Pedro Miguel Franco Arguelles

Bachelor in Computer Science Engineering

Clustering of protein structures

Dissertation plan 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

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

February, 2018

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 multiple 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 múltiplos 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

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 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 [32], 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 ancestry. 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 [7] [22].

- **Docking**

  Another application where it is necessary to group and compare structures is protein docking [19]. 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.

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 Squared Deviation (RMSD), which is essentially the averaged distance between all pairs of matched residues. As

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 [34]. 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.

Given the issues and usefulness of comparing and grouping protein structures, with this work we aim to explore alternative ways to measure the similarity of proteins and also to experiment with different clustering algorithms to group proteins, since it is not uncommon that they produce diverse results among themselves. We can then use the obtained results to compare them against other protein databases so we are able to determine if the clustering process has generated correct results while using the alternative measures of similarity. For instance, we can group docking models according to their resemblance to the target complex, or group structures with respect to their evolutionary process and then evaluate the correctness of the obtained groups by looking at classifications of databases such as GWIDD or SCOP, respectively.

Furthermore, some of the resources in this field rely on manual classification of structures while grouping them. For instance, the CATH (Class, Architecture, Topology, Homologue superfamily) database 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 [29].

As such, with this work we also want to explore automated methods that are able to group structures in different ways, not only to find the most effective way of grouping them, but also to focus on the information that it is obtained when using different grouping criteria.

Chapter 2

State of the art

In this chapter will describe some relevant elements of a protein, methods to measure structure similarity and provide a description of the state of the art regarding clustering and protein alignment algorithms.

2.1 Proteins

2.1.1 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 [6].

2.1.2 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 since they must adapt in order for them to continue carrying on with 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. They 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 [47].

2.1.3 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 must 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 [44].

2.2 Protein comparisons

As formerly 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 with the comparison between these two 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.

2.2.1 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.

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.

So far, there are a lot of algorithms and software that have been developed for sequence alignment. A few examples are FASTA [35], BLAST [2] and PSI-BLAST [3] for

database searches, PyMol [11] (supports commands for sequence alignments) for pairwise alignments and ClustalW [8] for multiple sequence alignments.

Despite the previously mentioned algorithms, 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 $B = b_1,...,b_m$. A substitution matrix $s(a_n, b_m)$ 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) [10] and BLOSUM (Blocks Substitution Matrix) [20]. 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.

2.2.1.1 Needleman-Wunsch

For global alignment, the Needleman-Wunsch [40] 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.

2.2.1.2 Smith-Waterman

For local alignment, the Smith-Waterman [54] 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.

2.2.2 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.

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 [38], MAMMOTH [43], VAST [18], MASS [12], SSM [31] and DALI [21]. Another possible approach that FAST [61] and SABERTOOTH [56] explore, is obtaining a structural alignment based on pairwise residue distances. TM-Align [60] is a methods that compute alignments based on a new metric developed specifically for the structural alignment problem. Furthermore, there are also multiple structural alignment methods such as: CBA [14], POSA [57], MultiProt [50], MALECON [41] and MUSTANG [30].

In the following sections, a few other algorithms will be described in order to provide a sense of how these techniques work.

2.2.2.1 Superposition based on Kabsch's algorithm

This algorithm allows us to compare two proteins by directly superimposing the structures [28]. 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 [11].

The steps of the algorithm have a fairly complex mathematical component, which is more thoroughly explained in [7], but is 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 Root Mean Squared Deviation.

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

2.2.2.2 Sequential Structure Alignment Program

The SSAP [42] 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 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 [53] (which will be described in the next chapter) 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.

2.2.2.3 Combinatorial Extension

One other method of obtaining a structural alignment is using combinatorial extension (CE) of an alignment path defined by Aligned Fragment Pairs (AFPs). 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 AFPs, then it builds an alignment path by adding AFPs until there are none left, or the remaining ones do not satisfy the requirements.

Currently, there is an available web server called jCE, provided by the Protein Data Bank [5], 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 [51].

