dQCuts: Simplifying Large Molecular Dynamic Simulations using Higher Order Statistical Signatures

Gabriel Vacaliuc[*]

Under the supervision of Arvind Ramanthan Ph.D.[**]

[*]Oak Ridge High School

[**]Oak Ridge National Laboratory

June 6, 2016

Abstract

Analysis of protein structures has shifted into a primarily computational biology realm in recent years. The caveat of Molecular Dynamic trajectories is the difficulty of the analyis after generation as a result of the data size and high dimensionality. Trajectories often grow larger than tens of Gigabytes and can reach Terabytes given a high-resolution and long simulations. These data sizes challenges analysis on a single machine because of the limitations in memory. The high dimensional landscape of protein structures prevents direct visual analysis and interpretation. However, in order to identify and study elusive conformational states, development of automated software to extract, probe, and analyze these states is necessary. To overcome these challenges, I present dQCuts, a new software package which automatically clusters trajectories of protein conformations in order to isolate distinct states during protein folding. dQCuts is the culmination of multivariate analysis and spectral clustering tailored for MD simulation analysis. Using a simulation consisting of 123 trajectories modeling the protein Ubiquitin, dQCuts managed to isolate 7 distinct folding substates. Understanding these functional states of proteins provides reference structures for Neutron scattering experiments and aids in drug discovery.

1 Introduction

Proteins are responsible for most of the work in a cell, varying from coordinating messages to protecting the body from viruses. To accomplish these tasks, proteins are dynamic, constantly folding and shifting, allowing or preventing access to relevant functional groups. Thus, a single protein may have several functionally relevant folding states, in which to perform specfic function. The nanoscale size of proteins makes identifying these states difficult, because optical imaging techniques cannot magnify sufficiently. X-ray crystallography and electron microscopy provide static images of a protein's structure, but rely on averaging millions of images, as a result of shifting molecules and damage caused by the high energy X-ray and electron beams. The averaging prevents the imaging of single structure, and inherently prevents observing changes in conformational structure over time. Thus, recovery of an approximation to the general structure is made possible through X-ray crystallography, but highly flexible portions of the proteins can be lost in the averaging as a result of large variance in the conformations [1].

With the rise of High Performance Computing, simulating physical problems is becoming common, with techniques such as Computational Fluid Dynamics and Molecular Dynamics (MD). Although computationally intensive, a protein's atoms and their forces on each other can be simulated to produce a dynamic model of protein folding over time based off an initial structure discovered by X-ray crystallography and Nuclear Magnetic Resonance. While MD simulations produce high fidelity models, they generate thousands to millions of frames and Gigabytes to Terabytes of data, leading to difficult data analysis [2]. In order to identify the functional folds, software capable of handling the large data and high dimensionality is needed.

The goal of this research is to develop analytic software to isolate functional conformational states in long MD simulations. This addresses the demand and need for analysis of large simulations, especially those too large to manipulate in Random Access Memory (RAM).

2 Background

2.1 Protein Structure

A protein is a macromolecule consisting of several bonded amino acids. Each amino acid contains bonded Nitrogen, Carbon$_\alpha$, Carbon, and Oxygen which serve as the peptide backbone of the fully connected protein. A peptide is defined as the compound formed by the condensation reaction of two amino acids. The amino acids are bonded together by peptide bonds through this condensation reaction, creating a polypeptide after several acids have bonded together. As seen in Figure 1, when a peptide bond is formed, the hydroxyl group on the last carbon atom of the first acid bonds with a hydrogen atom attached to the Nitrogen atom of the adjacent acid, expelling a water molecule and forming a single Carbon-Nitrogen bond [3]. The dehydrated acids are referred to as amino acid residues and denote the primary structure of a protein. The secondary structure refers to special patterns created by the bonding of amino acid side chains such as α-helices and β-sheets. The tertiary structure is the folded state of the protein, including bonding and arrangement of side chains, which ultimately controls the function and application of the protein [4].

