Pedro Miguel Franco Arguelles

Bachelor Degree in Science and Computer Engineering

Clustering of Protein Structures

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Engineering

Adviser: Ludwig Krippahl, Assistant Professor, NOVA University of Lisbon

Examination Committee

Chair: Prof. Dr. Carmen Pires Morgado
Rapporteur: Prof. Dr. Sofia Rocha Pauleta

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

May, 2021

Clustering of protein structures

Acknowledgements

First of all, I would like to thank the teachers and colleagues in the FCT-UNL Computer Science department that accompanied me during the course of my Master degree, as they were essential in pushing me forward through all the challenges that I faced during this time.

I would also like to give a special thanks to my advisor, professor Ludwig Krippahl, for supervising, guiding my work and having the patience to spend many hours sharing his knowledge with me.

Finally, to all of my closest friends and family, especially my parents, who supported me either morally, financially or with their knowledge, thank you.

Abstract

Proteins are very complex and important molecules that carry out a wide range of functions essential to life. The role of any given protein within a cell is heavily determined by its structure, which is recognized as a valuable resource of information when studying proteins. In applications such protein docking or phylogenetics, there is the need to compare such structures in order to obtain relevant information about the proteins that are being considered. As such, there is also a need to specify a some kind of measure that is able to indicate if two protein structures are similar or not. Currently, there are a few different measures that can be used for this task, however, due to the inherent complexity of protein structures it is very hard for a measure to take into account the numerous possible variations and perfectly quantify the dissimilarity among them.

Considering this issue, with this work we aim to use clustering algorithms and experiment with different structure similarity measures, in an attempt to find effective ways of grouping protein structures with the goal of obtain useful information that can be used in the previously mentioned applications.

Keywords: Proteins, Machine learning, Unsupervised learning, Clustering, Protein structures, Structural similarity measures

Resumo

As proteínas são moléculas de alta importância e complexidade que desempenham uma grande diversidade de funções essenciais à vida. O papel de uma dada proteína dentro de uma célula é fortemente influenciado pela sua estrutura, que é reconhecida como um valioso recurso de informação no estudo de proteínas. Em aplicações como o *docking* ou a análise filogenética de proteínas, há uma necessidade de comparar tais estruturas de forma a obter informação relevante sobre as proteínas que estamos a considerar. Como tal, também há a necessidade de especificar uma medida que seja capaz de indicar se duas estruturas de proteínas são ou não semelhantes. Atualmente, há medidas diferentes que podem ser usadas para esta tarefa, no entanto, devido à inerente complexidade das estruturas de proteínas é muito difícil para uma medida ter em conta as numerosas variações possíveis e quantificar perfeitamente as diferenças entre elas.

Tendo estes problemas em consideração, neste trabalho vamos usar algoritmos de *clustering* e experimentar diferentes medidas de semelhança, numa tentativa de encontrar maneiras efetivas de agrupar estruturas de proteínas com o objectivo de obter informação útil, que possa ser usada nas aplicações mencionadas anteriormente.

Palavras-chave: Proteínas, Aprendizagem automática, Aprendizagem não-supervisionada, Clustering, Medidas de semelhança estrutural

Contents

List of Figures

List of Tables

Glossary

backbone	Refers to the chain of amino acids that form the protein
clustering	Machine learning techniques that are used to split the data into groups of similar elements.
hyperparameters	Parameters that control the learning process.
optimizer	The Bayesian optimizer used throughout the experimental phase in order to estimate the hyperparameters.

Acronyms

AFP	Aligned Fragment Pairs
AMI	Adjusted Mutual Information
BIRCH	Balanced Iterative Reducing and Clustering using Hierarchies
CASP	Critical Assessment of protein Structure Prediction
CATH	Class, Arquitecture, Topology and Homologue superfamily protein structure classification database
CE	Combinatorial Extension
CV	Cross Validation
DBSCAN	Density-based spatial clustering of applications with noise
GDT-HA	Global Distance Test - High Accuracy
GDT-TS	Global Distance Test - Total Score
GMM	Gaussian Mixture Models
LOOCV	Leave One Out Cross Validation
MaxSub	Maximum Subset
PDB	Protein Data Bank
RMSD	Root Mean Square Deviation
SCOP	Structural Classification of Proteins database
TM-score	Template Modeling score

Symbols

Å Ångström

Chapter 1

Introduction

Proteins are large molecules with complex structures which carry out a wide range of functions in organisms. They are essential to life due to their versatility since they can act as antibodies, to help combat viruses and bacteria, they can take the enzymatic role in order to increase the rate of chemical reactions in the body, they can aid in hormone creation, in the transport of small molecules, etc.

There are a few factors which determine the function of a given protein, namely its amino acid sequence and spatial conformation. Despite this, it is known that as time passes sequences undergo mutations to its amino acids. As such, if enough time goes by, a given sequence may become unrecognizable when comparing it to what it used to be. Luckily, structures are not affected as heavily as sequences, which means that they tend to be much more conserved during a protein's evolution to the extent that they are a better tool for understanding their functionality, interactions and relationships.

In applications such as the ones mentioned ahead [29], the value of protein structures is recognized and used to obtain insightful information:

- **Evolutionary analysis**

  It is possible to identify common ancestors using both sequence and structural comparisons. If the proteins to compare have a relatively short evolutionary distance between each other, usually it is enough to compare the amino acid chains of both proteins to establish an ancestry relationship. The chains should be similar due to the short amount of time that has passed, which means the amino acid sequence shouldn't be significantly altered. However, this method works under the assumption that the changes to the sequences are minimal, which most likely will not happen if the evolutionary distances are greater. In this case, the most appropriate method is comparing the proteins by their three-dimensional structure which is better preserved than sequence. This approach may identify similarities among

proteins because even if the amino acid sequence is changed as time passes, there will still be some elements with a layout similar to a previous state of the protein. In practice however, most cases are more complicated and thus, require more complex approaches which use combinations of both types of comparisons [5] [20].

- **Docking**

  Another application where it is necessary to group and compare structures is protein docking [17]. This is an expression that is used to describe computational methods that output predictions on how two proteins - a receptor and a ligand - interact in order to form a molecular complex. The details of these processes are beyond the scope of this work, but in short, protein docking is comprised of two stages: search and scoring. The first stage is where we search through every spatial arrangement of both molecules so we can obtain a set of those that might resemble the true conformation of the complex. The scoring stage is where the generated set is analyzed and ranked according to some function. Generated predictions can also compared to measure their quality. Since both the predicted and target complexes possess the same sequences, their sequence alignments are easy to perform and should clearly identify the matching residues between the two complexes. On the other hand however, structure comparisons pose a more difficult problem as a consequence of the docking process providing us with several complexes that differ in atom position, contact regions or probe orientation for instance.

- **Predicting unknown functions**

  There are cases in which a newly discovered protein has an unknown function. When this situation arises, we may try to infer it from other known ones. In other words, when we know the three-dimensional structure of a protein but not its function, we can compare that structure to other proteins whose functions are known in order to find the most similar pairs. If there are pairs with significant structural similarities, then we can make a well informed prediction that the unknown function is the same as the known one.

1.1 Challenges and objectives

As we can see, the previous applications require that comparisons are made between protein structures in order to find groups of them that contain relevant information. Despite the advantages of comparing protein structures, doing so is not an easy task since structures are inherently complex and the differences among them are not uniform. Due to this complexity, it is difficult to devise a perfect method to represent similarity that is able to account for variations in atom positions, residue orientation, local mismatches and long sequence lengths. Currently there are several methods of measuring similarities,

some of which will be discussed in this document, and these can differ in aspects such as the choice of atoms, used distance metrics and type of result.

One of the most common measures is the Root Mean Square Deviation (RMSD), which is essentially the averaged distance between all pairs of matched residues. As a consequence of being an average, RMSD comes with some shortcomings, namely its inability to account for local variations and greater sequence lengths, which can have a negative impact in its calculation and in turn, overstate the dissimilarity of the structures [30]. This was just an example, but we can see that if we are relying on a specific measure to establish similarities and find groups within a protein structure dataset, it is very likely that the weaknesses and strengths of that measure directly impact the quality of groups formed.

Furthermore, once again due to structure complexity, some of the resources in this field rely on manual classification of structures while grouping them. For instance, CATH uses automated methods to classify proteins in the different levels of its hierarchy. In most classifications this automation is enough, however there are cases in which they are unable to identify the correct classification for a protein and as a consequence this task must be performed manually [26].

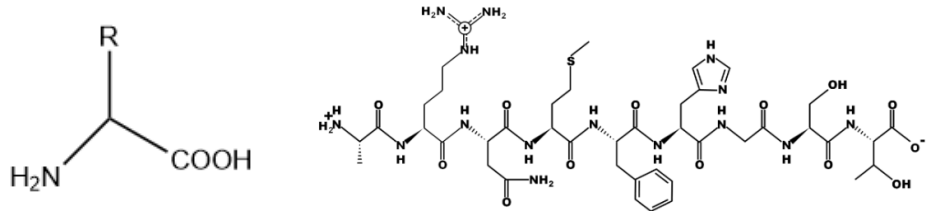
In short, given the challenges and usefulness of comparing and grouping protein structures, during the course of this work, a study was made using different combinations of clustering algorithms and protein similarity measures, in order to obtain a method that generally works well when clustering this type of data. Furthermore, the possibility of improving the quality of the clusters by combining different similarity measures or by applying non linear transformations was also explored.

1.2 The protein

The following subsections serve as a brief introductions to some of the core concepts of proteins as well as their composition.

1.2.1 Amino acids

Proteins are formed by polypeptides, which are in turn amino acid chains. These have three main components: an amine group (-NH2), a carboxyl group (-COOH) and a side chain connected to the alpha carbon (central carbon of an amino acid). This chain is what differentiates and identifies the 20 existing amino acids, due to being the only element which varies among them. Considering the described format, peptides have two terminal zones in their chains, one with the amine group and the other with the carboxyl group, also known as N-terminal and C-terminal respectively. Thus, all protein molecules are polymers assembled from combinations of 20 different amino acids connected through peptide bonds [4]. In Figure 1.1 we can see an example of a basic structure of an amino acid and several of them connected forming a chain.



(a) Basic amino acid structure (b) Peptide chain example

Figure 1.1: The amino acid

1.2.2 Structure

Proteins are very complex molecules whose function is determined by its structure. In order to help understand it, we generally describe protein structure in four levels:

- Primary structure: linear amino acid sequence with peptide bonds;
- Secondary structure: local folded structures, such as the α-helix and the β-sheet;
- Tertiary structure: the three-dimensional structure of the protein;
- Quaternary structure: some proteins are composed by several polypeptide chains, known as subunits. The interaction between subunits gives the protein its quaternary structure.

It is worth mentioning that primary structures, i.e. sequences are more volatile during the course of its evolution since they must adapt in order for them to continue carrying on their roles, whereas the tertiary structure is better conserved throughout these processes. This makes it so that structural conformation has an increased relevancy when it comes to analyzing proteins.

Within proteins, we can also find domains which are considered to be independent regions in the proteins, often viewed as compact and spatially distinct units with functionalities that usually help the overall protein carry out its function. It is important to mention that similar domains can be found in proteins with different functionalities [43].

1.2.3 Homology

Proteins change in order to continue carrying on their functions, which means that the ones present in organisms nowadays are the result of a long and continuous evolutionary process. Through this we can define the term homology, which in the protein context, means that two proteins share a common ancestor. Finding homology is useful because we may be able to infer unknown information of a given protein through an homologous one.

In order to determine if two proteins are homologous or not, we can analyze both their primary and tertiary structures in an attempt to find corresponding residues. Usually,

we can start by checking the primary structures to see if there are identical amino acid residues in a significant number of sequential positions throughout the amino acid chain. Generally, to establish sequence homology, around 30% of sequence identity must be found. If this percentage falls below this value we are required to further analyze the tertiary structures which are better preserved than sequences [40].

The concept of homology is a very broad one, which refers to the different evolutionary relationships among proteins. However, different types of homology are encapsulated within the term:

- Orthology: proteins that evolved from a common ancestral protein through a speti-ation event. They are essentially the same protein, only in different species.
- Paralogy: proteins that are related through a common ancestor through duplication events. Usually, these are proteins that evolved from a common ancestor to have different functions.
- Xenology: describes the relationship of two given proteins whose history involves at least one occurrence of horizontal transfer of at least one of them.

These are some of the main categories of homology, however, there are a few others, namely: ohnology, homoeology and gametology.

Chapter 2

State of the art

In this chapter we will briefly describe some of the available methods to measure protein structure similarity and provide the state of the art regarding clustering and protein alignment algorithms.

2.1 Protein comparisons

As previously mentioned, there are several uses for protein comparisons and these are made through either structural or sequential alignments. The former focuses on tertiary structures and the latter on primary structures. Despite these differences, their goals are very similar, which is to provide equivalences between residues so we can measure something that can be used to assess if two proteins are similar or not.

There are various scenarios we can come across while comparing two given proteins:

- The proteins are similar in sequence and in structure: this suggests that the two proteins are homologous. Since they are similar both in sequence and structure, we can infer that they share a common ancestor and that they have the same functionalities. This scenario usually occurs when we try to group proteins with low evolutionary distances for instance.

- The proteins are not similar both in sequence and structure: this case is most common when grouping proteins with high evolutionary distances. However, since there is nothing similar, it is hard to find any corresponding regions, which will result in the comparison between these two to generate vastly different similarity measures.

- The proteins are similar in sequence but not in structure: when studying structure prediction algorithms (docking) we may face this situation, specifically when the

predicted complex has a different structure from the target.

- The proteins are not similar in sequence but are in structure: once again, this case may appear when grouping proteins with higher evolutionary distances. As time passes, sequences become more prone to mutations whereas structure tends to be preserved.

Furthermore, the use of protein comparison techniques is subject to the available information about the proteins we want to compare. We can expect two different scenarios for this work. In the first one, we have information regarding both sequence and structure, which means we can start by aligning sequences in order to find correspondences between the amino acids and only then do we proceed to the structural alignment. The second case, is when both sequences are different to the point that an alignment between them is not possible. In this case, the structural alignment is the main resource of information.

In the subsections ahead we will se how these alignments through sequence and structure work, as well as provide examples of a few algorithms.

2.1.1 Sequence alignment

To reiterate, sequence alignment is the process of arranging protein chains and looking at the present amino acids in order to identify common regions or similarities among proteins. Usually, these processes provide us with a set of matches between the amino acids from two given proteins, as can be seen in Figure 2.1.

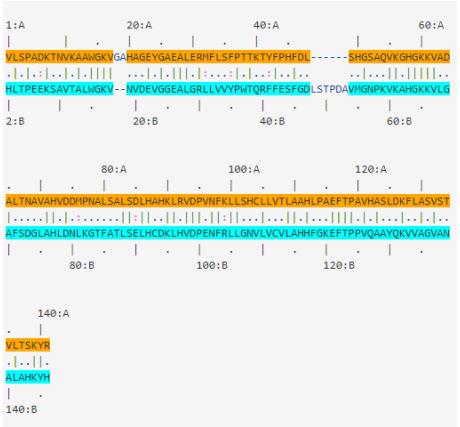


Figure 2.1: Example of a sequence alignment between 4HHB.A and 4HHB.B

There are two types of sequence alignments: global and local. In the first type of alignment, the goal is to find the best score from alignments of entire lengths of sequences. It is most common and best used when comparing full sequences of similar lengths. The second type, local, seeks the best score from partial sequences and works best when we want to compare a shorter sequence to a larger one or a partial sequence to a whole sequence.