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 AFPs of size m in a similarity matrix S, which represents all the possible AFPs that conform to the criteria for structure similarity.

For every two consecutive AFPs i and $i+1$ in the alignment path, at least one of three conditions must hold:

1. $p_{i+1}^A = p_i^A + m$ and $p_{i+1}^B = p_i^B + m$

2. $p_{i+1}^A > p_i^A + m$ and $p_{i+1}^B = p_i^B + m$
3. $p_{i+1}^A = p_i^A + m$ and $p_{i+1}^B > p_i^B + m$

In these conditions, p_i^A is the AFP's starting residue in protein A at the i^{th} position in the alignment path. This definition is extended for p_i^B. The first case describes two consecutive AFPs aligned without gaps. The second and third ones, also describe two consecutive AFPs but with gaps inserted in proteins A and B, respectively.

The AFP which starts the path can be selected from any position in the similarity matrix S, following this we consecutively add other AFPs while satisfying the previous three conditions. However, in order to limit the gap size, the second and third conditions are reinforced:

1. $p_{i+1}^A \leq p_i^A + m + G$
2. $p_{i+1}^B \leq p_i^B + m + G$

where G represents the maximum gap size allowed. This value affects the algorithm's computation time and as such, should be kept as small as possible, though not so small that as to affect the search resolution. Empirically, the authors suggest that G be set at 30 and m at 8.

Given these rules and definitions, we now need heuristics to evaluate similarities and extend the path.

1. distance D_{ij} calculated using an independent set of inter-residue distances, where each of them participates once and only once in the selected distance set:

$$D_{ij} = \frac{1}{m}\left(\left|d^A_{p_i^A p_i^A} - d^B_{p_i^B p_i^B}\right| + \left|d^A_{p_i^A+m-1,p_j^A+m-1} - d^B_{p_i^B+m-1,p_j^B+m-1}\right| + \sum_{k=1}^{m-2}\left|d^A_{p_i^A+k,p_j^A+m-l-k} - d^B_{p_i^B+k,p_j^B+m-l-k}\right|\right)$$

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

$$D_{ij} = \frac{1}{m^2}\sum_{k=0}^{m-1}\sum_{l=0}^{m-1}\left|d^A_{p_i^A+k,p_j^A+l} - d^B_{p_i^B+k,p_j^B+l}\right|$$

3. RMSD from optimally superimposed structures.

In these equations, D_{ij} represents the distance between two fragments from protein A and B defined by two AFPs at positions i and j in the alignment path. $d^A_{(i,j)}$ represents the distance between the α-Carbon atoms at positions i and j in protein A. Setting $i = j$ gives us the distance between two fragments of the same AFP. p_i^A denotes the AFP's starting

residue position in protein A at the i^th position in the alignment path. These definitions can also be extended for protein B.

The first distance measure is used to evaluate the combination of two AFPs, one currently in the alignment path and the other to be added depending on the distance value. The second distance measure serves as a way to evaluate an AFP, that is to say how well two fragments that form an AFP match each other. Lastly, the third is used as the last step to select the few best alignments and optimize gaps in the final alignment.

In order to extend the alignment path, the authors also suggest three different ways which can be used to add AFPs:

1. Consider all possible AFPs that extend the path and satisfy the similarity criteria.
2. Consider only the best AFP which extends the path and satisfies the similarity criteria
3. Use an intermediate strategy.

The first approach uses an exhaustive combinatorial search for the optimal path, as opposed to the second one which only searches within the best paths.

In order to decide whether or not to extend a path, the following conditions and respective heuristics have been used:

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. The authors state that the most accurate alignment is obtained when the selection of the AFP and extension of the path is done in three steps: the decision of whether candidates AFP are fit or not for the alignment is based on the first condition; the best AFP is selected based on the second condition; and the decision of extending or terminating the path is based on the third condition.

Finally, after obtaining the longest alignment path, we now evaluate its statistical significance through its z-score, based on the RMSD distance and number of gaps.