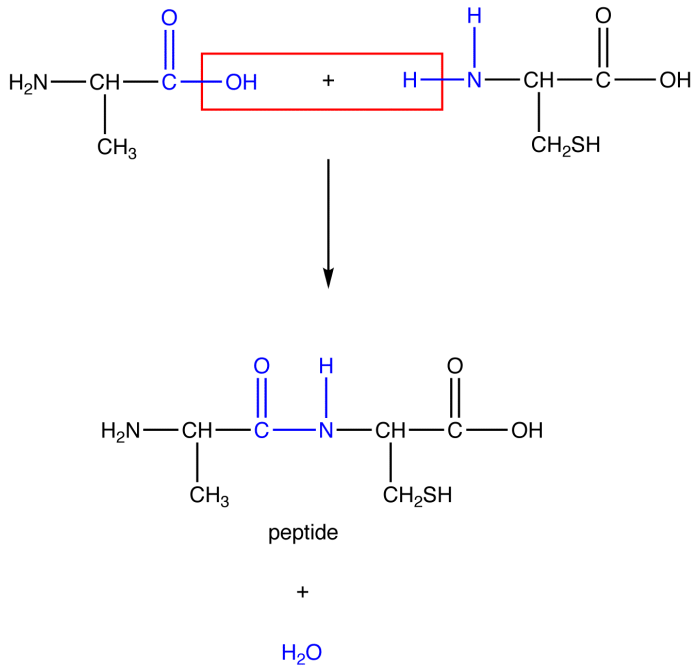

Figure 1: The diagram shows two amino acids bonding to form a peptide and expelling a water molecule [3].

Understanding the tertiary structure is key because it controls the functionality of the protein. For-

tunately, MDAnalysis, a software package for Python interprets trajectories, or data files which dictate the movement of atoms in a simulation, generated from software such as AMBER, GROMACS, and CHARMM [5, 6, 7, 8]. Beyond providing coordinate data of a protein's atoms, MDAnalysis can access various information contained in trajectories, or compute simple statistics which aid in the understanding of structure, energy, or side chain location of a single trajectory. MDAnalysis is therefore ideally suited to develop more advanced software for protein analysis in Python, as it provides infrastructure for extracting data and computing statistics.

Each acid residue contains a C_α atom, the center backbone atom, which bonds to the functional group of the residue. Thus, observing only the C_α atom is a good approximation to the protein backbone because of its centralized location in the residue and connection to the functional group. However, because coordinates are vectors, the direction of the vector is utterly dependent on the basis of the vector space. During the generation of a protein simulation, the protein may rotate or translate its position such that successive frames of a simulation may seem vastly different when the two structures are indiscernible on the right basis vectors. This is a standard issue known as the Kabsch or Procrustes problem which is solved by the Kabsch Algorithm and commonly applied to superimposing C_α atomic coordinates in bioinformatics [9]. However, this is an extraneous step which is resultant of the use of a vector quantity to represent conformational structure.

The alignment can be avoided by observing the backbone through dihedral torsion angles. Dihedral angles are defined by two planes, or at least four points in which the two planes share two points and each contain one of the remaining two points. Torsion angles are simply a measure of the "twist" along a bond. Figure 2 shows an example of a dihedral torsion angle along the $B - C$ atomic bond. Along the protein backbone, there are several dihedral angles which can provide information in which to characterize a specific conformation. Tracking the sines and cosines of the Φ and Ψ angles, identified in Figure 3, provides four independent scalar quantities per residue[1] which represent the conformational "structure".

[1]The dihedral definition requires $n-1$ and $n+1$ atoms for the n^{th} residue Φ and Ψ atoms, respectively, thus for a protein with N residues, there are only $N-1$ Φ and Ψ angles each.

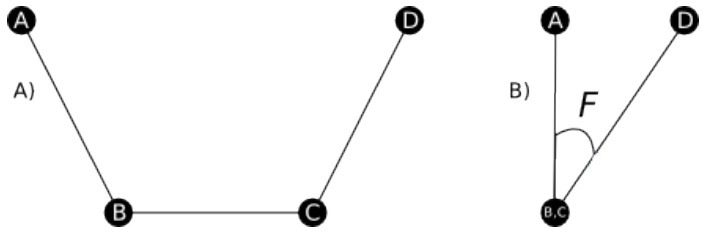

Figure 2: The diagram shows a four atom molecule with a torsion dihedral angle along the $B-C$ bond in a A) side view and shows the B) definition of F along the $B-C$ bond.

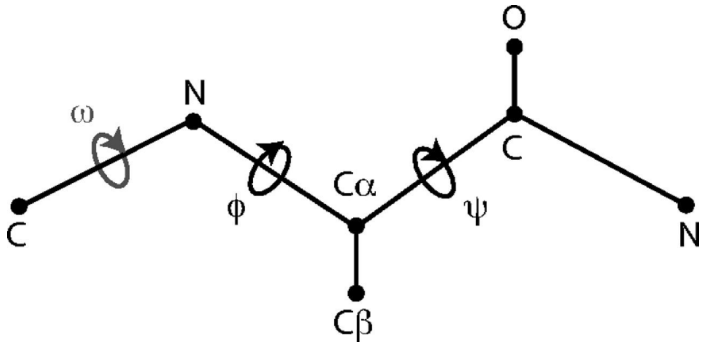

Figure 3: The diagram shows the location of torsion dihedral angles along the protein backbone.