As an example of how sequence alignment works, let us focus on the first ones to be introduced for local and global alignment. These have served as the base for others and are still used to this day to align sequences. To describe them, let us consider two protein sequences $A = a_1, ..., a_n$ and $B = b_1, ..., b_m$ and a substitution matrix $s(a_n, b_m)$ which provides scores for comparisons between residues a_n and b_m. These matrices can be simple ones, which attribute 1 or -1 scores in cases of matches or mismatches, or, we can use matrices that have been constructed based on statistical studies to be applied to particular scenarios. The most common ones being PAM (Point Accepted Mutation) [8] and BLOSUM (Blocks Substitution Matrix) [18]. Gaps of length k and l are given weight W_k and W_l, respectively. In order to find pairwise similarities, a scoring matrix H is used.

2.1.1.1 Needleman-Wunsch

For global alignment, the Needleman-Wunsch [36] algorithm is used.

1. **Initialization:** set the values of the first row and column of H according to the chosen gap penalty.
2. **Matrix filling:**

$$H_{i,j} = max \begin{cases} H_{i-1,j-1} + s(a_i, b_j), \\ H_{i,j-1} - W, \\ H_{i-1,j} - W \end{cases}$$

3. **Traceback:** starting at the bottom right corner of the matrix, we wish to make our way into the top left corner while maximizing the score.

2.1.1.2 Smith-Waterman

For local alignment, the Smith-Waterman [51] algorithm is used.

1. **Initialization:** set the values of the first row and column of H to zero.
2. **Matrix filling:**

$$H_{i,j} = max \begin{cases} H_{i-1,j-1} + s(a_i, b_j), \\ max_{k \geq 1} \left\{ H_{i-k,j} - W_k \right\}, \\ max_{k \geq 1} \left\{ H_{i,j-l} - W_l \right\}, \\ 0 \end{cases}$$

3. **Traceback:** starting at the highest score position of matrix H, we wish to find the path that maximizes the score and finishes in a position with value zero.

With this expression we cover the cases in which a_i and b_j are associated, a_i or b_j are at the end of deletions of length k or l respectively, and it also prevents the calculation of negative similarity by including a zero in such cases.

2.1.2 Structure alignment

When comparing two given protein structures, we can usually identify three main stages. Firstly, we search both protein structures in an attempt to detect similarities among them. The second stage, consists of aligning the structures based on the found similarities. Essentially, the structure alignment process corresponds to finding the parts of one protein that best match on the other one. Lastly, the third stage is all about evaluating the alignment process, usually through some metric or score that allows us to conclude if the proteins are similar or not. The figure below (2.2) shows us an example of the result of a structural alignment.

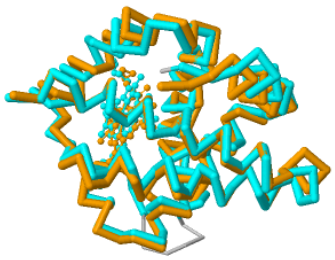


Figure 2.2: Example of a structure alignment between 4HHB.A and 4HHB.B

To date, there are many comparison methods that use information regarding the 3-D conformation of proteins. We can split these methods in a few categories. There are approaches that break protein structures into smaller units and then analyze the relationships between these units to determine the similarity of two structures. Some examples that fit this category are: RAPIDO [33], VAST [16], MASS [10], SSM [28] and DALI [21]. Another possible approach that FAST [63] and SABERTOOTH [53] explore, is obtaining a structural alignment based on pairwise residue distances. Furthermore, there are also multiple structural alignment methods such as: CBA [12], POSA [58], MultiProt [48], MALECON [37] and MUSTANG [27].

In the following subsections, some of the algorithms that are implemented by the available tools will be briefly described in order to provide a better idea of how they work.

2.1.2.1 Superposition based on Kabsch's algorithm

This algorithm allows us to compare two proteins by directly superimposing the structures [25]. It uses linear and vector algebra in order to find the best rotation and translation of two structures while minimizing RMSD. It requires a previous alignment between proteins so that we are able to measure the distance for each matched pair. An implementation of this algorithm can be found in the PyMol software [9].

The steps of the algorithm have a fairly complex mathematical component, which is more thoroughly explained in [5], but its base steps, considering proteins P and Q, are:

1. Determine the subsequences of alpha carbons to be used in the 3-D alignment:

$$\mathrm{M(P)} = (p^{(\alpha_1)}, p^{(\alpha_2)}, ..., p^{(\alpha_N)})$$
$$\mathrm{M(Q)} = (q^{(\beta_1)}, q^{(\beta_2)}, ..., q^{(\beta_N)})$$

2. Calculate centroids $p^{(c)}$ and $q^{(c)}$. If the atoms in the alignment do not have the same atomic weight, we can use the center of mass.

3. Translate all the atoms to the origin of the coordinate system, so that the centroids of M(P) and M(Q) coincide. We are then working with $x^{(i)}$ and $y^{(i)}$ coordinate sets.

4. Calculate the covariance matrix C, given by:

$$C = \sum_{\gamma=1}^{N} y^{(\gamma)} x^{(\gamma)T}$$

   then proceed to compute its singular-value decomposition (SVD). This process allows us to express matrix C as a product of matrices, $C = USV^T$.

5. Compute the rotation matrix $R = UV^T$.

6. Verify if $det(R) = 1$. If this determinant is negative, then we must redefine the rotation matrix to be $R = Udiag(1, 1, -1)V^T$.

7. Apply the rotation matrix to the $x^{(i)}$ coordinates.

After we apply the rotation matrix to the $x^{(i)}$ coordinates we are able to determine the squared distance from each $x^{(i)}$ to its corresponding $y^{(i)}$ point. Following this, we can now perform some calculation using metrics and scores such as the RMSD.

Through this result, we can now evaluate how good the superposition is and in turn, if the two structures are similar or not.

2.1.2.2 Sequential Structure Alignment Program

The SSAP [38] algorithm proposes a different approach that does not require explicit superposition of the two structures. Instead, it focuses on the geometric relationships within proteins and uses them to make the necessary alignment. The concept of this algorithm is that it produces a structural alignment by constructing an alternative view for each of the protein structures, which is essentially the calculated inter-residue distance vectors between each residue. Once the vectors and their respective matrices are calculated, the algorithm calculates several new matrices that represent the differences between vectors. Then, it uses a dynamic programming approach on them to determine the optimal local alignments, which are then summed into yet another different matrix. Finally, by using dynamic programming again on the resulting matrix, we are given the overall alignment of the structures.

Currently, the CATH database [50] provides a tool for applying the SSAP algorithm to protein data.

A more detailed version of the SSAP algorithm steps is described below:

1. For the first step of the algorithm, we need to provide the spatial coordinates of the atoms belonging to the proteins to compare. Furthermore, we must also specify an equivalence set, which is composed by the atoms considered for the alignment. Usually, the alpha carbons are chosen for this. The sequence of coordinates for the alpha carbon atoms in proteins P and Q is represented by:

$$\left\{p^{(i)}\right\}_{i=1}^{|P|} \qquad \left\{q^{(j)}\right\}_{j=1}^{|Q|}$$

2. In the next step we describe the structural environment, also known as views, of each residue. Views are the set of vectors from each alpha carbon atom to the alpha carbon atoms of other residues in the same protein. The formal representation of the view of atom $p^{(i)}$ is

$$\left\{p^{(i,r)}\right\}_{i=1}^{|P|}$$

   where $p^{(i,r)}$ designates the vector with origin at $p^{(i)}$ and terminus at $p^{(r)}$.

3. After the views are obtained, the consensus matrix follows. Using dynamic programming, we fill the matrix with values representing the similarity of the vectorial views.

4. Lastly, we want to determine the best path to go through the consensus matrix. The result of this process is an alignment that provides us with an equivalence set for the various segments of P and Q that are presumed to have structural similarity.

2.1.2.3 TM-Align

One of the most standard algorithms for these types of alignments is TM-Align [62]. Most times it is applied to alpha-Carbons but the algorithms' steps may be extended to other atoms of the structures. This algorithm uses an iterative heuristic process to optimize the end result. It has a complex mathematical component that it is best explained in the original paper, but a brief explanation of its main steps is provided below.

We start off by obtaining 3 different types of alignments:

1. Alignment of the secondary structures using dynamic programming in the score matrix which contains 1 or 0 values depending if the paired elements are identical or not, respectively. Furthermore, the classification of the secondary structures for a given pair of residues is assigned at this stage, taking into consideration the 5 neighboring residues.

2. The second type of initial alignment consists of a gapless matching two structures. In order to facilitate this, the smaller of the two structures is aligned against the larger one and the selected alignment is the one that generate the lowest TM-score (described in the next section).

3. The final type is obtained by also using dynamic programming, but this time it uses a gap penalty of -1 and the score matrix is the secondary structure score matrix combined with the distance score matrix selected in the previous type of alignment.

Having obtained these different alignments, they are now submitted to an heuristic iterative algorithm that finds the best alignment for the two structures. This procedure begins by rotating the structures in accordance with the rotation matrices obtained in the initial alignments and this process is repeated until the alignment becomes stable.

2.1.2.4 MAMMOTH

Matching molecular models obtained from theory (MAMMOTH) [39] is another algorithm available to compute structural alignments and is the one implemented in the MaxCluster software which was used for this work and will be described further ahead in this document.

Once again, the algorithm has a heavy mathematical component and its steps are better described in the original article, but a brief description is provided below. Much like other structure alignment algorithms, MAMMOTH uses an heuristic approach in order to reduce its complexity, that is split into 4 steps:

1. Start by computing the unit vectors root mean square (URMS) in the direction from C_α to $C_{\alpha+1}$ for each backbone chain and then shift the resulting vectors to the origin. Following this, it is now possible to calculate the URMS distance between

the two protein segments by computing the rotation matrix that minimizes the sum of square distance between corresponding unit vectors.

2. Use the obtained matrix in the previous step to find an alignment that maximizes the local similarity between the structures. For this task, there is the need to calculate a similarity score, which in this case is the expected minimum URMS distance between two random sets of unit vectors. Through several of these calculations it is now possible to fill a similarity matrix S, to which dynamic programming is applied in order to build a local alignment.

3. Next, we find the maximum subset of similar local structures whose distance falls under a given cutoff.

4. Finally, calculate the probability of obtaining the given proportion of aligned residues.

2.1.2.5 Combinatorial Extension

One other method of obtaining a structural alignment is using CE of an alignment path defined by Aligned Fragment Pairs (AFP). In this algorithm we define an AFP as two continuous segments (one from each structure) of the same size, aligned against each other. In short, the algorithm breaks the structures into AFP, then it builds an alignment path by adding AFP until there are none left, or the remaining ones do not satisfy a set of requirements.

Currently, there is an available web server called jCE, provided by the PDB [3], which is able to provide us with structural alignments using the CE algorithm.

Below, there is a more detailed explanation of the original algorithm described in [49].

Given two proteins A and B of length n^A and n^B, we define the alignment path by the longest continuous path P of AFP of size m in a similarity matrix S, which represents all the possible AFP that conform to the criteria for structure similarity.

The algorithm has three main steps:

1. Select an initial AFP

2. Extend the alignment path by adding AFP in accordance with a set of restrictions

3. Repeat the second step until we have gone through the length of each protein or until there are no more good AFP.

When extending the path we must take into consideration that for every two consecutive AFP i and $i+1$ in the alignment path, at least one of the following conditions must hold: there may be some gaps inserted in A or B, but not in both.

The decision of whether or not to extend the alignment path is decided according the following criteria:

1. single AFP: $D_{nn} < D_0$

2. AFP against the path: $\frac{1}{n-1}\sum_{i=0}^{n-1} < D_1$

3. whole path: $\frac{1}{n^2}\sum_{i=0}^{n}\sum_{j=0}^{n} D_{ij} < D_1$

Where D_0 and D_1 represent specified cut-off distances.

Given this, we must also use an heuristic to measure AFP distances and the following can be used:

1. Distance calculated using an independent set of inter-residue distances, where each of them participates only once in the selected distance set.

2. Distance calculated using a full set of inter-residue distances, where all possible distances except those for neighboring residues are evaluated:

3. Choose the one that generates the superposition with the lowest RMSD.

By following the steps we end up with a path built with AFP which can then be used to evaluate the structural similarity between the structures.

2.2 Measuring similarities

In order to evaluate how good a given comparison between two structures is, we require some metric or score that we can use to assess this. Below, we can find more detailed descriptions of the scores that were studied during the course of this work.

2.2.1 Root Mean Square Deviation

RMSD is one of the simplest ways to evaluate the similarity of a comparison of structures. It is calculated by:

$$RMSD = \sqrt{\frac{1}{n}\sum_{i=1}^{n} d_i^2}$$

where the average is calculated over the n pairs of matched atoms and d_i is the distance between the atoms in the i-th pair. This calculation can be applied for any subset of atoms, such as the most common α-Carbons of whole proteins or specific subsets.

Generally, we can interpret the result as:

- If the value is zero, or close to zero Å, it means the proteins are identical;
- If it ranges from 1Å to 3Å, they are very similar;
- Finally, if it is greater than 3Å, we consider that the proteins have little to no similarity.

The main issues of RMSD come from its inability to account for the variation of atom positions and sequence lengths. When these situations arise they may inflate the RMSD value for a comparison and thus, lead us to withdraw misleading conclusions. Besides this, RMSD significance may vary with protein length which means that the best alignment for two given proteins may not be the one that generates the lowest RMSD.

2.2.2 Global Distance Test - Total Score

An alternative to using RMSD, is the more efficient GDT-TS [59]. With this measure, we obtain the number of α-Carbon pairs whose distances do not exceed the given threshold.

Starting with an initial set of paired atoms between two structures, the GDT-TS procedure is as follows:

1. Calculate a superposition for the set of atoms.
2. Identify pairs whose distances exceed a given distance cutoff.
3. Calculate a new superposition without the identified pairs.
4. Repeat steps 2 and 3 until the set of atoms remains the same for two consecutive iterations.

Usually, the GDT-TS score for two structures is calculated as an average of calculations with different distance cutoffs [42]. For instance:

$$GDT-TS = \frac{maxC_1 + maxC_2 + maxC_4 + maxC_8}{4}$$

where the $maxC_x$ notation designates the maximum number of atom pairs under a distance cutoff of x Å. When computing a GDT score between two unrelated structures, we can usually expect a similarity value ranging from 10Å to 20Å, whereas the values for related structures are much higher. [19]

2.2.3 Global Distance Test - High Accuracy

As the name indicates, the GDT-HA is very similar to the GDT-TS, however the first one uses lower RMSD cutoffs in order to be more restrictive on what it considers to be two similar structures and to give a larger impact to local matches among them. It is given by:

$$GDT-HA = \frac{maxC_{0.5} + maxC_1 + maxC_2 + maxC_4}{4}$$

2.2.4 Template Modeling Score

Another measure of similarity between two protein structures is the TM-score [61]. It was designed in order to handle two recurring problems of these kinds of metrics: the high sensibility to local variations by other metrics and the difficulty of interpreting the magnitude of the results. It solves these by taking into account the number of matching residues in both proteins and by scaling the result in order for it to range from 0 to 1, respectively. We calculate the TM-score by:

$$TM - score = \frac{1}{L}\left[\sum_{i=1}^{L_{ali}} \frac{1}{1 + \frac{d_i^2}{d_0^2}}\right]$$

where L is the length of the protein we are comparing, L_{ali} is the number of the equivalent residues in the two proteins and finally, d_i is the distance of the i-th pair of the equivalent residues between two structures and d_0 is given by:

$$d_0 = \sqrt[3]{L - 15} - 1.8$$

2.2.5 MaxSub

Short for Maximum Subset (MaxSub), this measure is a variation of the TM-score. The only difference between them is in the d_0 variable, which is set to:

$$d_0 = 3.5$$

Since these share a great deal of similarity, they both tackle the same issues, however they produce different results.