2.2.2.4 Elastic Shape Analysis

Another different and unique approach is ESA [36] which is one of the most recent approaches to compare protein structures. The main difference when compared to other approaches is that ESA considers protein backbones as continuous 3-D curves. Thus, the alignment of two structures corresponds to the alignment of the two curves representative of their backbones. These curves, known in this framework as the Square Root

Velocity Functions (SRVF), are able to bend and stretch during the alignment in order for them to account for the variations between structures.

In this framework we consider a shape to be a similarity class of a curve in a Riemannian manifold. In order to obtain this similarity we require a distance metric between shapes, more specifically the geodesic path (length minimizing path), which corresponds to a curve's evolution with respect to a manifold geometry.

In short, the main idea of this approach is that we represent the protein backbones in a Riemannian manifold through SRVF functions and then find shortest path (curve evolution) from one curve to the other, with respect to the manifold's geometry.

Recently, an algorithm was developed based on this framework. The mathematical component of this algorithm is very complex and thus, better explained in [55]. A summarized version of the algorithms' steps is provided below:

1. Start by extracting the 3-D coordinates of the backbone to derive the initial input curve denoted by $P^{(j)}_{(3+k)\times n_j}$ for each protein j of length n_j. The superscript $j = 1$ and $j = 2$ identifies protein 1 and 2 respectively. The superscript $(3 + k)$ refers to the first x,y,z coordinates of atoms and k coordinates are supplementary auxiliary information.

2. Translate and scale by transforming the curves to their SRVF, represented by $Q^{(j)}_{(3+k)\times n_j}$.

3. Recompute the SRVF $Q^{(1)}_1$ and $Q^{(2)}_1$ corresponding to a new T (piecewise linear function) for each dimension $(3 + k) \times n$.

4. Obtain the optimal rotation matrix through Singular Value Decomposition.

5. Achieve optimal matching by using dynamic programming.

6. Calculate the geodesic distance between the curves.

As far as performance is concerned, ESA based algorithms seem to perform better than others. Despite seeming too complex mathematical wise, these approaches greatly reduce computation time, however, no known implementation was found despite this being the most promising algorithm.

2.3 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 instances of these scores. It worth mentioning though that other measures have been developed besides the ones listed in this section and are also to be experimented with, such as the Contact Map Overlap [4] and MaxSub [52].

2.3.1 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 1Åto 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 RSMD 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.

To tackle some of these issues, an extension of RMSD was introduced: the weighted RMSD (wRMSD).

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

This allows us to focus on specific subsets of atoms and control their weights w, on the calculation and thus diminishing the impact of regions known to be dissimilar.

2.3.2 Global Distance Test

An alternative to using RMSD, is the more efficient GDT [58]. This is a procedure that also serves as a way to evaluate protein structure alignments and it is mostly used in cases where the two proteins have a very similar amino acid sequence and different structure. With this algorithm, 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 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 score for two structures is calculated as an average of calculations with different distance cutoffs [46]. For instance:

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

where the $maxC_x$ notation designates the maximum number of atom pairs under a distance cutoff of x Å. The higher the values of GDT, the higher is the similarity between two structures.

2.3.3 Template Modeling Score

Another measure of similarity between two protein structures is the TM-score [59]. 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, and L_{ali} is the number of the equivalent residues in the two proteins. d_i is the distance of the i-th pair of the equivalent residues between two structures and d_0 is given by:

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

2.4 Machine learning

Machine learning [37] [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, we are trying to find similarities among proteins through their structures, however the available datasets may not include information regarding this subject, therefore we are working with unlabeled data and consequently, unsupervised learning.

2.4.1 Clustering

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.

One particularly important feature of these algorithms is the ability to detect outliers, also known as noise, in our dataset. This will be useful for instance when clustering structure prediction models, in which there may be several predicted structures that can be discarded due to their greater lack of similarity.

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.

2.4.1.1 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.

Affinity propagation

In an attempt to solve the inconveniences of having to specify the number of clusters before runtime, the Affinity Propagation [17] 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.