2.2 Software

Quasi-Anharmonic Analysis (QAA) is software which relies on the anharmonicity of proteins to project protein coordinate data onto a smaller subspace [10]. The analysis performed by QAA utilizes a technique in blind source signal separation which develops anharmonic basis vectors to maximize the fourth order statistic, kurtosis. The Joint Approximate Diagonalization of Eigen-Matrices (JADE) builds upon Principal Component Analysis (PCA) which has similar motivation by maximizing the skewness of the data rather than the variance [11]. This is especially useful when dealing with biological data because of the rareness of some significant conformational states in proteins. Performing JADE will project the data, separating and clustering it into groups of outliers skewed greatly from the majority data. These conformational groupings have separated anharmonically and now represent folding states or sub-states. The dimensionality reduction of a single trajectory is accomplished with QAA to aid in the isolation of folding states, but retains unitary input and lacks automated clustering.

Clustering requires the similarity between frames to be explicitly determined, and computing affinity between a million frames of high dimensional data is computationally intensive. Fortunately, PCA can

preserve nearly all a dataset's variance in a fraction of dimensionality. However, PCA is not an ideal choice for analyzing protein folding trajectories as the simulations are inherently anharmonic, and many conformations may be perturbations or oscillations away from the native state [10]. Gaussian data allows PCA to perform ideally because of PCA's blind maximization of the variance of a dataset.

This specific type of Independent Component Analysis (ICA), JADE, maximizes the fourth order statistic, kurtosis, rather than variance as in PCA. Kurtosis represents the skewness of a dataset, and anharmonic data is, by definition, skewed. Thus, JADE is a better suited algorithm for dimensionality reduction than PCA. Even after reducing the dimensionality with JADE, frames belonging to the same conformational grouping may separate and prevent accurate clustering. When represented in the full 40-60 dimensional ICA space, connected nodes may break connection by separating too far, which again requires a dimensionality limitation.

To limit the direct comparison of frames, or nodes, in constructing the affinity matrix, k-nearest neighbors is employed. k-nearest neighbors is a search technique which finds the k, a user input, nearest neighbors to a specific point, provided a distance metric. It relies on a strict data structure to perform efficiently, e.g. using a k-dimensional tree (k-d tree). A k-d tree is a specific type of binary search tree which stores k-dimensional data points, implicitly splitting the data equally with hyperplanes along subsequent axes in order to create a balanced tree. The second dimensionality limitation mentioned above prevents the nodes stored in the tree, or leaves and branches, from separating too far [12].

3 Methods

3.1 Design and Implementation

The standard for analysis of protein folding simulations is QAA because of the fundamentals it provides. However, it wasn't built to scale to large datasets, or follow through with subsequent pipeline analyses. I developed dQCuts, a follow-up to QAA, which performs automatic, out-of-core identification of protein folding states. Figure 4 presents a visual flow of the dQCuts program, with textual explanations referring to it in this section.

dQCuts is written in Python, heavily supported by the NumPy, Matplotlib, and SciPy libraries [13, 14, 15]. A small part of the clustering code is written in C++ and linked to Python using f2py [16].

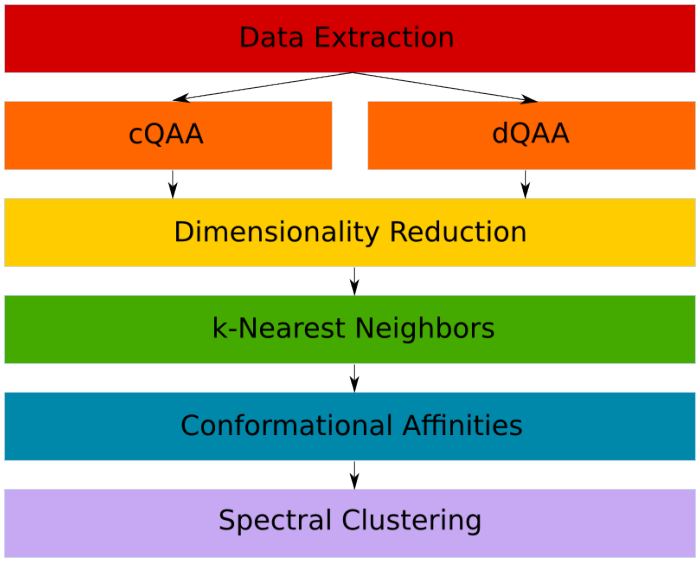

Figure 4: This flowchart details the computational flow of dQCuts.

dQCuts and QAA are designed for the same purpose, so dQCuts is an expansion of QAA in various ways. Firstly, dQCuts improves coordinate based as well as adding dihedral (angle) based analysis. While coordinate based is simpler to visualize because it works in the cartesian space, monitoring the statistics of the backbone dihedral angles provides more information on each residue and subsequently each conformation, and doesn't require before dimensionality reduction. dQCuts also inputs nearly limitless trajectories and retains the data on disk rather than in memory, allowing for a more complete analysis of a simulation. Finally, dQCuts automatically clusters the anharmonic nodes through image processing/clustering techniques.