2.3 Machine learning

Machine learning [31] [1] is a field of artificial intelligence, which can be applied to a wide range of problems, including speech, image and pattern recognition for instance. Machine learning aims to program our computers in a way that allows them to be able to learn with experience, either gained previous to, or during runtime and use it to automatically improve performance. Due to the vast application range, there isn't a particular solution that can be used to solve any problem we come across. There are two major categories in machine learning, each one best suited for a particular kind of task, they are supervised and unsupervised learning.

Supervised learning is used when we have labeled data and we want to sort or classify each entry in our dataset. Furthermore, it can even be used to make predictions for new entries in our dataset. It is called supervised because there is available labeled data that can be analyzed in order to improve our results. In other words, we can say the program

is learning from past experience and using it to achieve better results when applied to new data.

Unsupervised learning is used in the opposite case, in which there is no labeled data to analyze. So, instead of learning from previous experience, it focuses on analyzing the dataset and providing a better understanding and insight into our data in an attempt to find patterns, regularities, outliers and correlations among our input data.

For this work in particular we are trying to find similarities among proteins through their structures, but, even though the available datasets may contain the correct labellings for each structure, this information is not being used in the learning process, only to access its performance. Therefore, we are essentially working with unlabeled data and consequently, unsupervised learning.

2.3.1 Clustering algorithms

Clustering is a form of unsupervised learning which aims to group elements in different clusters. Each cluster is defined by a set of elements which maximize a given similarity measure among members of the same cluster, while minimizing that same measure relative to members of other clusters.

Regarding cluster membership, it can be one of three types: exclusive, overlapping or fuzzy. Exclusive clustering dictates that each data point must belong to a single cluster. Overlapping clustering allows points to be part of more than one cluster. Lastly, in fuzzy clustering every point is a member of every cluster and that membership has a value that ranges from 0 to 1.

The clustering process may also be complete, which requires that every data point must belong to a cluster, or partial, which allows data points not to be assigned to any cluster.

Furthermore, clustering algorithms may belong to different categories, each of which affects how the data is processed and grouped. This causes different algorithms to produce different clusters for the same dataset. So, in order to choose an algorithm, we must first consider the kind of data we want to be processed, what kind of clusters we can expect from it and both the advantages and disadvantages of the algorithms. In the subsections ahead, some insight on the clustering categories will be provided, as well as some instances of their algorithms.

2.3.1.1 Partitional clustering

This is one of the most popular and basic techniques for data clustering. In partitional clustering, the dataset is processed with the usual goal of maximizing a measure of similarity for members of the same clusters. During runtime, while the dataset is being

clustered, the algorithms often reallocate data points to different clusters until convergence is reached. The result of these algorithms is a division of the data points into different non-overlapping clusters.

This type of clustering may be useful when predicting a protein's function, since it groups structures into non-overlapping clusters, we can expect that the protein whose function we want to predict is inserted in a group composed by those with similar structure and thus, function.

***k*-means** [1] is an example of this category of algorithms. It consists of dividing the available data in k clusters, with k being specified by the user before the algorithm's execution. Each of these clusters is represented by the mean vector of the members of the cluster, which is the prototype of that cluster. In prototype based clustering, we assign each example to the closest prototype. The algorithm goes as follows:

1. We start with an initial set of k prototypes.
2. Update the clusters by assigning the closest members to them.
3. Calculate the mean vectors for each cluster with the new members.
4. Repeat steps 2 and 3 until some stopping criteria is reached or until the updates no longer change the clusters.

To determine the initial set of prototypes there are a few applicable methods, such as the Random Partition and Forgy methods. In the first case, each data point is assigned to a random cluster and from this attribution we calculate the initial centroid of each cluster. The second method chooses k random examples to be centroids of the clusters.

k-Means has a disadvantage when compared to other algorithms, which is the need to specify the k number of clusters before execution. This value needs to be chosen according to the problem at hand and the available data. If it is too high or too small, the obtained clusters may contain irrelevant information.

***k*-medoids** is a similar algorithm to the one mentioned before since it accomplishes the exact same task in a similar fashion. The differences between these two algorithms is that k-medoids chooses its centroids from the actual data points, instead of mean points and uses distance matrices as input. For this work in particular, such differences made it an eligible algorithm for experimentation.

Affinity propagation

In an attempt to solve the inconveniences of having to specify the number of clusters before runtime, the Affinity Propagation [15] algorithm was introduced. It solves this problem by introducing a message passing concept between the data points. There are two kinds of messages: availability and responsibility.

Responsibility messages are passed between data points and based on each of their perspectives, indicate how suitable is it for others to become prototypes.

Availability messages on the other hand, are sent by prototype candidates to all other data points and indicate how adequate the candidate seems to be based on the support it has for being a prototype.

For this algorithm, we need three components:

- A similarity matrix $s_{i,k}$, filled with coefficients which indicate how alike two elements are. In this matrix, $s_{k,k}$ indicates the tendency that an elements has to become a prototype.
- A responsibility matrix $r_{i,k}$.
- An availability matrix $a_{i,k}$.

In the first step of the algorithm, we initialize the availability with zeros, $a(i,k) = 0$. Next up, responsibility is calculated using:

$$r_{i,k} \leftarrow s_{i,k} - \max_{k' \neq k}(a_{i,k'} + s_{i,k'})$$

Since in the first iteration, availabilities are set to zero, this matrix simply represents similarity between i and k. To update the availabilities, the following equations are used:

$$a_{i,k(i \neq k)} \leftarrow min\left(0, r_{k,k} + \sum_{i' \notin \{i,k\}} max(0, r_{i',k})\right)$$
$$a_{k,k} \leftarrow \sum_{i' \neq k} max(0, r_{i',k})$$

In order to identify the element which best represent the clusters, at each iteration we calculate for each element i, the element k' which maximizes the sum between the responsibility and availability. If $k' = i$, that means i is a prototype of a cluster, otherwise, i belongs to the cluster whose prototype is k'.

The algorithm can stop after a fixed number of iterations, after local decisions remains constant during a few iterations, or if the exchanged messages fall below some threshold.

Despite being an appropriate algorithm for the task at hand, it ended up being excluded in the early stages of the experiments as a consequence of its complex parameterization, which needed to be very specific for each sample if we wanted to obtain decent results.

2.3.1.2 Hierarchical clustering

These algorithms build clusters by either merging smaller clusters or dividing bigger ones. Such approaches are called divisive and agglomerative, respectively. Either way, their goal is the same: to provide us a hierarchical view of the dataset. In the agglomerative method, each entry of the dataset forms its own cluster and at each iteration, the two most similar clusters are linked. This process is repeated until all the dataset is linked. If we are using the divisive method, instead we start with a cluster that contains all the

dataset and we divide it until we have single element clusters [23]. The result of either agglomerative or divisive clustering tends to be a dendrogram or a tree shaped view that displays the data hierarchy.

At each iteration of an agglomerative algorithm, which is the one used for this work, we need to select the two closest groups to be merged, using some distance measure.

This algorithm is useful for this work since it will provide us the required views and information to make an evolutionary study of proteins. Furthermore, it also is able to cluster distance matrices, which is the form that our data takes.

2.3.1.3 Fuzzy clustering

Usually, clustering algorithms create partitions of our data on which each data point belong to one and only one cluster. Instead of this, fuzzy clustering [23] assigns to each data point a degree of membership to every cluster.

Fuzzy C-Means is an example of this type of algorithms. It is based on the minimization of a criterion function, for instance:

$$J_m = \sum_{i=1}^{N} \sum_{j=1}^{C} u_{ij}^m \parallel x_i - c_j \parallel^2$$

where m is the fuzzyness coefficient and is strictly greater than 1 and u_{ij} is the degree of membership of data point x_i to cluster center c_j.

Membership u_{ij} and cluster centers c_j are given by:

$$u_{ij} = \frac{1}{\sum_{k=1}^{C} \left(\frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)}$$

$$c_j = \frac{\sum_{i=1}^{N} u_{ij}^m * x_i}{\sum_{i=1}^{N} u_{ij}^m}$$

Considering k to be the iteration steps, the algorithm goes as follows:

1. Initialize the membership matrix, u_{ij};
2. Calculate the center vectors, c_j;
3. Update u_{ij} for k and $k+1$ iterations, with respect to c_j values;
4. Check if the difference between u_{ij} for k and $k+1$ iterations falls bellow a given threshold. If it does the algorithm stops, otherwise it returns to the second step.

In the protein's context, fuzzy clustering may provide insightful information and different groupings of predicted complexes. Since generated predictions may vary in atom position or probe orientation for example, it may advantageous that complexes belong to more than one cluster. However, since the experiments focused on SCOP data this algorithm was ultimately ruled out. Specifically, in order to measure the performance

of the clusterings, we need to compare it to an existing classification hence we require that the memberships for each structure are either 1 or 0 and not some value in between.

2.3.1.4 Density-based clustering

In these approaches, clusters are formed based on higher density zones of the dataset while other scattered elements may be considered noise.

DBSCAN: as described previously, there are a few issues with prototype-based clustering algorithms: determining the number of clusters and the inability to consider relationships among points.

The Density-based spatial clustering of applications with noise (DBSCAN) [13] algorithm introduces a different approach which deals with these problems. Through a set of definitions applied during execution, the algorithm can filter out noise, establish relations among data points and form clusters accordingly.

- **Definition 1**: The Eps-neighborhood of a point p, denoted by $N_{Eps}(p)$ is defined by $N_{Eps}(p) = \{q \in D \mid dist(p,q) \leq Eps\}$.
- **Definition 2**: (directly density-reachable) A point p is directly density-reachable from a point q with respect to Eps, MinPts if:
  1. $p \in N_{Eps}(p)$ and
  2. $\mid N_{Eps}(p) \geq MinPts \mid$ (core point condition)
- **Definition 3**: (density-reachable) A point p is density-reachable from a point q with respect to Eps and MinPts if there is a chain of points $p_1,...,p_n, p_1 = q, p_n = p$ such that p_{i+1} is directly density-reachable from p_i.
- **Definition 4**: (density-connected) A point p is density-connected to a point with respect to Eps and MinPts if there is a point o such that both, p and q are density-reachable from o with respect to Eps and MinPts.
- **Definition 5**: (cluster) Let D be a set of points. A cluster C with respect to Eps and MinPts is a non-empty subset of D satisfying the following conditions:
  1. $\forall p, q$ if $p \in C$ and q is density-reachable from with respect to Eps and MinPts, then $q \in C$. (Maximality)
  2. $\forall p, q \in C$: p is density-connected to q with respect to Eps and MinPts. (Connectivity)
- **Definition 6**: (noise) Let $C_1,...,C_k$ be the clusters of the databases D with respect to parameters Eps_i and $MinPts_i$, i = 1,...,k. Then we define the noise as the set of points in the database D not belonging to any cluster C_i, i.e. noise = $p \in D \mid \forall i : p \notin C_i$.

The algorithm starts by receiving the entire set of points as input and then it chooses an random point that has not been visited yet. Afterwards, it fetches the neighborhood of this point to be compared against MinPts. If the size of its neighborhood is lower than MinPts, this point is labeled as noise, otherwise it is a core point and it forms a cluster to which it joins its neighborhood. If there is a neighbor of this point which is also a core point, the two clusters are merged. This process is repeated until all points are visited. It is worth noting that a point which was labeled as noise in a given iteration can become part of a cluster in further iterations.

This algorithm does in fact solve the issues of prototype-based clustering, however it has its own flaws, as it is not very effective when the clusters have varying densities and the dataset has high dimensionality.

Density-based clustering may aid in the detection of noise in protein docking complexes and as such, remove structures that do not resemble the target complexes, but despite its usefulness for other contexts it is not the most appropriate one to use with the SCOP data. This is mostly due to its density concept, which will consider some structures as noise and in turn affect the cluster evaluation since not all of them will be labeled.

2.3.1.5 BIRCH

The Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) is an algorithm meant for larger datasets because it uses an approach that attempts to summarize the data without losing much information in the process. This is accomplished by using Clustering Features (CF), each of which represents a sub cluster through a triple:

$$CF = (N, \overrightarrow{LS}, \overrightarrow{SS})$$

where N is the number of data points in a cluster, $\overrightarrow{LS}$ is the linear sum and $\overrightarrow{SS}$ is the square sum of the N data points, given by:

$$\overrightarrow{LS} = \sum_{i=1}^{N} \overrightarrow{x_i}$$

$$\overrightarrow{SS} = \sum_{i=1}^{N} \overrightarrow{(x_i)^2}$$

Furthermore, we can combine multiple CFs into one, by simply computing the sum of each member of the triple.

As the name of the algorithm suggests, it uses hierarchies, which are established through the CF tree that is in turn composed by the CFs that represent our sub clusters. Such tree is composed by two types of nodes:

- Nonleaf nodes

  These nodes contain a maximum of B entries, each of which contains a triple and a pointer to the CFs that compose it.

- Leaf nodes

  These nodes contain a maximum of L entries, each of which contains a set of CF triples as well as pointers to both the previous and next node.

The CF tree is built iteratively at runtime and in order to accomplish that it follows the following algorithm:

1. Identify the appropriate leaf: Recursively descend the CF-tree by choosing the closest child node according to some distance metric (Euclidean distance, for instance).

2. Modifying the leaf: When a leaf node is reached, find the closest leaf entry, CF, and then check if they can be merged without violating the threshold condition, otherwise, create a new CF entry. If there the maximum limit of entries is exceeded, split the parent node.

3. Modify the path to the leaf: Travel back to the root of the tree, updating the information on the parent nodes. In case there is a need for a split, we must insert a new nonleaf entry in the parent node that points to the new leaf entry. If the number of entries in the parent also exceeds B, it must also be split.

Given these concepts and processes, in order to start clustering our data, BIRCH is divided into 3 steps:

1. Load into memory by building a CF tree

   At this point, the algorithm attempts to build the tree with a threshold T. If it fails to do so, it increases this value and tries again, building a smaller tree in the process.

2. Global clustering

   Apply a hierarchical clustering approach to the CF tree in order to group the data points with respect to their CF representations.

3. Cluster refining

   This last stage serves only to optimize the results obtained thus far, as it scans the current clusters searching for any inconsistencies and also applying the necessary corrections.

The main advantage of this algorithm lies in the way it treats the data that we want to cluster, which allows for the clustering to be fast when the dataset is very large and our computing resources are limited, which was the case during the experimental phase. [60].

2.3.1.6 Gaussian Mixture Models

GMM is an algorithm whose approach to clustering is based on Gaussian distributions, which is one of the most present types of distributions in datasets. The core concept is that when using it, we assume that our data fits into a specific set of independent Gaussian distributions, i.e. a mixture, with different parameters which vary depending on the data each of them contains.

As stated, GMM assumes that our dataset was drawn from x Gaussian distributions:

$$N(x|\mu, \sum_j)$$

Such that the probability given in a mixture of K Gaussians is:

$$p(x) = \sum_{k=1}^{K} w_k * N(x|\mu_k, \sum_k)$$

where w_j represents the weight of the kth Gaussian in the mixture, whose values must obey:

$$\sum_{k=1}^{K} w_k = 1 \quad 0 \leq w_k \leq 1$$

The challenge here is now to estimate the parameters θ of the distributions in the mixture so that we can find the ones that best describe the input data. In order to accomplish this, the Expectation-Maximization algorithm is used and it consists of using hidden variables Z which are related to our dataset X:

1. Guess the parameters θ of the models;
2. Compute the best values for Z given θ. In other words, estimate the probability that a given distribution generated a given point;
3. Adjust parameters θ according to Z;
4. Repeat steps 2-3 until convergence.