2.4.1.2 Hierarchical clustering

These algorithms build clusters by either merging smaller clusters or dividing bigger ones, these 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 [24]. The result of either agglomerative or divisive clustering tends to be a dendrogram or a tree shaped view that displays the data hierarchy.

This is useful for this work since it will provide us the required views and information to make an evolutionary study of proteins or to find the closest protein structure when trying to infer protein functionality.

At each iteration of an agglomerative algorithm, we need to select the two closest groups to be merged, using some distance measure. Some examples of possible criteria are:

- **Single-linkage**: join the clusters that have the shortest distance between a pair of their members.
- **Complete linkage**: join the clusters that have the longest distance between a pair of their members.

Bisecting *k*-Means is an algorithm that fits this category, more specifically, the divisive approach. It uses a combination of the k-means algorithm and hierarchical clustering. We begin this algorithm by choosing a cluster to split. Next up is the bisecting step, on which we find two sub-clusters using the k-Means algorithm with $k = 2$. The bisecting step is repeated for a number of iterations and takes the split that produces clusters with the highest overall similarity. This whole process is repeat until we reach the desired number of clusters.

2.4.1.3 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 [24] assigns to each data point a degree of membership to every cluster.

In this work, 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.

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, 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.

2.4.1.4 Density-based clustering

In these approaches, clusters are formed based on higher density zones of the dataset. Other more scattered elements from our data may be considered noise.

For this work, 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.

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) [16] 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 database 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.

The authors of the article mention an heuristic to determine the values for parameters MinPts and Eps. These values are set to those of the least dense cluster in the data set. Since they represent the lowest density that is not considered to be noise, it is effective to use them as global parameters.

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.

2.4.1.5 Shared Nearest Neighbors

As we have established, there are some challenges that arise when using clustering algorithms: *k*-Means does not do well with clusters of different sizes or non-globular shapes, DBSCAN fails when the dataset has varying densities or high dimensionality, etc.

The Shared Nearest Neighbors (SNN) algorithm is another approach that addresses a lot of these issues. It uses the Jarvis-Patrick [26] approach to define the similarity between a given pair of points, which takes into account how many nearest neighbors they have in common. This definition removes the problems with varying densities. Furthermore, the algorithm identifies and builds clusters around core points. These clusters also deal with noise, by grouping only data from regions of uniform density. One other feature of this algorithm, is that it finds clusters that other approaches overlook, such as those in lower density regions and also single element clusters.

This algorithm can be used in functionality prediction through its concept of neighborhood, since we may clearly identify other structures that are most similar to the one whose function is unknown. It may also be used in protein docking in order to group the predicted complexes that are most similar to the target.

Since one of the cores of SNN belongs to the Jarvis-Patrick algorithm, a brief description of it is provided below:

1. Start by calculating the set of k nearest neighbors for each data point.
2. Then, merge points to form clusters if they share at least K_{min} of their nearest neighbors.

To determine the nearest neighbors, a distance cutoff is required, such as the RMSD of two structures.

As stated in [15], the SNN algorithm goes as follows:

1. **Compute the similarity matrix.** (This corresponds to a similarity graph with data points for nodes and edges whose weights are the similarities between data points.)
2. **Sparsify the similarity matrix by keeping the *k* most similar neighbors.** (This corresponds to only keeping the k strongest links of the similarity graph.)
3. **Construct the shared nearest neighbor graph from the sparsified similarity matrix.** At this point, we could apply a similarity threshold and find the connected components to obtain the clusters (Jarvis-Patrick algorithm.)
4. **Find the SNN density of each point.** Using a user specified parameters, Eps, find the number points that have an SNN similarity of Eps or greater to each point. This is the SNN density of the point.
5. **Find the core points.** Using a user specified parameter, MinPts, find the core points, i.e., all points that have an SNN density greater than MinPts.
6. **Form clusters from the core points.** If two core points are within a radius, Eps, of each other, then they are placed in the same cluster.
7. **Discard all noise points.** All non-core points that are not within a radius of Eps of a core point are discarded.
8. **Assign all non-noise, non-core points to clusters.** We can do this by assigning such points to the nearest core point.