The software is composed of three parts. The first is an implementation of QAA of the form coordinate-QAA (cQAA) or dihedral-QAA (dQAA). Both are implemented to receive input from plural trajectories and avoid memory issues in cQAA and dQAA by storing only a single trajectory in RAM at a time by utilizing NumPy's memmapping features to concatenate contiguous data into a disk stored array. The red in Figure 4 represents this data extraction.

cQAA: Coordinate-based QAA performs its analysis in the $\mathbb{R}^3$ Cartesian space, and observes the movement of the C_α atoms in the protein backbone. For each frame in a given trajectory, the Cartesian coordinates of the C_α atom in each of the N residues are recorded to create a $3N$-dimensional vector representing the molecular conformation of the frame. For each trajectory, the data is extracted in this vectorized form and aligned upon the trajectory's mean structure.

dQAA: The sines and cosines of both the Φ and Ψ angles as defined in Figure 3 yields four scalar data points on each residue. For each frame in a trajectory, a $4N$-dimensional vector that represents the conformation is constructed.

While cQAA and dQAA run independently, as seen in Figure 4, their computational workflows are nearly identical after the distinction in data extraction. The data is concatenated to a memory mapped array in order to avoid memory errors. In the case of cQAA, on the final trajectory the entire dataset is aligned upon the mean structure. The result is a data matrix of the size $[3-4N, Ns]$, where $Ns = NumberOfSamples$. To put this in perspective, using double precision floating point storage many datasets claim over 2GB of RAM.

Dimensionality Reduction: With the data stored on disk, the dimensionality is reduced before clustering using a type of ICA, JADE, which is shown in Yellow in Figure 4. JADE, which begins with a specified subset of Principal Components and forms anharmonic basis vectors which maximize the skewness of the data. Thus, the data is represented in a lower dimensional landscape while retaining significant variance and not sacrificing anharmonicity.

Spectral Clustering: The clusters of the simulation are contained in a intra-connected graph of each nodes' closest neighbors. A fast search method, k-nearest neighbors, first builds an n-dimensional tree and can be queried for a particular node's neighbors. Sci-Kit Learn provides a fast k-nearest neighbors algorithm used to limit affinity calculation to only the closest nodes [17]. The search, k-nearest neighbors prepares a dataset to construct an affinity matrix as shown in Figure 4.

Although the dimensionality is reduced with JADE, the dimensionality chosen to construct the n-dimensional tree must be limited slightly further, depending on the data. Using the newly truncated dimensional data to find the *k*-nearest neighbors and generate a similarity graph, clustering is done by solving:

$$A * x_i = \lambda_i * x_i, i = 1, \ldots, n \tag{1}$$

where A represents the similarity graph, n is the desired number of clusters, and x_i and λ_i are an eigenpair where x is the eigenvector and λ the eigenvalue. The computation to solve the eigenproblem is non-trivial and computationally difficult. Because the simulation represents an image, techniques in image clustering are applied to the higly dimensional protein data. DNCuts, an algorithm spun off of Shi and Malik's NCuts, intelligently downsamples the simulations similarity matrix, computes the clusters

in the subspace, and upsamples to return approximate clusters in a fraction of the time [18, 19]. The method performs a downsampling while effectively "squaring" the similarity matrix, where in order to avoid significant memory issues in matrix multiplication, an original subroutine is employed to perform blockwise matrix multiplication while concurrently truncating values below a threshold, and temporarily storing the sparse matrix on disk. The threshold is set below 1% of the mean of original data and is necessary because of the drastic density increase in sparse matrix multiplication. The in-method memory-mapping limits RAM use to a single submatrix, and allows simple reassembly post-multiplication.

The clusters are discovered by collecting the indices that lie outside two standard deviations of the mean value in each eigenvector corresponding to the largest eigenvalues. All the code and example datasets are accessible on github [20].

3.2 Experimental Parameters

dQCuts has been tested with two protein simulations, each for a distinct purpose. The first is a 12.5 μstrajectory of a pentapeptide, with only five amino acids in each conformation. It provides a proof-of-concept for our methods by showing dQCuts can isolate distint conformational groupings automatically.

Out of the full 125,000 frame pentapeptide simulation, 62,500 frames were used in this analysis. While the $5 aminoacids * 3 cartestiandimensions = 15$ dimensional space is manageable for clustering and post-processing, 100% of the variance is preserved in only the first eight independent components. Thus, the dataset is halved by reducing the space to only eight dimensions with 62,500 frames. The dimensionality in which to restrict the k-nearest neighbors was chosen to be four dimensions in order to prevent separation of clustered points. Sixteen eigenvectors were calculated, which result in several identified states.