With these methods, we are able to compute the best w, μ and $\sum$ for our distributions.

Since this algorithm allows us to specify the amount of distributions, i.e. clusters, to use on our data, this is a useful algorithm in the context of this work. However, we can expect some inconsistencies in the generated results due to the fact that the algorithm assumes that the data fits into Gaussian distributions which may or may not be the case [47].

2.3.2 Cluster evaluation

In order to determine whether or not if the clustering algorithms performed well, we needed some way to evaluate the final results. To this end, there are various evaluation criteria that have been developed and they usually fit one of two categories, internal and external. In the following sections the evaluation criteria used in the experiments are presented. [11] [44]

2.3.2.1 Internal quality criteria

These types of metrics focus on evaluating the compactness of the clusters, which in other words is equivalent to measuring the homogeneity of members of the same cluster. Despite this, we must take into account that by themselves these types of criteria cannot tell us if a given clustering is good or not, only if the grouped elements are similar to each other or not. What actually dictates if our clusters are good, are the metrics that evaluate our results with respect to an existing classification that makes sense to humans and for that, we have the external quality criteria which will be described in the next section. Due to this, when analyzing our results we must take into account both internal and external quality criteria.

- **Calinski-Harabasz score:** this score represents the ratio between the within cluster dispersion and the between cluster dispersion and it is calculated by the following equations:

$$CH(k) = \left[\frac{B(k)}{W(k)}\right] \times \left[\frac{(n-k)}{(k-1)}\right]$$

  Where N is the number of data points, k corresponds to the number of clusters and W_k and B_k represent the within cluster and between cluster dispersions, respectively, which in turn are obtained by:

$$W_k = \sum_{q=1}^{k} \sum_{x \in C_q} (x - c_q)(x - c_q)^T$$

$$B_k = \sum_{q} n_q (c_q - c)(c_q - c)^T$$

  Here, C_q is the set of points that belong to cluster q, c_q is its center and lastly, n_q is the number of its data points. By looking at the way this score is calculated, we can conclude that it tends to be higher when the clusters are dense and well separated, thus, the higher the result the better the model used to cluster the data.

  Despite its usefulness overall, this is not a good metric for this particular context due to the fact that its value does not have a theoretical upper limit and its dimensions

also vary depending on the data used on the clustering process. This means that if we have two similar clusterings that were obtained by clustering different structure similarity measure matrices, the resulting CH values may be vastly different from one another, even if they are similar according to other metrics. Given this, it would be hard to evaluate the disparity of clusters obtained using different protein similarity measures. [6] [41]

- **Silhouette score**: is the overall representation of how well each data point fits its cluster. The silhouette score depends on:
  - $a(i)$ = average dissimilarity of i to all other objects of cluster A.
  - $d(i, C)$ = average dissimilarity of i to all objects of cluster C.
  - $b(i) = \min_{C \neq A} d(i, C)$

  It is possible for us to use different distance measures to determinate the dissimilarity, such as the Manhattan or Euclidean distances. To calculate the silhouette score for a given point, we use the following expression:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

  From this definition, we can infer that the score for a data point will range from -1 to 1. Given that this metric has both a lower and upper bound, it does not share the CH drawbacks which makes this a metric that will allow us to compare the performance of the clusters regardless of algorithm, sample or distance matrix used. Furthermore, it favors clusterings that are compact and convex, which is what we are aiming towards. [46] [41]

2.3.2.2 External quality criteria

These measures are more useful when the overall structure of the clusters can be compared to some predefined classification of the data [44]. In this work, the datasets that contain the data to be clustered, already have groups and hierarchies formed within them. SCOP and CATH are databases with hierarchies aimed at better understanding evolutionary relationships, hence have their own classification for the structures and can be used to compare the ones obtained with the clustering algorithms.

The external criteria mentioned ahead will aid us in performing these comparisons and in turn assess the performance of the clustering algorithms in order to determine if they are generating good or bad results:

- **Adjusted Mutual Information (AMI)**: returns 1 if the two provided input clusterings are in complete accordance with each other and 0 or a negative value if its the

opposite situation. This score accounts for chance and is biased towards clusterings with a higher amount of clusters. It is given by:

$$AMI(U,V) = \frac{MI(U,V) - E\{MI(U,V)\}}{avg\{H(U),H(V)\} - E\{MI(U,V)\}}$$

MI can be thought of as a measure reduction in uncertainty for predicting outcome of a system after we have observed the other part, which in this case is the existing classification of the data and H is the entropy associated with the partitions. A more detailed description can be found in its original paper. As for its usefulness for this work, it shows us the fraction of structures that are correctly grouped without taking into account stochastic effects. [55] [41]

- **Homogeneity**:

  This measure reflects the degree to which data points are similar to each other. In other words, a clustering result satisfies the homogeneity criteria if the members of each cluster belong to a single class [45] [41]. It is given by:

$$h = 1 - \frac{H(C|K)}{H(C)}$$

  Where $H(C|K)$ is the conditional entropy of the classes depending on the cluster assignments and $H(C)$ the entropy of the classes. These coefficients can be determined through:

$$H(C|K) = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|}\frac{n_{c,k}}{n} \times \log\left(\frac{n_{c,k}}{n_k}\right)$$

$$H(C) = -\sum_{c=1}^{|C|}\frac{n_c}{n} \times \left(\frac{n_c}{n}\right)$$

  In short, this metric will evaluate if the resulting clusters tend to contain structures from a single classification.

- **Completeness**:

  Since homogeneity only focuses on assessing the quality of the clusters, we need a counterpart to it, in order to evaluate how much of the data was assigned to the correct cluster. This is essentially what completeness is: a measure that reflects the portion of the total data points that have correctly been assigned to its cluster. It is given by:

$$c = 1 - \frac{H(K|C)}{H(K)}$$

Both the conditional entropy and cluster entropies can be calculated in a similar fashion to that of the homogeneity measure [45] [41]. Given this, in the protein context, Completeness allows us to know if all members with the same SCOP classification have been assigned to the same cluster.

2.4 Hyperparameter optimization methods

Usually, when studying machine learning models there is a need to decide which parameters to use before we actually apply them to our data, be it the number of clusters we want our data to be split in, the weight of a given variable in the model, a coefficient, etc. These parameters, named hyperparameters, which need have to be known before running the algorithm can impact the quality of our results either negatively or positively depending on whether or not they have the correct values for the data at hand. [32]

2.4.1 Genetic algorithm

Genetic algorithms base themselves on the concept of evolution and natural selection in order to find the best solution for a given problem. Specifically, they generate individuals tailored to the problem at hand, who are then selected to reproduce based on their fitness, which means the characteristics of the better parents will be passed down to the next generation. This process is divided in the following steps:

1. Generate an initial population

   In this first step, we need to create a set of individuals, each of which containing a set of values that correspond to the variables in our problem that we wish to optimize (genes).

2. Calculate the fitness function

   One of the most important concepts in this type of algorithm is the fitness functions, which is the one that determines how good each individual is. In this step, we calculate the fitness of all of the individuals in our population.

3. Selection

   At this stage, we select the best individuals from our population so that they can pass their genes to the next generation. It should be noted that this selection is made randomly, however, individuals with a higher fitness score do have a higher probability of being chosen.

4. Crossover

   This step is were we actually combine our individuals. There are many ways to accomplish this but the main concern here is only that we take genes from both parents in order to produce a new individual.

5. Mutation

   The final step is to apply mutations, which are changes that are made on the genes of each individual of the population and have a very low chance of occurring.

The second through fifth steps are to be repeated until a certain amount of iterations or until the algorithm reaches convergence, i.e. the best individual remains the same for a few consecutive iterations. With these steps we are then very likely to obtain the best individual whose genes that produce the best value for our fitness function. [54] [32]

2.4.2 Bayesian optimization algorithm

An alternative method to estimate the parameters for our clusterings are Bayesian optimization algorithms. This type of algorithm uses a set of initial data points to build a probabilistic model of our fitness function using Gaussian processes in order to describe its values with respect to the input parameters. This allows us to find the parameters that minimize our fitness functions by using interpolation techniques that explore areas of that model without requiring exhaustive computations of said function.

The Bayesian optimization algorithm typically use Gaussian processes in order to model the fitness function, which are important because they give us a range of expected values in unexplored areas between sampled data points. Furthermore, these probability distributions also us to introduce the prior and posterior concepts in the optimization. To put it simply, the prior probability corresponds to our prediction of what the data looks like before we actually observe it and the posterior probability is the combination of the prior probability combined with new data samples.

In short, the optimization algorithm is as follows:

1. Given a set of data observations, build a probabilistic model, using the previously described Gaussian processes, for the fitness function f whose minimum we wish to find;

2. Optimize an acquisition function, such as the Expected Improvement which evaluates f where we expect the value to be the lowest;

3. Update the probabilistic model based on the results of the previous evaluation;

4. Sample the next observations while balancing exploration and exploitation, i.e. we should calculate f where we expect its value to be minimized but also in regions where we do not have a lot of information;

5. Repeat steps 1 through 4 for i iterations

Due to this, with Bayesian optimization we can find the best hyperparameters for our clustering algorithms without having to perform an exceedingly high amount of calculations, as opposed to the previously described genetic algorithms. However, i must

be an appropriate value for the problem at hand, otherwise, at the end of the optimization process we are not guaranteed to have found the absolute minimum of f which can lead to misleading results. [52] [32]

2.5 Available data and tools

This section will provide a brief description of the available software that was considered for the experiments of this work as well as the content of some of the existing protein data platforms and the role they play in their field.

2.5.1 Tools

2.5.1.1 MaxCluster

MaxCluster is a protein alignment and clustering tool. It allows us to compute RMSD, GDT, TM-score and MaxSub for a given alignment of two structures using the MAMMOTH algorithm. This program does support list processing, which was the preferred approach, since it saves its users the trouble of creating a proper representation of the resulting data for each single alignment by returning a single file. [19]

2.5.1.2 Python

Python a is very popular, high-level and general-purpose programming language. It can be used to build standalone applications, automate tasks and many other things. Besides these uses, Python has been one of the most used languages in machine learning since it provides several powerful libraries designed for the type of tasks the field requires.

DEAP

The Distributed Evolutionary Algorithms in Python library (DEAP) is a framework that covers a broad range of artificial intelligence algorithms. Among this collection, there are tools that allow its users to build a genetic algorithm specifically tailored to the task at hand [14].

Matplotlib

A data visualization library. Matplotlib is a common and flexible tool which allows the user to create a wide range of plots, such as 2-D histograms, bar charts, scatter plots and tables. [22]

Numpy

Another one of the most popular Python libraries. Numpy provides its user with N-dimensional arrays, easy to use array manipulation and transformation as well as complex algebraic computations with overall better performance than native arrays. [56]

Scikit-learn

Clustering algorithms are one of the main aspects of this work, which is why the Scikit-learn [41] library was used. It was developed for the Python programming language and

it provides us with implementations of supervised and unsupervised algorithms, among which, are clustering functions and evaluation methods.

Seaborn

Much like Matplotlib, this is yet another data visualization library, however, Seaborn provides easier to use plots for statistical analysis. [57]

SKOPT

This Python module contains the implementation of a framework which allows its users to create instances of algorithms that can optimize other expensive and noisy black-box functions. Among said algorithms, we have the Bayesian optimizer, which was used in the experiments of this work.

2.5.2 Data sources

The following platforms are some of the available data sources besides SCOPe, that could have been used to conduct these experiments. They are very well known and used in the protein analysis context even if most of them serve different purposes.

2.5.2.1 CAPRI

The Critical Assessment of PRedicted Interactions (CAPRI) [24] is an experiment designed in order to test protein docking algorithms. CAPRI is a blind experiment in the sense that the participants do not know the structure that they are trying to predict. However, after each round is finished the results are evaluated and made available to the public.

2.5.2.2 CASP

The Critical Assessment of protein Structure Prediction (CASP) [34] is the complement of the CAPRI experiment. After the predictions are submitted in the CAPRI experiments there is the need to evaluate and rank the models produced by the participants' algorithms, which is exactly what CASP does. Having finished the evaluation of each experiment, several resulting files are made available to the public. These files contain very useful information, specifically, the coordinates of the prediction and target models and summaries of the experiments.

2.5.2.3 CATH

The Class, Arquitecture, Topology and Homologue superfamily protein structure classification database (CATH) [50] protein structure database is an online platform that holds data regarding evolutionary relationships of protein domains. Currently, it contains 95 million protein domains which are classified into 6119 superfamilies. The acronym originates from the hierarchical classification of the domains:

- Class (C): ranges from 1 to 4 and describes the classification of the folds, respectively, mainly α, mainly β, mixed α β and lastly, little secondary structure.
- Architecture (A): describes the three-dimensional conformation of the secondary structures.
- Topology (T): holds information regarding how the secondary structure elements are connected and arranged.
- Homologous superfamily (H): level that indicates if there is evidence of common ancestry.

Furthermore, protein domain superfamilies were subclassified into functional families. These are groups of protein sequences and structures that have a high probability of sharing the same function.

2.5.2.4 PDB

The Protein Data Bank (PDB) [3] is the largest repository of information regarding 3-D structures of biological molecules, including proteins. It is maintained by the Research Collaboratory for Structural Bioinformatics (RCSB) and it is one of the most important resources for studying protein structure. Currently, this platform freely supplies data for researchers and other databases, which is available in PDB format and includes information such as atomic coordinates, molecule names, primary and secondary structures.

2.5.2.5 SCOP

The Structural Classification of Proteins database (SCOP) [35] database contains the description of the relationships of all its known protein structures. It is hierarchically organized in four levels:

1. Superfamily: describes information of far evolutionary distances, meaning the proteins have low sequence identity, however through structure it is possible to trace common ancestors.
2. Family: describes information of near evolutionary distances, but in this classification, sequence and structural similarities are more evident.
3. Fold: describes geometric relationships among the proteins. Specifically, this level indicates that proteins have common folds if they have similar secondary structures.
4. Class: describes the classification of the folds, which are: all α-helices, all β-helices, $\alpha\beta$ for those with both, $\alpha + \beta$ for those with both and are largely segregated and finally, Multi-domain is the classification for the domains that have no current known homologues.

With the information contained in this database it is possible to perform an evolutionary study of proteins. By clustering their structures in order to find similar protein groups, we can then compare the obtained results with the ones present in the database. This will allow us to test both the effectiveness of the clustering algorithms and the measures used to compare protein structures.