It worth mentioning that this process also uses what is essentially the DBSCAN algorithm, in steps 4 through to 8. Thus, the parameters MinPts and Eps are determined according to the non-noise least dense cluster.

In regards to proteins, we can use this technique to find the nearest neighbors of a given protein, which correspond the ones that are most similar to it.

2.4.2 Cluster evaluation

Several authors have studied and applied cluster techniques, however, when it comes to what a good cluster looks like, it seems that they do not reach a consensus. Since clustering can be applied to a vast range of problems, good clusters vary with the problem at hand.

Despite this, some evaluation criteria have been developed and usually fit one of two categories, internal and external [13] [48].

2.4.2.1 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. In this case, it will allow us to see if the proteins that belong to the same clusters are similar to one another.

- **Sum of Squared Errors (SSE)**: its value is determined by the following calculation:

$$SSE = \sum_{k=1}^{K} \sum_{\forall x_i \in C_k} \| x_i - \mu_k \|^2$$

  where C_k represents the set of instances of cluster k and μ_k is the mean vector of cluster k. The components of μ are determined according to:

$$\mu_k = \frac{1}{N_k} \sum_{\forall x_i \in C_k}$$

  In this calculation, N_k corresponds to the size of the set of instances of cluster k.

  We can interpret the SSE as the measure of total error obtained by representing the N_k instances $x_1, ..., x_n$ by the cluster centers $\mu_1, ..., \mu_k$. The value of SSE varies according to the distribution of instances among clusters. In this criteria, the optimal partitioning is the one that minimizes the SSE, hence the closer the value is to zero, the better clusters we have.

- **Related minimum variance criteria**

  It is possible to simplify the previous SSE expression in order to eliminate the mean vectors from the equation, resulting in:

$$SSE = \frac{1}{2} \sum_{i=1}^{c} n_i \bar{s}_i$$

  where $\bar{s}_i$ can be replaced by the average, the median or the maximum distance between points in a cluster.

- **Scattering criteria**

  There is also a set of criterion functions which can be obtained from scatter matrices. The total scatter matrix is given by:

$$S_T = S_W + S_B$$

  where S_W is the within-cluster scatter matrix and S_B the between-cluster scatter matrix. Their values are given by:

$$S_W = \sum_{k=1}^{K} \sum_{x \in C_k} (x - \mu_k)(x - \mu_k)^T$$

$$S_B = \sum_{k=1}^{K} N_k (x - \mu_k)(x - \mu_k)^T$$

  This method is convenient for clustering because while trying to minimize the within-cluster scatter the between-cluster scatter tends to be maximized. In other words, instances from the same clusters will be more similar among each other and dissimilar when compared to instances of different clusters. Since we are analyzing the amount of scatter in matrices, we need a scalar measure for this. Three scalar criteria may be derived from S_W,S_B and S_T.

  - **The trace criterion**: is one of the simplest scalar measures for scatter matrices. It corresponds to the sum of the diagonal elements of a given matrix. Minimizing the trace of the within-cluster scatter matrix is similar to minimizing the SSE and as such, it is an acceptable criterion:

$$tr[S_W] = \sum_{k=1}^{K} \sum_{\forall x_i \in C_k} \| x_i - \mu_k \|^2$$

    One other criterion similar to these, is the one that maximizes the between-cluster criterion:

$$tr[S_B] = \sum_{k=1}^{K} N_k \| \mu_k - \mu \|^2$$

  - **The determinant criterion**: in this case, we use the determinant of the scatter matrix, which roughly measures the square of the scattering volume, since it is proportional to the product of the variances in the directions of the principal axes. If we assume that S_W is nonsingular, the determinant criterion may be used:

$$|S_W| = |\sum_{k=1}^{K} S_k|$$

where S_k is the scatter matrix for the k^{th} cluster and is calculated by:

$$S_k = \sum_{x \in C_k} (x - \mu_k)(x - \mu_k)^T$$

If S_W is singular, then the determinant criterion is not an appropriate one.