4 Results

cQAA:

After alignment of the 25 trajectories to minimize local and global Root Mean Squared Deviation (RMSD) of structures, the cumulative sum of eigenvalues of the pentapeptide dataset's covariance matrix,

shown in Figure 5, presents the percent of variance preserved when restricting the dimensionality. The plot shows 100% of the variance preserved with only eight principal components retained.This leads us to the data projected onto the lower dimensional anharmonic basis vectors, with the first three dimensions seen in Figure 6.

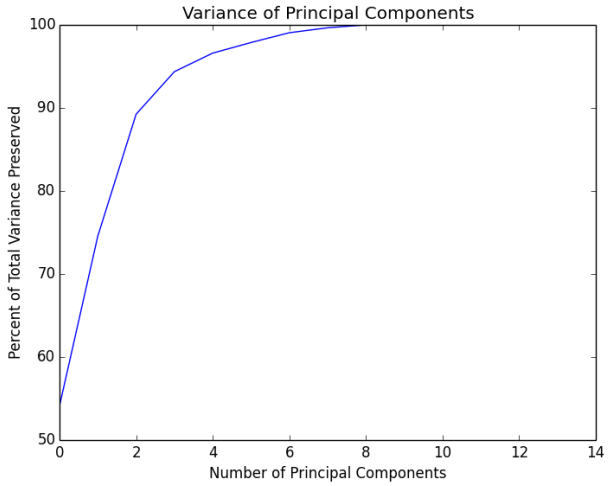

Figure 5: The graph shows the cumulative sum of the eigenvalues of the co-variance matrix presents the percentage of the total variance preserved.

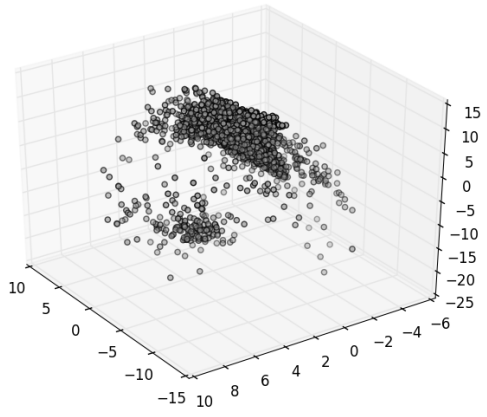

Figure 6: In this three dimensional trace of high dimensional data, each point represents a conformational structure of the pentapeptide after projection on anharmonic basis vectors.

To correctly identify the k-nearest neighbors, the search is restricted to a lower dimensional landscape which prevents cluster seperation. Because the pentapeptide is a small example, the plot shown in Figure

7 is rather simple, but there is a slight concave down knee point at four dimensions which was used to perform the search.

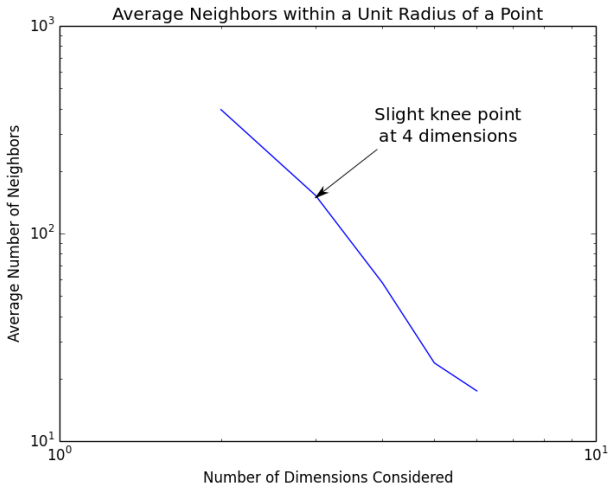

Figure 7: This Log-Log plot identifies the average number of neighbors within a unit radius of a point.

The leads to the identified clusters of pentapeptide, in which all 16 clusters are presented in Figure 8. The plots are of the average cartesian coordinates of the C_α atoms of the conformations belonging to a cluster. Three distinct states are seen in Figures 9, 10, and 11. The clusters belonging to the first state in Figure 9 are clusters 0-2, 5, 6, and 8-13. Clusters 3 and 15 form the second state as seen in Figure 10. The third state is comprised of clusters 4, 7, and 14, shown in Figure 11. Figure 12 shows this quantitatively by showing the similarity of the mean structure vectors. Each entry $a_{i,j} = ||c_i - c_j||_2$, for $i, j = 0, \ldots, 16$ where c_i and c_j are the i^{th} and j^{th} cluster's mean coordinate vector, respectively. Figure 12 is colored so as to reflect the differences of structures, with the attached colorbar dictating the distances of structural vectors.