Nowadays, there are 3 different SCOP platforms that contain both structures and classification data. The first one, SCOP, is the original one which was released in 1994 and contains the tree-like hierarchical structure through which we are able to study the relationships between the proteins present in the database. SCOP's last release was in June of 2009, after which SCOP2 was introduced, with the goal of providing a better framework for protein structure annotation and classification. Such improvements were achieved by changing the hierarchy's representation into a directed acyclic graph that allows other types of relationships among the data, however at this time, SCOP2 is still a prototype that only contains a portion of the total data. Lastly, SCOPe (extended) can be seen as the direct continuation of the classic SCOP hierarchy and is still maintained and updated regularly to this day, hence it was the chosen platform for this work. Though the SCOP versions differ among themselves, both their annotation and classification processes remain the same, meaning that the structures are evaluated according various factors, namely, automated comparison methods that take both sequences and structures into account, the function of a given protein and manual curation from experts. Figure 2.3 illustrates the standard process and the mentioned classification factors. [7] [2]

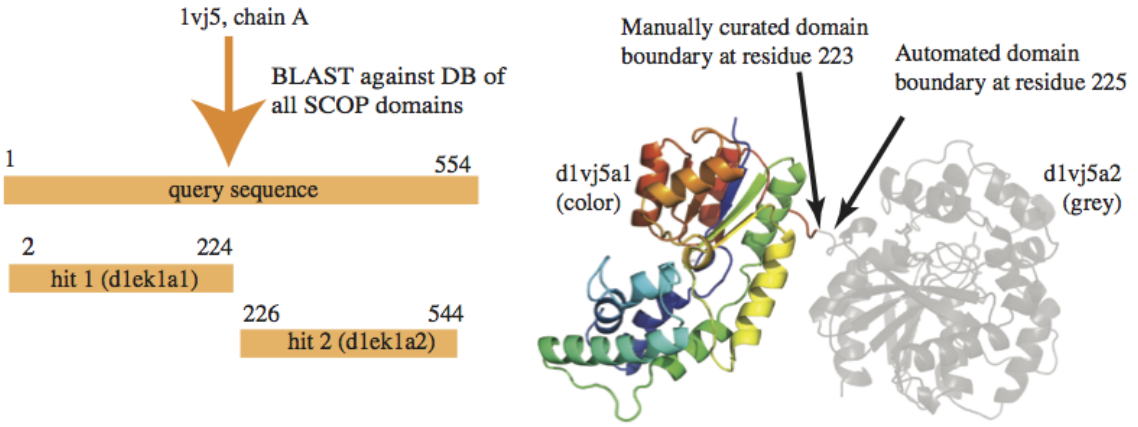


Figure 2.3: Standard SCOPe classification process

Chapter 3

Development and experimental phase

To reiterate, so far we have established the importance of protein databases and the roles they play in the study of proteins, be it in phylogenic analysis, docking, function prediction, etc. Furthermore, we also discussed how these platforms organize and classify their data, which is the result of a thorough manual analysis by an expert based on several factors combined with automated methods

Since the classifications that result from these analysis affect the conclusions that drawn from the protein data at hand, it is best that there are as few gaps during the course of such analysis. Given this, we want to improve the way that the protein structures are grouped in order to also improve the way they are organized by automated methods and to aid in extracting more meaningful and accurate information.

RMSD can be considered one of the mentioned gaps due to the drawbacks that stem from way it is calculated, hence, with the experiments made we aimed to mitigate these drawbacks in the clustering process by studying the other similarity measures presented in this document, namely, GDT-HA, GDT-TS, MaxSub and TM-score.

As for the studied algorithms, many of them were originally considered for this study but were ultimately ruled out because they required difficult parametrization (Affinity Propagation), noise detection hindered the results of the experiment (DBSCAN) or simply because the concept applied by the algorithm did not make sense for the problem at hand (Fuzzy C-Means). Given this, only algorithms without these restrictions were studied, specifically: Hierarchical, BIRCH, K-Medoids and Gaussian Mixture Models.

The following experiments are based upon protein structures from the SCOPe database, which allows us to use its available classifications to evaluate how good each of the similarity measure and clustering algorithm performs by looking at the clustering evaluation metrics they produced, specifically, how many correct classifications we got overall (AMI).

3.1 Alignment computation and available hardware

In order to start clustering the structures, it was essential to obtain the structural alignments and the resulting values. To accomplish this task, the MaxCluster program was used. Maxcluster supports list processing that allows the computation of all against all alignments for the given input structures, which is both faster to calculate and easier to process than single alignments, however, during runtime this feature seems to hold the alignment matrix in memory until it has been fully computed, which means that we may run out of resources during this computation. Given this issue, with the available resources, the maximum amount of structures per sample is around 2500. Furthermore, since the time required to compute each sample can take several days and requires nearly the entire available computational resources, the following experiments were limited to 7 different sets of protein structures. To provide a little bit more context, all the performed computation were made using an Ubuntu 16.04 virtual machine running on a I7-7700HQ CPU with 8 GB of RAM.

3.2 Hyperparameter tuning

In the initial stages of the experiments, a brute force approach was used, meaning that every single combination of hyperparameters was computed for each sample and for each algorithm that was studied. Despite requiring the computation of thousands of clusterings this was at first feasible because the early testing samples were small, however, as the other used samples grew in size this approach became unusable. Given that for this work we are considering large enough samples of the whole SCOPe database we must find another approach for this problem since if we want to apply these matrix combinations we are facing millions of clustering computations.

Due to these performance issues, in this work two hyperparameter optimization methods were applied, namely, a genetic algorithm and bayesian optimization.

Starting with the genetic algorithm, early experiments also made it clear that it was not the method best suited for this particular situation. It was found that during its run, the algorithm would always find the best individual in the first or second generations, which meant that depending on our population size and maximum stagnant iterations, we were once again spending a lot of time computing clusters and more often than not performing repeated computations, due to the fact that these algorithms tend to preserve the best individuals for the following generations.

Finally, we reached Bayesian optimization, which uses a probabilistic model to predict which combinations of weights are best for our matrices. This approach greatly reduces the number of clusterings calculated since we control how many times each clustering algorithm will be computed during its execution. We can argue that there will be wasteful computations much like there are in the genetic algorithm, however they will be significantly reduced since the search method requires less tests with the parameters.

3.3 Combining structure similarity measures

One of the main approaches to be experimented in this work, is the concept of using different protein similarity measures in the clustering process in order to compensate for each other's shortcomings so that we are able to obtain a clustering that is in line with the SCOPe database and also makes sense according to the experts' analysis. In short, this experiment attempts to find the ideal percentages that we should take from each measure in order to increase the number of correct structure labellings.

3.3.1 Measure correlation

Since the Bayesian optimizer is attempting to optimize the coefficients of five different similarity measures, it is expected that it will need to run for a high number of iterations. If we assume that each weight ranges from 0 to 100 and the coefficients from 1 to 10, that means that there are several millions of possible combinations of these hyperparameters. Thus, in order to reduce the complexity of this problem for the optimizer, the correlations between GDT-HA/GDT-TS and MaxSub/TM-score were calculated. An example of these correlations can be seen in figure 3.1, and even though this is just an example from one of the studied samples, the results shown are consistent across all others. Such high correlations between the previously mentioned pairs were to be expected, since the computation methods for GDT-HA/GDT-TS and MaxSub/TM-score pairs are very similar.

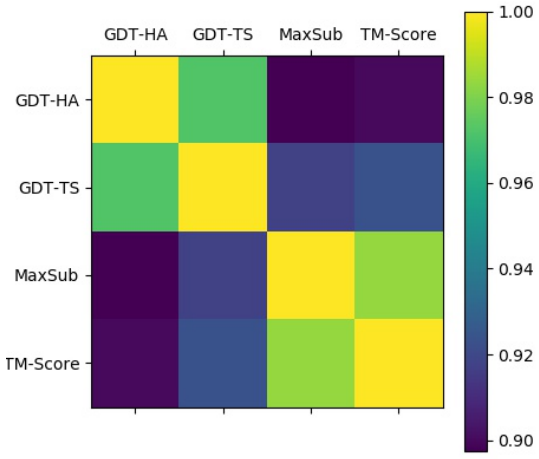


Figure 3.1: Example of the correlation between different similarity measures for a given sample.

Given this, the mentioned pairs of measures were combined with each other, by averaging their values before the optimizer procedure begins, which makes it so that it has less variables to optimize.

3.3.2 Data processing

From the all against all alignments, Maxcluster outputs a single file that contains two types of information: the numbers that represent each structure within each Maxcluster execution, ranging from 1 to the amount of structures in the input, and the pairwise similarities for each structure combination according to the used protein similarity measure.

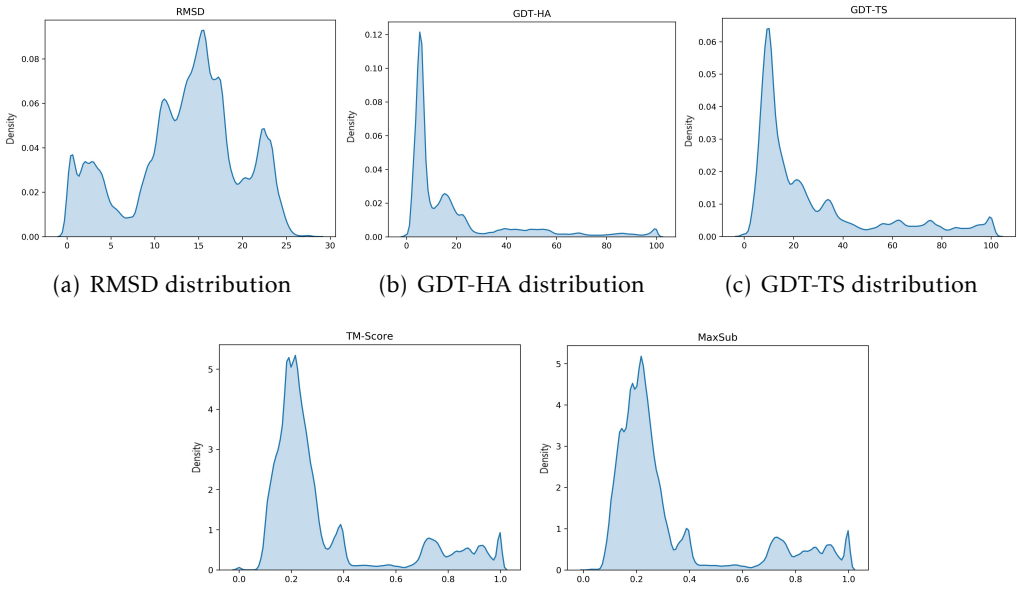
One issue that was raised in the early stages of the experiment was that the studied structure similarity measures are on different scales. RMSD for instance, starts from 0 and does not have a limit, while GDT is presented as a percentage, from 0 to 100. Since we want to combine different similarity measures through their respective distance matrices, they all need to be in the same scale.

There are two main ways of achieving this, those being normalization and standardization, which are obtained by:

$Normalization : z_i = \frac{x_i - min(x)}{max(x) - min(x)}$

$Standardization : z = \frac{X - \mu}{\sigma}$

In order to choose the most appropriate way to scale our data, its density estimation was determined. Usually, when the data distributions are more resemblant of a Gaussian distribution, standardization is the preferred approach. Since it is not the case, as we can see in the distributions for one of the samples, shown below in figure 3.2, normalization was chosen in order to rescale the data.



(a) RMSD distribution (b) GDT-HA distribution (c) GDT-TS distribution

(d) TM-Score distribution (e) Maxsub distribution

Figure 3.2: Density of values present in samples for each of the different similarity measures

It is worth mentioning that even though only one sample's kernels are shown, the remaining studied samples also follow similar patterns.

3.3.3 Integrating Bayesian optimization

Summing up, for each protein similarity measure combination for the current sample, the clustering process and the respective weight estimation (also shown in 3.3) as follows:

1. Load SCOPe labels for the current sample
2. Load the similarity matrices for the current sample
3. Combine MaxSub/TM-score and GDT-TS/GDT-HA
4. Normalize the matrices
5. Run the Bayesian optimizer using AMI as fitness function in order to determine the weights and coefficients of each similarity measure. At each iteration:
   a) Obtain the combined matrix (RMSD + GDT-TS/GDT-HA + MaxSub/TM-score) with the hyperparameters explored in the current iteration, i.e, by summing each weighted component.
   b) Apply one of the studied clustering algorithm to the resulting matrix
   c) From the output labels, calculate the evaluation metrics
   d) Add the results to the optimizer sample and update the probabilistic model
   e) Repeat this loop for a fixed number of iterations

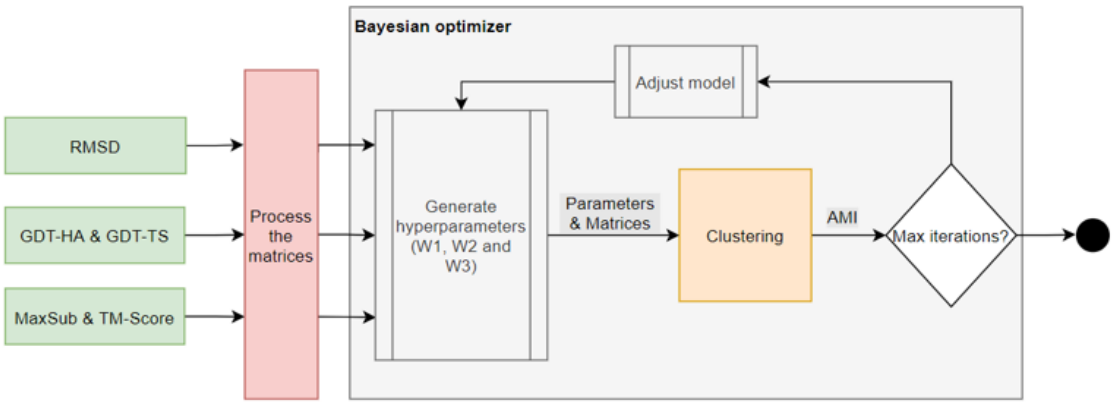


Figure 3.3: Process used to cluster the structures using a combination of measures

In order to analyze the results from this process, a comparison must be made by looking at the metrics that were generated from a combination of similarity measures with the ones generated by the each of the measures by themselves. Through this analysis, which will be discussed ahead, we will see whether or not the combination of measures can improve the clusters and if so, in which aspects they being improved.

3.3.4 Clustering the structures

Having cleared the process used during this experiment, let us now discuss the observed results. To reiterate, the goal of this experiment was to attempt to use different measures during the clustering process hoping that they would compensate each other's weaknesses. However, the results obtained from the optimizer did not demonstrate this.

The first step in the experiment was to cluster the data using the individual distance matrices from each similarity measure, so that we have reference values to compare the results to. Right away, by applying the process described in the previous section, it is possible to identify a clear gap between the studied algorithms and the protein similarity measures in terms of their performances. During the experimental part, MaxSub and TM-score convincingly outperformed RMSD and GDTs, by consistently achieving better AMI outputs, since the corresponding distributions are generally higher, which can be seen in figure 3.4 where the results across all samples displayed. It was already suspected that RMSD would perform worse than other measures due to the shortcomings that were discussed previously, however, the magnitude of this disparity in regards to MaxSub and TM-score was not expected.