– **The invariant criterion**: the eigenvalues $\lambda_1, ..., \lambda_m$ of

$$S_W^{-1} S_B$$

are the basic linear invariants of scatter matrices. Their values measure the ratio of between-cluster to within-cluster scatter in the direction of the eigenvectors. Good data partitions, are the ones for which the nonzero eigenvalues are large.

- **Silhouette score**: [49] is a graphical display which provides information 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.

2.4.2.2 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 [48]. 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. Similarly, the Uniprot database has information regarding domain functionality, which can be used to check if the proteins in our clusters share the same functionality as the protein whose function we are attempting to predict. Lastly, using data from the GWIID database, in the case

of docking we can simply compare the groups of modeled structures against the target complex, since they are supposed to have the same structure.

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:

- **Precision-Recall**: these measures represent how relevant and how complete the clustering result is, respectively. They are calculated by:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

  where TP represents the true positives, FP the false positives and FN the false negatives.

- **F-Measure**: is a combination of both Precision and Recall which represents their harmonic mean:

$$F - Measure = 2 * \frac{Precision * Recall}{Precision + Recall}$$

- **Rand index**: is a measure of similarity between two data partitions. Let C_1 be defined as the result of our clustering process and C_2 a given clustered structure. Furthermore, a is the number of pairs of instances that are assigned to the same cluster both in C_1 and C_2; b is the number of pairs of instances that are assigned to the same cluster in C_1 but not in C_2; c is the number of pairs of instances that are in the same cluster in C_2, but not in the same cluster in C_1; and d is the number of pairs of instances that are assigned to different clusters in C_1 and C_2.

$$RAND = \frac{a + d}{a + b + c + d}$$

  The result ranges from 0 to 1, with the latter indicating that C_1 and C_2 match perfectly.

Chapter 3

Data and tools

This chapter will provide a brief description of the available tools that will be used to developed the required code and the platforms that hold valuable data for this work.

3.1 Development tools

Scikit-learn

Clustering algorithms are one of the main aspects of this work, which is why the Scikit-learn [45] library will be 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. By using this library, the time spent developing this work is expected to be significantly reduced.

SciPy

When applying machine learning algorithms to a given dataset we usually have the need to manipulate its data so it can be duly processed. Furthermore, in this specific work there are some algebraic and vectorial calculations that may be needed. For these tasks we may use SciPy [27], which is another free Python library developed for scientific computations.

Matplotlib

Data visualization is an important part of machine learning since we need a way to see the results of the algorithms we are using. To this end, the Matplotlib [23] library for Python may be used. It allows us to generate 2-D histograms, bar charts, scatter plots, etc.

3.2 Protein data

CAPRI

The Critical Assessment of PRedicted Interactions (CAPRI) [25] 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. Thus, the data on this platform can be used in order to cluster protein docking models according to the target information.

CATH

The CATH [53] 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, which is information that can be used to test the accuracy of the results of clustering algorithms.

GWIDD

The Genome-wide docking database (GWIDD) [33] is an integrated resource for structural studies of protein–protein interactions. This platform stores experimental and modeled 3-D structures obtained by docking techniques which are freely available and formatted with accordance to the PDB file format. We can use this data to cluster the modeled structures and evaluate them with respect to the target complexes.

PDB

The Protein Data Bank (PDB) [5] 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 (such as GWIDD), which is available in PDB format and includes information such as atomic coordinates, molecule names, primary and secondary structures.

Its data may be valuable in the sense that it may be clustered and then compared with other databases' classifications.

SCOP

The Structural Classification of Proteins (SCOP) [39] 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.

Uniprot

Uniprot [9] is a collection of resources regarding protein information. Its composed of three databases that collectively store the different types of information of this platform:

- Uniprot knowledgebase (UniprotKB): is the central hub for the collection of functional information on proteins. Each entry contains detailed information on each protein, such as their amino acid sequence, protein name and taxonomy.