The Kabsch algorithm is applied internally to the clusters by aligning each structure of a cluster member onto the mean structure of the cluster to understand how self-similar a cluster is. A low RMSD value means very similar structures, while a high value translates to different structures. Table 1 presents these values. State 1 clusters have significantly lower RMSD values.

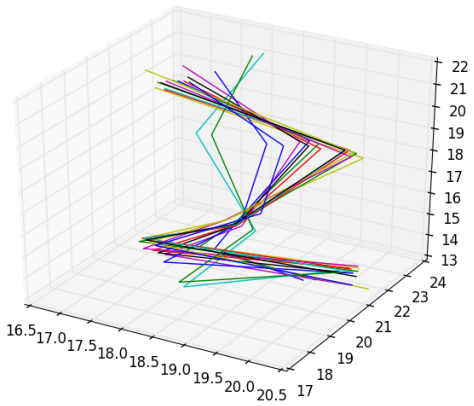

Figure 8: All clusters of the pentapeptide discovered through cQAA are plotted on Cartesian coordinates.

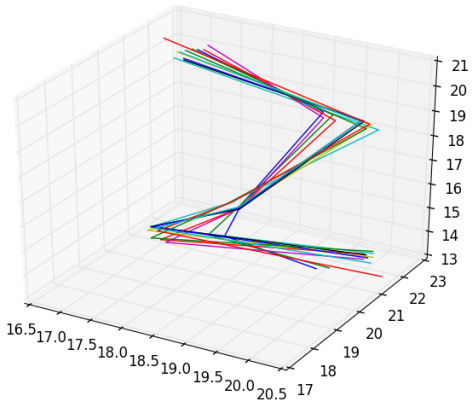

Figure 9: The Cartesian space shows the Pentapeptide clusters belonging to State 1.

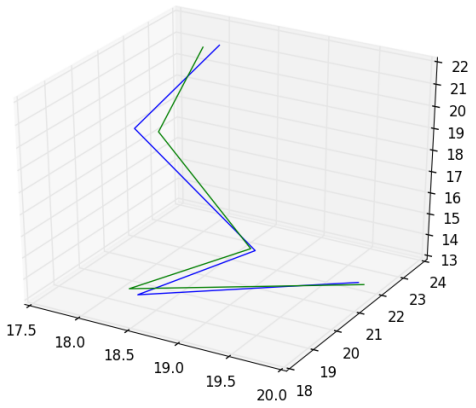

Figure 10: The Cartesian space shows the Pentapeptide clusters belonging to State 2.

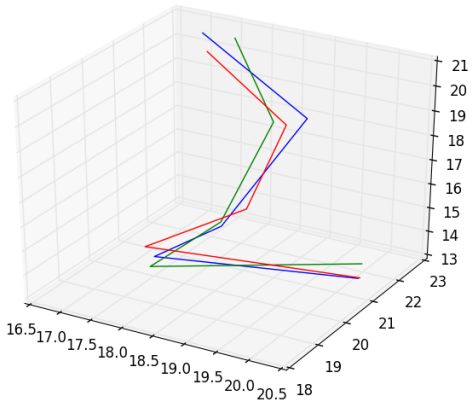

Figure 11: The Cartesian space shows the Pentapeptide clusters belonging to State 3.

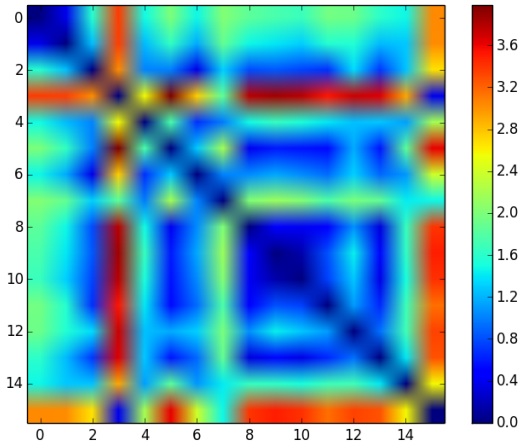

Figure 12: A Similarity Matrix of the Pentapeptide's cluster's mean structures shows the difference between clusters.