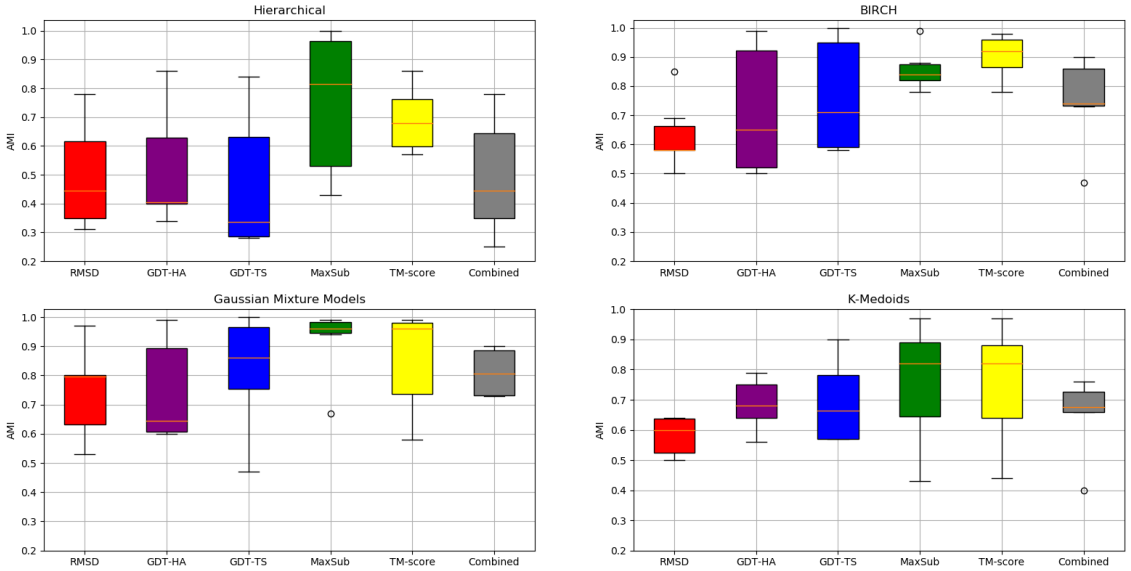


Figure 3.4: These plots show the AMI values obtained by clustering the structures of different samples by using the studied similarity measures by themselves and the ones obtained through their combination (Combined column).

As far as algorithms go, it was a bit more difficult to make predictions regarding which of them would be the best performer. Despite that, by analyzing the data it was fairly simple to conclude that the GMM algorithm outclassed the Hierarchical, BIRCH and K-Medoids algorithms. However, despite being able to achieve good results using just a single structure similarity measure, that was not always the case and as such the Bayesian optimizer was used in order to improve the metrics in such samples.

The optimizer ran for 200 iterations, meaning that that was the number of different combinations of weights and coefficients that the clusters were obtained with. Despite the long run, the optimizer tended to reached convergence fairly quickly, usually around 50 iterations. Since this happened consistently, we can infer that either the optimizer was fed data that was not very helpful to the clustering process or that it determined a set of very good parameters in the beginning of the run. Given the disparity between the results of MaxSub/TM-score and remaining measures, the first option is the most likely, because the optimizer will tend not to use the worse performing measures since they will only lower the AMI output. This is further supported by the plots shown in figure 3.5, where it is shown that the majority of clusterings were obtained by using very low values for the weights of the RMSD and GDT matrices, essentially nullifying or minimizing their effect on the clustering process.

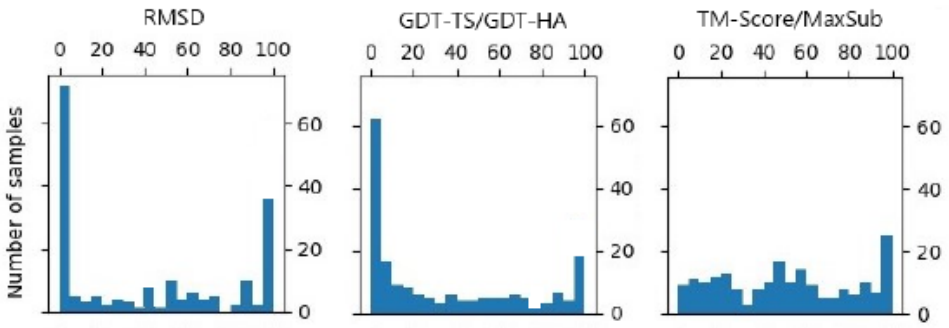


Figure 3.5: In this set of plots are displayed the distributions of values explored by the optimizer during its run. More specifically, the y axis represents the amount of times that each x value, i.e. weight, was chosen during the Bayesian optimizer run.

Furthermore, as evidenced by the optimizer plot and the AMI values generated by individual matrices, it turns out that by combining the different similarity matrices into one the clustering results are not improved, since figure 3.4 also shows that the distribution of AMI values obtained through the combinations of measures are consistently lower than the ones obtained through clustering with a single similarity measure. Given that throughout this experiment with all the algorithms and samples the results that the optimizer provided were very consistent, we can conclude that we cannot improve the clustering process by using a combined distance matrix.

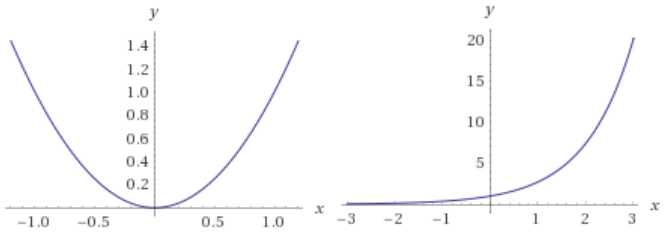
In conclusion, it was verified that the resulting AMI of a combination of measures can only go as high as the highest AMI value for a single measure, thus, since MaxSub and TM-score are usually the better performers it is simply best to use one of them in the clustering process. Thus, it is not worth it to pursue this approach any further.

3.4 Applying non-linear transformations

3.4.1 Transforming the matrices

Since the previous experiment did not yield the results that were expected, we took a step back and tried to focus on the individual similarity matrices. The goal was once again to attempt to improve the quality of these matrices in order to provide better data for the clustering process. One way to achieve this might be by applying non-linear transformations to the matrices, i.e. transformations that change the proportions of the similarities among the structures represented in the matrices. This may be helpful because in some cases these representations are not enough to draw the correct "border" for each cluster, which is where these transformations come in to try and make it clearer for the clustering algorithms to do this.

The studied polynomial functions were variations of the parabole and exponential and they were chosen with the mindset of improving RMSD based clustering due to the drawbacks that come with its calculation and also because it is one of the easiest methods to use and therefore, the one that is most commonly used. As can be seen from their plot in figure 3.6, these functions have vertical asymptotes which vary according to their coefficients. This is mainly useful with RMSD because as previously discussed, above certain values it loses its significance and it becomes harder to interpret, meaning that it is also more difficult to replicate what would be the experts' input in the clustering. Due to this, the goal is now to use these functions and pass our matrices through them, so that higher RMSD values are grouped closer to each other, which mitigates the negative impact that they might have in the clustering process. The same logic can also be applied to the remaining similarity measures although to a smaller degree, since by definition they are already have a minimum and maximum value and thus, we can only expect that the impacts of outliers in each sample is reduced.

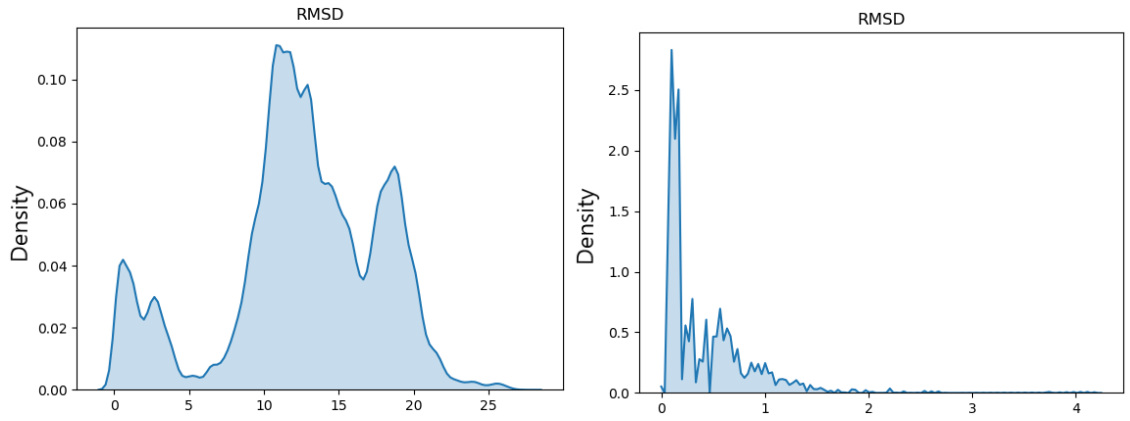


(a) Parabole (b) Exponential

Figure 3.6: Non-linear transformations plots

In order to better illustrate the experiment, let us consider figure 3.7, where we can see the impact that applying a non-linear transformation has on the input data. In that figure, it is possible observe that the RMSD values are spread out through a large spectrum of values which can be a bad thing considering that some algorithms, like Gaussian Mixture

Models, cluster the structures by looking at the "shape" of the data. However, if a non-linear transformation is applied to that data, the distribution becomes much less spread out, less noisy and more dense.



(a) Original RMSD distribution

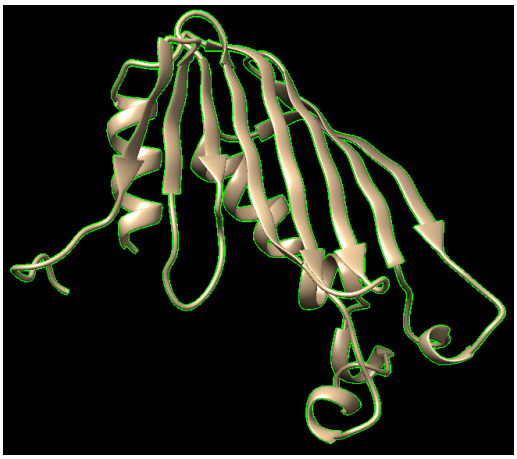
(b) Modified RMSD distribution

Figure 3.7: This figure shows the difference between the RMSD values distribution from one of the studied samples and that same distribution after an exponential transformation was applied to it.

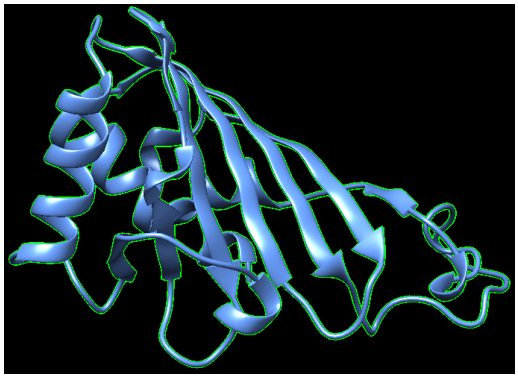
To further illustrate this, let us consider figure 3.8, where three domains are displayed. The domains, 1ofg, 1cer and 1gcu all belong to the previously mentioned sample and they belong to two different superfamilies. Regarding them, by using RMSD's base distance matrix together with the Gaussian Mixture Models algorithm, that is the matrix without any non-linear transformation, it resulted in a clustering that misslabeled domain 1gcu into the same superfamily of the domain 1ofg, instead of the correct one, that of 1cer. By looking at the disposition of the domains' coordinates in 3D space it is understandable that the algorithm got somewhat "confused" which led to its mistake, since we can clearly see that they all share considerable similarities, particularly their center sections which contain a set of parallel β-sheet.

This is further supported by figure 3.9 where we can see an alignment between domain 1gcu to the other two mentioned domains. In it, is shown that 1gcu does indeed match much better with 1ofg than with 1cer, since the former alignmnent yields an RMSD value of 3.4Å and the latter 10.7Å.

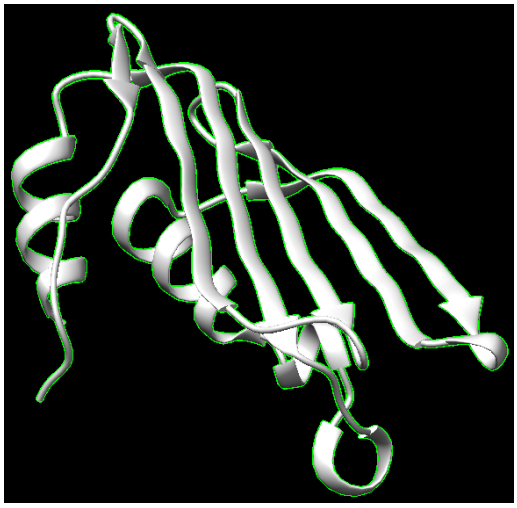
This is once again due to the shape of the data, which causes enough spread and deviation in the distribution to fool the algorithm into inserting the domain in the wrong cluster. Now, if we apply a non-linear transformation, such as the one exemplified in figure 3.7, the spectrum and density of values that hinder the clustering process, that is to say RMSD values above 8 for instance (this threshold varies from sample to sample according to the RMSD values present in the distribution), is reduced and the domain 1gcu is grouped into the correct cluster. This is very helpful for the algorithm and it is a process that makes sense in this context since we know that RMSD loses significance



(a) 1ofg domain



(b) 1cer domain



(c) 1gcu domain

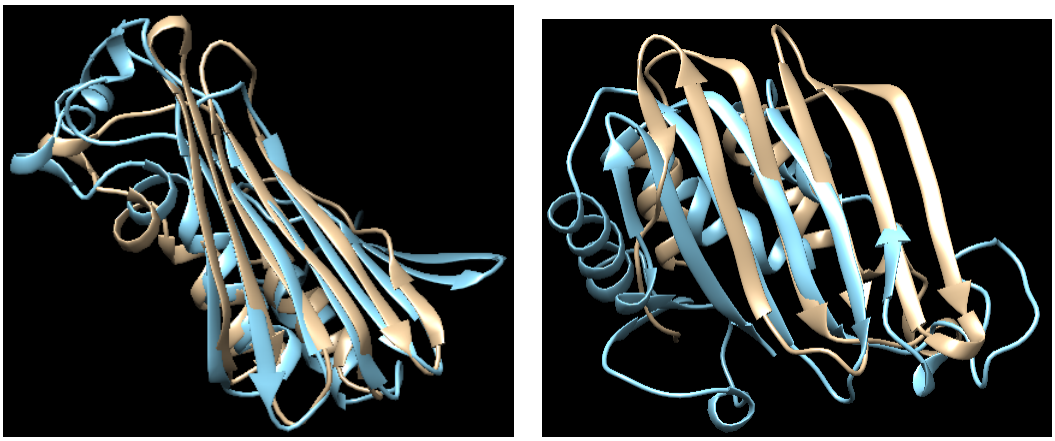
Figure 3.8: Display of the domains' backbone layout in 3D space

when it is high. After applying this change in the matrix, the border between each cluster can be drawn more clearly and thus, it becomes easier to assign a cluster to each domain of the sample.

3.4.2 Clustering with the transformed matrices

Let us now take one of the available samples as a reference and apply the mentioned transformations. Using the Bayesian optimizer we can adapt the process described previously, however, instead of using it to estimate the weights and coefficients of the matrices we now use it to estimate the coefficients of the mentioned functions for a single matrix. The used process for this experiment (also represented in 3.10) looks like this:

1. Load SCOPe labels for the current structure sample



(a) 1gcu aligned with 1ofg (b) 1gcu aligned with 1cer

Figure 3.9: This figure shows the difference between the alignments of domains from the 1gcu, 1ofg and 1cer structures.