- Uniprot reference clusters (UniRef): contains clustered sets of protein sequences originating from UniprotKB.

- Uniprot archive (UniParc): stores most of the available and known protein sequences.

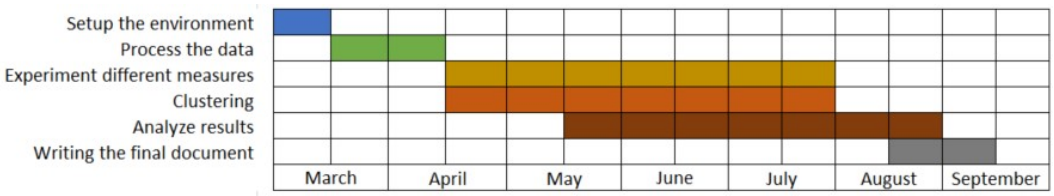
The data from UniprotKB may prove to be useful for this work when attempting to predict a protein's function based on its structure.

Chapter 4

Work plan

This chapter contains the proposed approach along with an estimate time line for task completion.

4.1 Proposed approach



Setup the environment

As was mentioned in the previous chapter, the Scikit-learn library is a powerful tool that contains implementations of many clustering algorithms, which when used with Matplotlib (data visualization) and SciPy (data processing) makes it easier for us to produce and analyze results. The libraries were developed for the Python programming language and as such, it will be the preferred choice for this work. Initially, a short amount some time will be spent adapting to the Python language and learning how to use the mentioned libraries. This will mostly consist in thoroughly reading the respective documentation of the libraries in order to make the most of them.

Process the data

Next up, the protein structure data must be acquired from the previously mentioned platforms so it can be processed. Specifically, datasets from SCOP and CATH will be used to cluster data that will allow phylogenetic analysis, GWIID and CAPRI can provide protein docking models which can be grouped according to their quality and lastly, with

data from Uniprot and CATH we can cluster structures with the goal of inferring protein function.

Each of these databases have their own data representation, whose descriptions are duly documented in the respective platforms. This means that before using the data, there is the need to further read the documentation in order to understand the different formats.

Experimenting different measures

Once the data is obtained, we must implement and experiment with the similarity measures, starting with the RMSD, GDT and TM-Score (described in section 2.3).

To accomplish this, we need to calculate the similarity matrices for the different datasets, which can be done through the implementation use of the jCE web server made available by the PDB platform.

Clustering

As we obtain the required similarity matrices, according to the different measures, we may also start the clustering process and see how the structures are grouped. At this stage we can expect a wide range of results, originated from combinations of clustering algorithms and structure similarity measures.

The first case that is to be explored is the grouping of complexes generated by docking techniques. This is one of the simplest cases since all these models have the same amino acid sequence, due to being the same proteins but in different positions, orientations, etc. For this case, partitional, fuzzy and density-based algorithms seem the most efficient ones and thus will be the first ones to be applied in this application.

Following this, one other interesting case is the grouping of structures in order to better understand their evolution. In this application, there is more variety both in sequences and in structures which in turn implies that there may be several possible groupings of structures, however an hierarchical approach may be the most informative one and will be one of the first to be tested in this case.

In the last case, we will cluster proteins structures in an attempt to predict protein functionality. As with evolutionary relationships, its likely that a lot of diversity will be found when grouping structures which opens a lot of possibilities to explore with clustering algorithms. However, in this application we will start by using partitional clustering algorithms since these form non-overlapping clusters, which seem the most indicated to infer functionality.

Analyze results

When the clustering process starts producing results, we can also start evaluating the quality of the resulting groups and in turn the different measures used. This evaluation is based on the criteria mentioned in section 2.4.2 and the databases previously described.

Writing the final document

The completing step to this work shall be the documentation of the developed code as well as a discussion of the obtained results.

Bibliography

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

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. "Basic local alignment search tool." In: *Journal of molecular biology* 215.3 (1990), pp. 403–410.

[3] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs." In: *Nucleic acids research* 25.17 (1997), pp. 3389–3402.