Table 1: Mean and Median internal RMSD values of each cluster of Pentapeptide

Cluster #	Mean RMSD	Median RMSD
0	0.8644	0.7015
1	0.7875	0.6059
2	0.7113	0.5560
3	0.5198	0.3850
4	0.8699	0.7486
5	0.3293	0.3051
6	0.8774	0.7605
7	0.9805	1.0001
8	0.2897	0.2770
9	0.3778	0.2779
10	0.4568	0.3482
11	0.3878	0.2698
12	0.3438	0.3084
13	0.5008	0.4097
14	1.0498	0.9251
15	0.6743	0.4962

Table 2: Average mean and median RMSD values of States 1, 2, and 3

State #	Mean RMSD	Median RMSD
1	0.5388	0.4382
2	0.5970	0.4406
3	0.9667	0.8913

5 Discussion

While the Pentapeptide is a small and simple example, Figure 5 presents the cumulative variance after retaining the n most significant Principal Components. dQCuts, at the core, is a statistical analysis of the multidimensional data achieved by comparing frames in simulation. It is seeking to understand how the data varies through frames and structures, in order to isolate groups which exhibit similar structures. Thus, the preservation of 100% of the variance, but trimming the data to eight components is keeping 100% of the gold when panning for gold, but dumping nearly all of the extra debris.

After projecting the full dataset onto the newly computed Independent Components, Figure 6 shows the first three dimensions of the anharmonic dataset. In this figure two clear clusters or groups have separated completely in the first three dimensions. Note the group of points gathering towards the bottom of Figure 6 and the even larger grouping on top. Since three independent states are isolated, the three dimensional trace of the eight dimensional dataset likely forces a third separate grouping to group into the larger first state.

Again, because the Pentapeptide is such a small example, Figure 7 is overly simple, loosely representing a line. However, it is still a significant plot because it provides a quantitative measure for the separation of clusters in high dimensions. In order to force clustering, a dimensionality restriction for the kd-tree and affinity generation is chosen that corresponds to a high ($> 10^2$) average number neighbors in a unit radius.

Figures 8, 9, 10, and 11 all present the mean structures of the automatically identified clusters. Because of the simplicity of the Pentapeptide backbone, three states clearly form in Figure 8. Visually, their structures each have unique features that allow each state to set itself apart, as seen in Figures 9, 10, and 11. Quantitatively, this is supported in Figure 12, as the distances are low intra-state and significantly higher inter-state. Each entry $a_{i,j}$ in the plotted matrix, or Figure 12 represents the distance between the mean structure of the i^{th} and j^{th} clusters.

Another important metric to measure, is the internal dynamism of each of the clusters and states. This is measured by aligning each frame of a cluster to the mean structure of the cluster and averaging the post-alignment RMSD of each frame. The structures of an individual cluster should nearly perfectly superimpose each other, leading to diminishing RMSDs. Note that all values in Tables 1 and 2 are quite small, but there appears to be significantly higher values in state 3, corresponding to a higher degree of dynamism inside a cluster, or a less stable structure. As expected, state 1 yields the lowest values, identifying

it as the most stable folding state, otherwise known as the native state.

6 Conclusions

New software was needed in order to simplify long MD simulations and recover the functional folds of a protein. The groundwork for dQCuts was laid by QAA, which uses Blind Source Signal Separation to group of conformational structures. However dQCuts more effectively meets the researcher's demand for analytic software with a more developed coordinate analysis and more complete computational flow. The out-of-core memory storage combined with a powerful image clustering algorithm proves dQCuts to be a versatile and useful analysis. The results from the Pentapeptide are encouraging, as they show that dQCuts solves a difficult problem in protein modeling, and could be applied to larger and more interesting proteins. The three states identified have good internal qualities, with low RMSD values, and clearly separate themselves when compared.

7 Future Work

While dQCuts is the culmination of two years of work, several avenues for more developments and further research have opened. Short term goals for increased software speed or decreased memory usage for example, but state-switching prediction and ultimately understanding cellular decisions on a macromolecule level is at the edge of what is possible.

With a proven and functional method, improvements and optimization techniques are necessary, to provide the best software possible to researchers. One of the biggest subsets of optimization is speed enhancements of relatively slow code. This was introduced in the implementation of a Python DNCuts with a matrix normalization subroutine. Vectorization with NumPy wasn't possible, and Python loops are very slow compared to compiled C software. Loops in C are demonstrably orders of magnitude faster than Python, and C can be wrapped in Fortran, precompiled, and linked to Python with relative ease [16]. Several functions would receive significant speed benefits through these techniques because the limiting factor to these methods is simply how fast they can iterate. Parallelization would be another significant improvement in specific areas of the software, as several methods or algorithms such as the initial data

extraction, affinity matrix generation, and sparse blockwise matrix multiplication are at least partially perfectly parallel. Although the large MD data is disk stored through the software, direct computation on matrices must be done in RAM, which can lead to memory issues, especially in methods provided through external packages, such as ARPACK's eigensolving included in SciPy. The solution to this isn't clear, as it will force a rewrite of SciPy's library using memory-mapped arrays (laborious but likely trivial) or perform all computation using the constituent matrices of a sparse matrix (laborious and non-trivial). Another alternative is to expand the software to distributed datasets, as seen in Spark, which does have an API for Python. Obstacles along this route exists in writing software to perform the Linear Algebra needed accross the datasets.