2. Load one of the similarity matrices for the current sample
3. Run the Bayesian optimizer using AMI as fitness function in order to determine the coefficients of the transforming function. At each iteration:
   a) Process the similarity matrix using the non-linear transformation with the hyper parameters provided by the optimizer
   b) Apply one of the studied clustering algorithm to the resulting matrix
   c) From the output labels, calculate the evaluation metrics
   d) Add the results to the optimizer sample and update the probabilistic model
   e) Repeat this loop for a fixed number of iterations

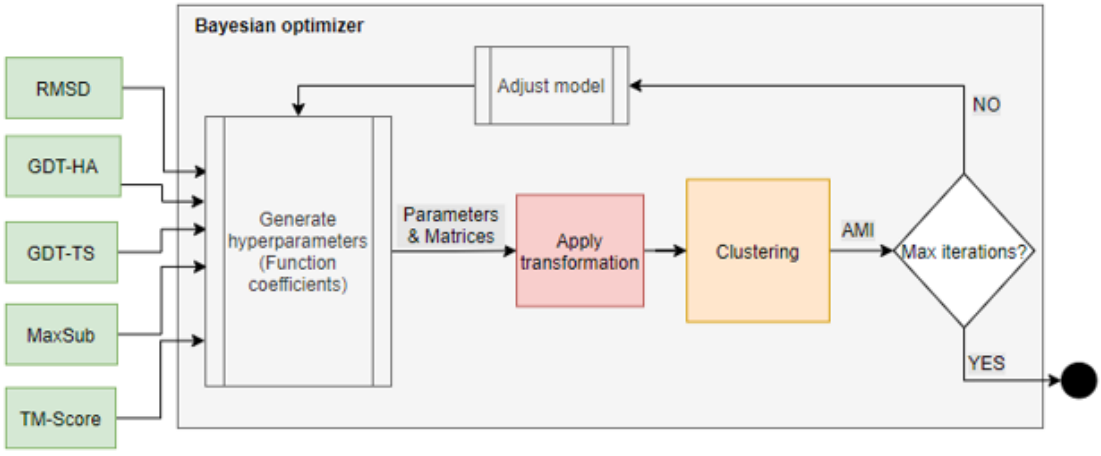
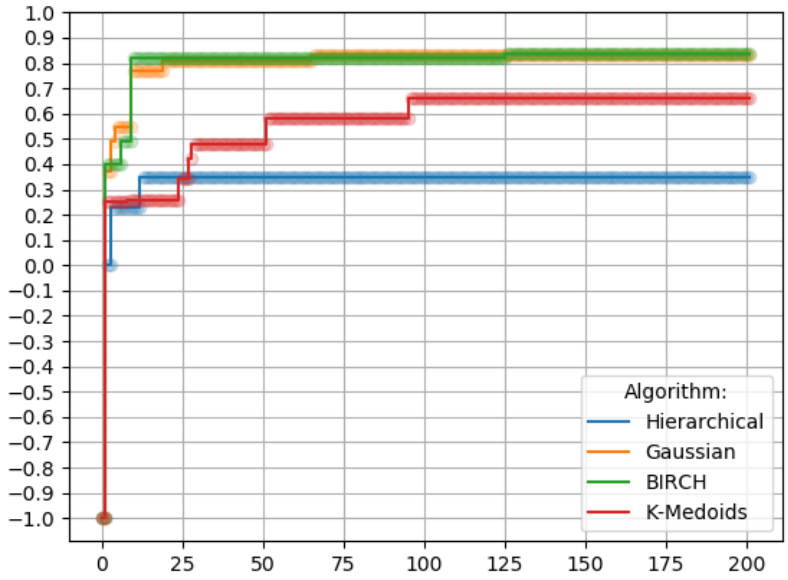


Figure 3.10: Process used to cluster the structures using a non linear transformation

In the plots shown in figures 3.11, we can see the convergence plot from the optimizer runs for a given structure similarity measure. Right away it is possible to verify that these transformation have a positive effect on the clustering process given that the majority of the AMI metrics for the different measures and algorithms were improved.



(a) This plot shows us the highest AMI value at each iteration, generated by clustering the data using a matrix transformed by a parabolic function

	AMI	
Algorithm	**Base**	**Transformed**
Hierarchical	0.35	0.35
BIRCH	0.75	0.84
GMM	0.8	0.82
K-Medoids	0.64	0.66

(b) This table shows us the comparison between the best result obtained by the optimizer, i.e., by transforming the matrix (Transformed column), and the ones obtained by clustering the original matrix (Base column).

Figure 3.11: Clustering results obtained from the modification of an RMSD matrix using a parabole as a non-linear transformation

Because this approach yielded positive results, it was replicated throughout other samples and the results were very consistent, meaning that the resulting cluster evaluation metrics were improved when compared to the results obtained without non-linear transformations, and as such the resulting clusters were also better overall.

It should be mentioned that even though this experiment contemplated the Hierarchical clustering algorithm, the obtained results have shown us that this approach is

somewhat useless when applied to this particular algorithm. Since this algorithm clusters the data by iteratively finding the next most similar structure to the one that it is currently considering, based on a distance metric, we cannot expect this approach to improve the quality of the resulting clusters. This is because even though the previously mentioned non-linear transformations change the data in the similarity matrices, in most cases, it will still preserve the relationships among the structures, only with different proportions. Thus, for that particular algorithm the next most similar structure at each given iteration is prone to being the same regardless of the transformation. To reiterate, this downside does not apply to the other studied algorithms since they cluster the data by looking at the overall structure, and not at the individual elements. Due to these reasons, this model can be discarded from the remaining processes, since it does not show either good baseline results or signs of improvement with the transformations.

3.4.3 Cross Validation and model selection

After optimizing the best hyper parameters for each sample it was possible to establish that the non-linear transformations can indeed help us in obtaining better AMI values, and as a consequence, better clusters out of each sample. However, so far, this approach assumes that the user will determine a set of hyper parameters by using a similar approach to the one described in the previous section.

Given that this an expensive process, both in terms of computational resources and time, an attempt was made to provide a set of hyper parameters that can assure at least a small but consistent improvement to the overall quality of the clusterings, regardless of the provided sample. This type of process is usually done by combining hyper parameter tuning with cross validation techniques.

Cross Validation (CV) is an essential step in this process, as it allows us to get an unbiased estimate of how well a model will perform when used on unseen data, which is the goal at this point, to provide a good general recommendation of algorithm, measure and transformation that will work well for new sets of proteins.

If we were to estimate the parameters that maximize the AMI average using all the available samples in the optimizer run, the final results, even though they would be positive, would be biased towards the dataset that was used. In this scenario, since the model's parameters are very tuned to training dataset used in the optimizer, if we applied that same model to a new sample, it would most likely under perform, as it is over fitted to initial set and thus, it may not be prepared to handle irregularities or patterns that other samples might introduce.

Given this issue, Cross Validation suggests that the available data is split into 3 sets: training, validation and testing. The training set, as the name indicates, is the one where the models will be trained, which means, it will be the set that the optimizer will use to obtain the best models. The validation set is where we will verify the performance of the models obtained from the training set. Finally, the test set is the one that will not

be part of the training process and will only be used to get an unbiased estimate of the generalization performance.

Since the set of samples is small, a Leave One Out Cross Validation (LOOCV) strategy was applied. This is a particular case of CV, where the models are trained using N-1 samples, i.e. leaving one out in order to validate the performance of the trained model. Finally, this process is replicated for the remaining combinations of samples, always validating the trained models' performances with the sample that is left out, as indicated by table 3.1. A representation of the full CV process can be seen in figure 3.12. It should be noted that the optimizer will work slightly differently at this point, since now it will maximize the average AMI, as opposed to a single AMI, obtained by clustering all the samples in the training set at each iteration with the set of parameters that is being explored. [1]

	Training						Validation
Case 1	S1	S2	S3	S4	S5	S6	S7
Case 2	S1	S2	S3	S4	S5	S7	S6
Case 3	S1	S2	S3	S4	S6	S7	S5
Case 4	S1	S2	S3	S5	S6	S7	S4
Case 5	S1	S2	S4	S5	S6	S7	S3
Case 6	S1	S3	S4	S5	S6	S7	S2
Case 7	S2	S3	S4	S5	S6	S7	S1

Table 3.1: This table shows the sets of samples used to estimate the hyper parameters that maximize the average AMI and the sample with which they are tested with, using the Leave One Out Cross Validation method.

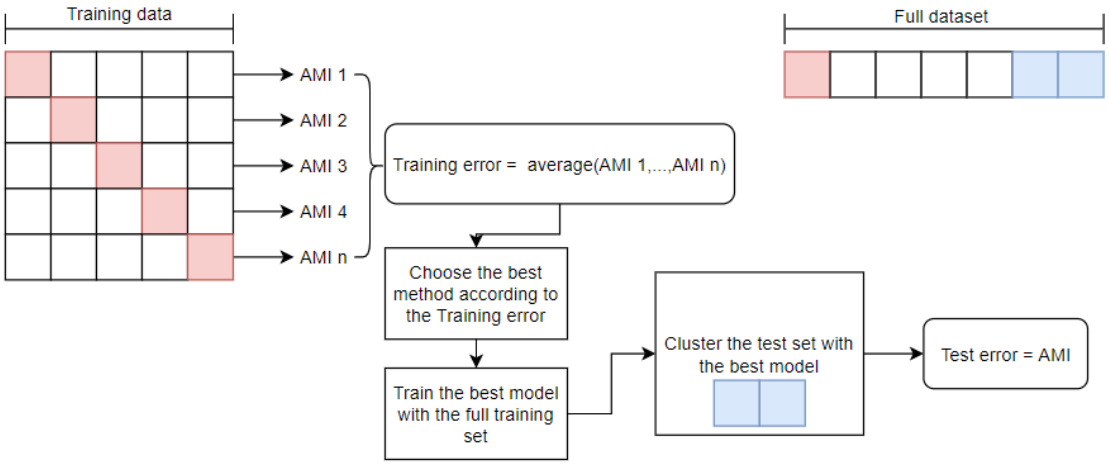


Figure 3.12: This figure illustrates the Leave One Out Cross Validation process. For each clustering method that is being tested we will train n models on different CV folds, validating the performance of each one on the corresponding validation sample (Red square). After determining which method has the highest AMI on average, the full training set will be used to train another model, which will cluster the testing samples in order to obtain the test error.

At this stage of the CV process, we will now obtain a set of AMI values that stem from clustering the validation samples with the best models from the training set. These AMIs are now averaged in order to obtain the training error, which will help us in the process of selecting the best model.

It should also be mentioned that for two of remaining algorithms, namely, BIRCH and GMM, the optimizer must take into account some parameters. For BIRCH, they are the Threshold and the Branching Factor. The first one controls the maximum radius that a given sub cluster can have, meaning that if a new element is added to it and exceeds the Threshold, a new sub cluster will be created with that element. The Branching Factor, controls the maximum number of sub clusters per tree node. As for GMM, we have the covariance type, which essentially controls the shape of the mixture elements, *tol* is the threshold which stops the Expectation Maximization algorithm if it is exceeded, *regcovar* is a value added to the diagonal of the covariance matrix and *initparams* defines how the algorithm is going to be initialized. For both of these algorithms, the studied spectrum of values is shown in tables 3.2 and 3.3.

BIRCH	
Branching factor	[100, 1000]
Threshold	[0.1, 10]

Table 3.2: BIRCH tuning parameters

GMM	
Covariance type	full, diag, spherical, tied
tol	$[1*10^{-6}, 1*10^{-3}]$
regcovar	$[1*10^{-7}, 1*10^{-4}]$
initparams	random, kmeans

Table 3.3: GMM tuning parameters

According to the training results, shown in 3.13, the best model is generally a combination between the MaxSub measure clustered with the Gaussian Mixture Models algorithm, without an additional transformation. At this stage, CV indicates that we now select this model as our winner in order to train a finalized one, with the full length of the training set. Such model, is the one presented in table 3.4.

GMM	
Covariance type	diag
tol	$2*10^{-5}$
regcovar	$7*10^{-6}$
initparams	kmeans

Table 3.4: Finalized GMM model

The final step of the CV process is now to determine the test error, by applying these

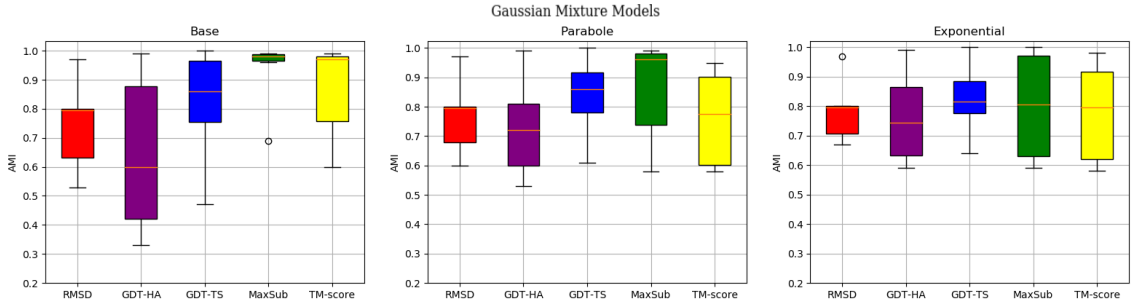
sets of parameters to the test set, which corresponds to 2 new samples that have not been used in any training process, and thus, the obtained model is not biased towards it. The clustering of these new samples with the finalized model generated the AMI values presented in table 3.5.

	Default GMM	Trained model
Test 1	0.98	1
Test 2	0.72	0.72

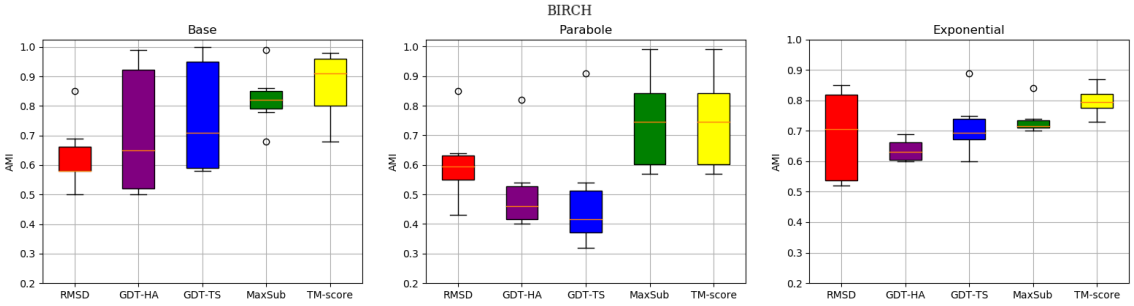
Table 3.5: Resulting AMIs from the clustering of the test set with the trained model, compared to the values obatined with default GMM parametrization

As this table shows, there was a small improvement on the performance of one of the samples, which makes sense with this model. This is because as opposed to the non linear transformations approach, we are not modifying the input data fed to the algorithm, rather, we are just tweaking its execution parameters. These parameters, although they may vary from context to context, do not have the power to dramatically improve the final results since the logic behind the algorithm remains the same, they can only worsen them if they are not well adjusted at all to the data at hand.

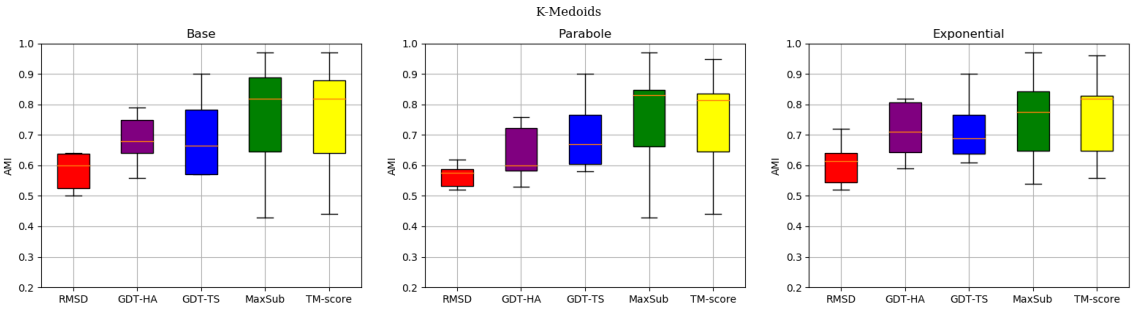
In short, it was expected that the application of non linear transformations would improve the clusters, but this final turn of events was unsurprising, since both MaxSub and TM-score have been consistently performing better than RMSD and the GDTs during the course of the experimental phase. Nevertheless, as both RMSD and the GDTs have demonstrated improvements when clustering with a transformation, also using the GMM algorithm, it means that the databases that use these measures frequently could possibly benefit from these transformations in order to improve the quality of the obtained clusters. But, as the results show, for these measures, this generalist approach does not always guarantee good results and not every combination between measures and algorithms can be improved by transformations. On the other hand TM-score/MaxSub generally are the best performers when using their original distance matrices, i.e. without transforming them, across the studied algorithms. However, these two measures seem to be either unaffected or hindered by the application of non-linear transformations. Given this, it simply makes no sense to recommend other measures when MaxSub and TM-score are consistently better, and as such anyone who uses RMSD or GDTs should do it less and less.