[4] R. Andonov, N. Malod-Dognin, and N. Yanev. "Maximum contact map overlap revisited." In: *Journal of Computational Biology* 18.1 (2011), pp. 27–41.

[5] H. Berman, J Westbrook, Z Feng, G Gilliland, T. Bhat, H Weissig, I. Shindyalov, and P. Bourne. "The Protein Data Bank Nucleic Acids Research, 28, 235-242." In: *URL: www. rcsb. org Citation* (2000).

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

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

[8] R. Chenna, H. Sugawara, T. Koike, R. Lopez, T. J. Gibson, D. G. Higgins, and J. D. Thompson. "Multiple sequence alignment with the Clustal series of programs." In: *Nucleic acids research* 31.13 (2003), pp. 3497–3500.

[9] U. Consortium. "UniProt: a hub for protein information." In: *Nucleic acids research* 43.D1 (2014), pp. D204–D212.

[10] 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.

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

[12] O. Dror, H. Benyamini, R. Nussinov, and H Wolfson. "MASS: multiple structural alignment by secondary structures." In: *Bioinformatics* 19.suppl_1 (2003), pp. i95–i104.

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

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

[15] L. Ertöz, M. Steinbach, and V. Kumar. "Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data." In: *Proceedings of the 2003 SIAM International Conference on Data Mining*. SIAM. 2003, pp. 47–58.

[16] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *Kdd*. Vol. 96. 34. 1996, pp. 226–231.

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

[18] 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.

[19] I. Halperin, B. Ma, H. Wolfson, and R. Nussinov. "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.

[20] 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.

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

[23] 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.

[24] 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.

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

[26] R. A. Jarvis and E. A. Patrick. "Clustering using a similarity measure based on shared near neighbors." In: *IEEE Transactions on computers* 100.11 (1973), pp. 1025–1034.

[27] E. Jones, T. Oliphant, P. Peterson, et al. *SciPy: Open source scientific tools for Python*. 2001–. URL: http://www.scipy.org/.

[28] 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.

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

[30] A. S. Konagurthu, J. C. Whisstock, P. J. Stuckey, and A. M. Lesk. "MUSTANG: a multiple structural alignment algorithm." In: *Proteins: Structure, Function, and Bioinformatics* 64.3 (2006), pp. 559–574.

[31] 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.

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

[33] P. J. Kundrotas, Z. Zhu, and I. A. Vakser. "GWIDD: genome-wide protein docking database." In: *Nucleic acids research* 38.suppl_1 (2009), pp. D513–D517.

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

[35] D. J. Lipman and W. R. Pearson. "Rapid and sensitive protein similarity searches." In: *Science* 227.4693 (1985), pp. 1435–1441.

[36] W. Liu, A. Srivastava, and J. Zhang. "A mathematical framework for protein structure comparison." In: *PLoS computational biology* 7.2 (2011), e1001075.

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

[38] 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.

[39] A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. "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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

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

[45] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

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

[47] 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.

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

[49] 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.

[50] 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.

[51] 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.

[52] N. Siew, A. Elofsson, L. Rychlewski, and D. Fischer. "MaxSub: an automated measure for the assessment of protein structure prediction quality." In: *Bioinformatics* 16.9 (2000), pp. 776–785.

[53] I. Sillitoe, T. E. Lewis, A. Cuff, S. Das, P. Ashford, N. L. Dawson, N. Furnham, R. A. Laskowski, D. Lee, J. G. Lees, et al. "CATH: comprehensive structural and functional annotations for genome sequences." In: *Nucleic acids research* 43.D1 (2014), pp. D376–D381.

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

[55] S. Srivastava, S. B. Lal, D. Mishra, U. Angadi, K. Chaturvedi, S. N. Rai, and A. Rai. "An efficient algorithm for protein structure comparison using elastic shape analysis." In: *Algorithms for Molecular Biology* 11.1 (2016), p. 27.

[56] 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.

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

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

[59] 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.

[60] 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.

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