Besides the software improvements in speed or optimization which are clearly possible, developing new techniques for the end of the pipeline such as analyses on the identified clusters is an important next step. Providing researchers with simple and accurate predictions on how and when proteins will switch between identified states in models allows understanding of the motivations of certain folds on a molecular level. The ultimate goal with prediciting these folding patterns is the monitoring of actual cellular proteins, leading to the prediction of cellular activity, and thus molecular level diagnostic tools in the medical field. Such a tool that could monitor and then proceed to influence proteins would reinvent modern medicine, and the socioeconomic faculties associated. Such a tool would also require leaps and bounds of progress in nanotechnology.

References

[1] Dave Lawson. *A Brief Introduction to Protein Crystallography*. URL: https://www.jic.ac.uk/staff/david-lawson/xtallog/summary.htm.

[2] Peter Freddolino. *MD Simulation of Protein Folding*. May 2015. URL: http://www.ks.uiuc.edu/Research/folding/.

[3] Gamini Gunawardena. *Peptide*. Online.

[4] Jim Clark. *The Structure of Proteins*. 2012. URL: http://www.chemguide.co.uk/organicprops/aminoacids/proteinstruct.html.

[5] Naveen Michaud-Agrawal et al. "MDAnalysis: A toolkit for the analysis of molecular dynamics simulations". In: *Journal of Computational Chemistry* 32.10 (2011), pp. 2319–2327. ISSN: 1096-987X. DOI: 10.1002/jcc.21787. URL: http://dx.doi.org/10.1002/jcc.21787.

[6] David A. Case et al. *Amber 11*. University of California, San Francisco.

[7] H. J. C. Berendsen, D. Van Der Spoel, and R. Van Drunen. "Gromacs: A message-passing parallel molecular dynamics implementation". In: *Comp. Phys. Comm* 91 (1995), pp. 43–56.

[8] B. R. Brooks et al. "CHARMM: The biomolecular simulation program". In: *Journal of Computational Chemistry* 30.10 (2009), pp. 1545–1614. ISSN: 1096-987X. DOI: 10.1002/jcc.21287. URL: http://dx.doi.org/10.1002/jcc.21287.

[9] W. Kabsch. "A solution for the best rotation to relate two sets of vectors". In: *Acta Crystallographica Section A* 32.5 (Sept. 1976), pp. 922–923. DOI: 10.1107/S0567739476001873. URL: http://dx.doi.org/10.1107/S0567739476001873.

[10] Arvind Ramanathan et al. "Discovering Conformational Sub-States Relevant to Protein Function". In: *PLoS ONE* 6.1 (Jan. 2011), e15827. DOI: 10.1371/journal.pone.0015827. URL: http://dx.doi.org/10.1371/journal.pone.0015827.

[11] Jean-François Cardoso and Antoine Souloumiac. "Jacobi angles for simultaneous diagonalization". In: *SIAM J. Mat. Anal. Appl.* 17.1 (Jan. 1996), pp. 161–164.

[12] Jon Louis Bentley. "Multidimensional Binary Search Trees Used for Associative Searching". In: *Commun. ACM* 18.9 (Sept. 1975), pp. 509–517. ISSN: 0001-0782. DOI: 10.1145/361002.361007. URL: http://doi.acm.org/10.1145/361002.361007.

[13] S. van der Walt, S.C. Colbert, and G. Varoquaux. "The NumPy Array: A Structure for Efficient Numerical Computation". In: *Computing in Science Engineering* 13.2 (Mar. 2011), pp. 22–30. ISSN: 1521-9615. DOI: 10.1109/MCSE.2011.37.

[14] J.D. Hunter. "Matplotlib: A 2D Graphics Environment". In: *Computing in Science Engineering* 9.3 (May 2007), pp. 90–95. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.55.

[15] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. [Online; accessed 2015-09-23]. 2001–. URL: http://www.scipy.org/.

[16] Bogdan Vacaliuc et al. “Python for Development of OpenMP and CUDA Kernels for Multidimensional Data”. In: *Application Accelerators in High-Performance Computing, Symposium on* (2011), pp. 159–167. DOI: http://doi.ieeecomputersociety.org/10.1109/SAAHPC.2011.26.

[17] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[18] P. Arbeláez et al. “Multiscale Combinatorial Grouping”. In: *Computer Vision and Pattern Recognition*. 2014.

[19] J. Shi and J. Malik. “Normalized cuts and image segmentation”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22.8 (Aug. 2000), pp. 888–905. ISSN: 0162-8828. DOI: 10.1109/34.868688.

[20] Gabriel Vacaliuc. *GitHub*. 2016. URL: http://www.github.com/gvacaliuc.