(a) This plot shows the AMI values generated by clustering different samples using the Gaussian Mixture Models clustering algorithm



(b) This plot shows the AMI values generated by clustering different samples using the BIRCH clustering algorithm



(c) This plot shows the AMI values generated by clustering different samples using the K-Medoids clustering algorithm

Figure 3.13: Comparison between the quality of the clusterings obtained by maximizing the average AMI at each step of the optimizer. The Base column contains the values of the optimization process with parametrization tuning (except for K-Medoids), the other columns show the performance with the non linear transformations on top of that.

Chapter 4

Conclusion and future work

Conclusions

In conclusion, the results obtained throughout the experiment were a mixed bag, with respect to what was expected. It turns out that the clustering process using the combination of similarity measures does not improve the quality of the generated clusters regardless of the used algorithm. After analyzing the results, it was made clear that MaxSub and TM-Score were consistently better that the other measures, and as a consequence, by combining them with others, the results just tend to be as high as the ones obtained when using just one of these measures in the process.

On the other hand, we were able to improve RMSD and GDT based clusterings by applying non-linear transformations to those similarity matrices, even if by a small margin. Despite this, it seems that even though the clustering process with these measures was improved, they are still not on par with both TM-Score and MaxSub, as they tended to outperform the others without additional data processing. At this point, these two can be recommended to be regularly used instead of the more popular RMSD, however, should RMSD and the GDTs be kept in use, the studied non-linear transformations could boost their performances by a small margin.

Regarding the studied algorithms, it was also clear that the best performing algorithm was GMM, especially when paired with MaxSub. This is because it was the algorithm that contained the better distributions of AMI values for each of the studied measures either with or without applying non-linear transformations. Following GMM, both BIRCH and K-Medoids are capable of performing decently when used to cluster protein structures since they can also group domains fairly well. However, the Hierarchical algorithm was the one with the poorest results and cannot really be recommended considering the performance of the other studied algorithms.

Finally, despite not being able to improve the combination between the GMM and

MaxSub clustering through non linear transformations or parametrization, the main objective of this thesis, which was to study a good way to cluster protein structure data, was fulfilled, since the presented results indicate that this combination will generally yield quality clusters of proteins.

Limitations

In terms of limitations, one that was present throughout the experiments was the size and amount of the samples. Since the available hardware was not very powerful, the samples were restricted to a size of around 2500 structures. A few experiments were made with bigger samples, however, if the mentioned threshold was exceeded, the distance matrices could not be computed because MaxCluster would run out of usable memory and crash.

The other major limitation present in the experiments was the need to specify the number of clusters we wished to extract from each sample. During the experiments, since the samples were part of the SCOPe database, for every structure there was a known label available, which was used to determine the number of clusters beforehand. The problem with this is that in a more realistic scenario, we would perhaps want to split a set of novel structures, without labellings, into different clusters, and in this case the number of clusters would not be known.

Future work and final remarks

Given that throughout the experiments the previously mentioned limitations were present, in future prospects, it would be interesting if SCOPe and other protein related platforms would be willing to use their resources to cluster their data with GMM and MaxSub in order to see if the resulting clusters are more resemblant of the analysis of their experts. This of course would mean that there would be few limitations in terms of both the quantity of samples and structures in each one of them. Finally, in order to mitigate the drawback of having to specify the number of clusters, a new study could be made, focusing on ways to identify some aspect of the obtained clusters, such as internal quality cluster evaluation metrics (Silhouette score), that can indicate to the user that the number of desired clusters is wrong and should be adjusted in order to better fit to the data at hand. Alternatively, other algorithms that do not use the number of clusters as a parameter could also be studied.

As for the academic purposes that this thesis was developed, the explored topic proved to be the toughest academic challenge for the author and promoted the development of new skills, such as learning the Python language, machine learning techniques, exploring scientific articles and applying computer science to other areas.

Bibliography

[1] E. Alpaydin. *Introduction to Machine Learning*. Second edition. The MIT Press, 2010.

[2] A. Andreeva et al. «SCOP2 prototype: A new approach to protein structure mining.» In: *Nucleic acids research* 42 (Nov. 2013). DOI: 10.1093/nar/gkt1242.

[3] H. Berman et al. «The Protein Data Bank Nucleic Acids Research, 28, 235-242.» In: *URL: www. rcsb. org Citation* (2000).

[4] C. I. Branden et al. *Introduction to protein structure*. Garland Science, 1999.

[5] F. J. Burkowski. *Structural bioinformatics: an algorithmic approach*. CRC Press, 2008.

[6] T. Caliński and J. Harabasz. «A dendrite method for cluster analysis.» In: *Communications in Statistics-theory and Methods* 3.1 (1974), pp. 1–27.

[7] J.-M. Chandonia, N. K. Fox, and S. E. Brenner. «SCOPe: Structural Classification of Proteins—extended, integrating SCOP and ASTRAL data and classification of new structures.» In: *Nucleic Acids Research* 42.D1 (Dec. 2013), pp. D304–D309. ISSN: 0305-1048. DOI: 10.1093/nar/gkt1240. eprint: http://oup.prod.sis.lan/nar/article-pdf/42/D1/D304/3647135/gkt1240.pdf. URL: https://dx.doi.org/10.1093/nar/gkt1240.

[8] M. Dayhoff, R. Schwartz, and B. Orcutt. «22 A Model of Evolutionary Change in Proteins.» In: *Atlas of protein sequence and structure*. Vol. 5. National Biomedical Research Foundation Silver Spring, MD, 1978, pp. 345–352.

[9] W. L. DeLano. *PyMOL*. 2002.

[10] O. Dror et al. «MASS: multiple structural alignment by secondary structures.» In: *Bioinformatics* 19.suppl_1 (2003), pp. i95–i104.

[11] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification*. John Wiley & Sons, 2012.

[12] J. Ebert and D. Brutlag. «Development and validation of a consistency based multiple structure alignment algorithm.» In: *Bioinformatics* 22.9 (2006), pp. 1080–1087.

[13] M. Ester et al. «A density-based algorithm for discovering clusters in large spatial databases with noise.» In: *Kdd*. Vol. 96. 34. 1996, pp. 226–231.

[14] F.-A. Fortin et al. «DEAP: Evolutionary Algorithms Made Easy.» In: *Journal of Machine Learning Research* 13 (2012), pp. 2171–2175.

[15] B. J. Frey and D. Dueck. «Clustering by passing messages between data points.» In: *science* 315.5814 (2007), pp. 972–976.

[16] J.-F. Gibrat, T. Madej, and S. H. Bryant. «Surprising similarities in structure comparison.» In: *Current opinion in structural biology* 6.3 (1996), pp. 377–385.

[17] I. Halperin et al. «Principles of docking: An overview of search algorithms and a guide to scoring functions.» In: *Proteins: Structure, Function, and Bioinformatics* 47.4 (2002), pp. 409–443.

[18] S. Henikoff and J. G. Henikoff. «Amino acid substitution matrices from protein blocks.» In: *Proceedings of the National Academy of Sciences* 89.22 (1992), pp. 10915–10919.

[19] A Herbert and M. Sternberg. *MaxCluster: a tool for protein structure comparison and clustering*. 2008.

[20] L. Holm and C. Sander. «Mapping the protein universe.» In: *Science* 273.5275 (1996), p. 595.

[21] L. Holm and C. Sander. «Protein structure comparison by alignment of distance matrices.» In: *Journal of molecular biology* 233.1 (1993), pp. 123–138.

[22] J. D. Hunter. «Matplotlib: A 2D graphics environment.» In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[23] A. K. Jain, M. N. Murty, and P. J. Flynn. «Data clustering: a review.» In: *ACM computing surveys (CSUR)* 31.3 (1999), pp. 264–323.

[24] J. Janin. «Assessing predictions of protein–protein interaction: the CAPRI experiment.» In: *Protein science* 14.2 (2005), pp. 278–283.

[25] W. Kabsch. «A solution for the best rotation to relate two sets of vectors.» In: *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography* 32.5 (1976), pp. 922–923.

[26] M. Knudsen and C. Wiuf. «The CATH database.» In: *Human genomics* 4.3 (2010), p. 207.

[27] A. S. Konagurthu et al. «MUSTANG: a multiple structural alignment algorithm.» In: *Proteins: Structure, Function, and Bioinformatics* 64.3 (2006), pp. 559–574.

[28] E Krissinel and K Henrick. «Protein structure comparison in 3D based on secondary structure matching (SSM) followed by C-alpha alignment, scored by a new structural similarity function.» In: *Proceedings of the 5th international conference on molecular structural biology*. Vol. 88. Vienna. 2003.

[29] I. Kufareva and R. Abagyan. «Methods of protein structure comparison.» In: *Homology Modeling*. Springer, 2011, pp. 231–257.

[30] S. C. Li. «The difficulty of protein structure alignment under the RMSD.» In: *Algorithms for Molecular Biology* 8.1 (2013), p. 1.

[31] T. M. Mitchell. *Machine Learning*. First edition. McGraw-Hill, Inc., 1997.

[32] N. Mori, M. Takeda, and K. Matsumoto. «A comparison study between genetic algorithms and bayesian optimize algorithms by novel indices.» In: *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. ACM. 2005, pp. 1485–1492.

[33] R. Mosca, B. Brannetti, and T. R. Schneider. «Alignment of protein structures in the presence of domain motions.» In: *BMC bioinformatics* 9.1 (2008), p. 352.

[34] J. Moult et al. «A large-scale experiment to assess protein structure prediction methods.» In: *Proteins: Structure, Function, and Bioinformatics* 23.3 (1995), pp. ii–iv.

[35] A. G. Murzin et al. «SCOP: a structural classification of proteins database for the investigation of sequences and structures.» In: *Journal of molecular biology* 247.4 (1995), pp. 536–540.

[36] S. B. Needleman and C. D. Wunsch. «A general method applicable to the search for similarities in the amino acid sequence of two proteins.» In: *Journal of molecular biology* 48.3 (1970), pp. 443–453.

[37] M. E. Ochagavia and S. Wodak. «Progressive combinatorial algorithm for multiple structural alignments: application to distantly related proteins.» In: *Proteins: Structure, Function, and Bioinformatics* 55.2 (2004), pp. 436–454.

[38] C. A. Orengo and W. R. Taylor. «SSAP: sequential structure alignment program for protein structure comparison.» In: *Methods in enzymology* 266 (1996), pp. 617–635.

[39] A. R. Ortiz, C. E. Strauss, and O. Olmea. «MAMMOTH (matching molecular models obtained from theory): an automated method for model comparison.» In: *Protein Science* 11.11 (2002), pp. 2606–2621.

[40] W. R. Pearson. «An introduction to sequence similarity ("homology") searching.» In: *Current protocols in bioinformatics* (2013), pp. 3–1.

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

[42] A. Poleksic. «Algorithms for optimal protein structure alignment.» In: *Bioinformatics* 25.21 (2009), pp. 2751–2756.

[43] C. P. Ponting and R. R. Russell. «The natural history of protein domains.» In: *Annual review of biophysics and biomolecular structure* 31.1 (2002), pp. 45–71.

[44] L. Rokach and O. Maimon. «Clustering methods.» In: *Data mining and knowledge discovery handbook*. Springer, 2005, pp. 321–352.

[45] A. Rosenberg and J. Hirschberg. «V-measure: A conditional entropy-based external cluster evaluation measure.» In: *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*. 2007.

[46] P. J. Rousseeuw. «Silhouettes: a graphical aid to the interpretation and validation of cluster analysis.» In: *Journal of computational and applied mathematics* 20 (1987), pp. 53–65.

[47] H. Sahbi. «A particular gaussian mixture model for clustering.» In: *4th International Symposium on Neural Networks (ISNN 2007), Nanjing, China*. 2007.

[48] M. Shatsky, R. Nussinov, and H. J. Wolfson. «A method for simultaneous alignment of multiple protein structures.» In: *Proteins: Structure, Function, and Bioinformatics* 56.1 (2004), pp. 143–156.

[49] I. N. Shindyalov and P. E. Bourne. «Protein structure alignment by incremental combinatorial extension (CE) of the optimal path.» In: *Protein engineering* 11.9 (1998), pp. 739–747.

[50] I. Sillitoe et al. «CATH: comprehensive structural and functional annotations for genome sequences.» In: *Nucleic acids research* 43.D1 (2014), pp. D376–D381.

[51] T. F. Smith and M. S. Waterman. «Identification of common molecular subsequences.» In: *Journal of molecular biology* 147.1 (1981), pp. 195–197.

[52] J. Snoek, H. Larochelle, and R. P. Adams. «Practical bayesian optimization of machine learning algorithms.» In: *arXiv preprint arXiv:1206.2944* (2012).

[53] F. Teichert, U. Bastolla, and M. Porto. «SABERTOOTH: protein structural alignment based on a vectorial structure representation.» In: *BMC bioinformatics* 8.1 (2007), p. 425.

[54] A. Thengade and R. Dondal. «Genetic algorithm–survey paper.» In: *MPGI National Multi Conference*. Citeseer. 2012, pp. 7–8.

[55] N. X. Vinh, J. Epps, and J. Bailey. «Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance.» In: *Journal of Machine Learning Research* 11.Oct (2010), pp. 2837–2854.

[56] S. v. d. Walt, S. C. Colbert, and G. Varoquaux. «The NumPy array: a structure for efficient numerical computation.» In: *Computing in Science & Engineering* 13.2 (2011), pp. 22–30.

[57] M. L. Waskom. «seaborn: statistical data visualization.» In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021.

[58] Y. Ye and A. Godzik. «Multiple flexible structure alignment using partial order graphs.» In: *Bioinformatics* 21.10 (2005), pp. 2362–2369.

[59] A. Zemla. «LGA: a method for finding 3D similarities in protein structures.» In: *Nucleic acids research* 31.13 (2003), pp. 3370–3374.

[60] T. Zhang, R. Ramakrishnan, and M. Livny. «BIRCH: an efficient data clustering method for very large databases.» In: *ACM Sigmod Record*. Vol. 25. 2. ACM. 1996, pp. 103–114.

[61] Y. Zhang and J. Skolnick. «Scoring function for automated assessment of protein structure template quality.» In: *Proteins: Structure, Function, and Bioinformatics* 57.4 (2004), pp. 702–710.

[62] Y. Zhang and J. Skolnick. «TM-align: a protein structure alignment algorithm based on the TM-score.» In: *Nucleic acids research* 33.7 (2005), pp. 2302–2309.

[63] J. Zhu and Z. Weng. «FAST: a novel protein structure alignment algorithm.» In: *PROTEINS: Structure, Function, and Bioinformatics* 58.3 (2005), pp. 618–627.