte, C. *Proc. Natl. Acad. Sci. U.S.A.* **2009**, *106*, 19011−6.

(7) Takahashi, R.; Gil, V. A.; Guallar, V. *J. Chem. Theory Comput.* **2013**, *10*, 282−288.

(8) Karpen, M. E.; Tobias, D. J.; Brooks, C. L. *Biochemistry* **1993**, *32*, 412−420.

(9) Gordon, H. L.; Somorjai, R. L. *Proteins* **1992**, *14*, 249−264.

(10) Haack, F.; Fackeldey, K.; Röblitz, S.; Scharkoi, O.; Weber, M.; Schmidt, B. *J. Chem. Phys.* **2013**, *139*, 194110.

(11) Shao, J.; Tanner, S. W.; Thompson, N.; Cheatham, T. E. *J. Chem. Theory Comput.* **2007**, *3*, 2312−2334.

(12) Berendsen, H. J. C.; Spoel, D. V. D.; Drunen, R. V. *Comput. Phys. Commun.* **1995**, *91*, 43−56.

(13) Seeber, M.; Cecchini, M.; Rao, F.; Settanni, G.; Caflisch, A. *Bioinformatics* **2007**, *23*, 2625−2627.

(14) Ester, M.; Kriegel, H.-P.; Sander, J.; Xu, X. *Kdd* **1996**, 226−231.

(15) Zhou, H.; Wang, P.; Li, H. *J. Inf. Comput. Sci.* **2012**, *9* (7), 1967−1973.

(16) Cossio, P.; Laio, A.; Pietrucci, F. *Phys. Chem. Chem. Phys.* **2011**, *13*, 10421−10425.

(17) McGibbon, R. T.; Pande, V. S. *J. Chem. Theory Comput.* **2013**, *9*, 2900−2906.

(18) Meila, M. Comparing Clusterings: An Axiomatic View. *Proceedings of the 22nd International Conference on Machine Learning* **2005**, 577−584.

(19) Luxburg, U. *Stat. Comput.* **2007**, *17*, 395−416.

(20) Kleinberg, J. *Adv. Neural Inf. Process. Syst.* **2002**, 446−453.

(21) Ward, J. H. *J. Am. Stat. Assoc.* **1963**, *58*, 236.

(22) Guyon, I.; Luxburg, U. V.; Williamson, R. C. *Adv. Neural Inf. Process. Syst.* **2009**.

(23) Rand, W. M. *J. Am. Stat. Assoc.* **1971**, *66*, 846.

(24) Reichart, R.; Rappoport, A. The NVI Clustering Evaluation Measure. *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*; Stroudsburg, PA, 2009; pp 165−173.

(25) Meila, M. Comparing Clusterings by the Variation of Information. In *Learning Theory and Kernel Machines*; Scholkopf, B., Warmuth, M. K., Eds.; Lecture Notes in Computer Science 2777; Springer: Berlin Heidelberg, 2003; pp 173−187.

(26) Kryszczuk, K.; Hurley, P. Estimation of the Number of Clusters Using Multiple Clustering Validity Indices. In *Multiple Classifier Systems*; Gayar, N. E., Kittler, J., Roli, F., Eds.; Lecture Notes in Computer Science 5997; Springer: Berlin Heidelberg, 2010; pp 114−123.

(27) Ng, R. T.; Han, J. Efficient and Effective Clustering Methods for Spatial Data Mining. *Proceedings of the 20th International Conference on Very Large Data Bases*, San Francisco, CA, 1994; pp 144−155.

(28) Pal, N.; Biswas, J. *Pattern Recogn.* **1997**, *30*, 847−857.

(29) Gil, V. A.; Guallar, V. *Bioinformatics* **2013**, *29*, 2363−2364.

(30) Mullner, D. J. *Stat. Soft.* **2013**, *53*, 1−18.

(31) Dalcin, L.; Paz, R.; Storti, M.; D'Elia, J. *J. Parallel Distrib. Comput.* **2008**, *68*, 655−662.

(32) Lucas, M. F.; Cabeza de Vaca, I.; Takahashi, R.; Rubio-Martinez, J.; Guallar, V. *Biophys. J.* **2014**, *106*, 421−429.