Pedro Miguel Franco Arguelles

Bachelor in Computer Science 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

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

March, 2019

Clustering of protein structures

This document was created using the (pdf)LaTeX processor, based in the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

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

List of Figures

Listings

Acronyms

AFP	Aligned Fragment Pairs.
CASP	Critical Assessment of protein Structure Prediction.
CATH	Class, Arquitecture, Topology and Homologue superfamily protein structure classification database.
CE	Combinatorial Extension.
DBSCAN	Density-based spatial clustering of applications with noise.
GDT-HA	Global Distance Test - High Accuracy.
GDT-TS	Global Distance Test - Total Score.
MaxSub	Maximum Subset.
PDB	Protein Data Bank.
RMSD	Root Mean Square Deviation.
SCOP	Structural Classification of Proteins database.
SNN	Shared Nearest Neighbors.
TM-score	Template Modeling score.

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 [12], 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 [4] [10].

- **Docking**

  Another application where it is necessary to group and compare structures is protein docking [8]. 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 Square 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 [13]. 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, the Class, Arquitecture, Topology and Homologue superfamily protein structure classification database (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 [11].

Given the issues and usefulness of comparing and grouping protein structures, with this work we aim to:

- Explore ways to counter the drawbacks of representing protein structure similarity through a single measure by adding a complementary one.
- Experiment clustering algorithms in order to group protein structures and determine which one of them is more adequate for this task.
- Assess which structure similarity measure combinations produce the best clusters for the different algorithms.

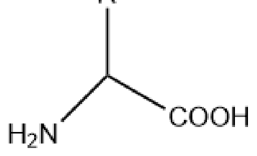
Chapter

2 Proteins

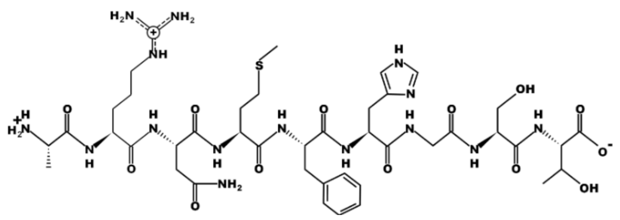
This chapter provides a brief description of the core protein concepts for this work.

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 [3]. In Figure 2.1 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 2.1

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 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 [17].

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 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 [16].

Chapter 3

State of the art

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

3.1 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.

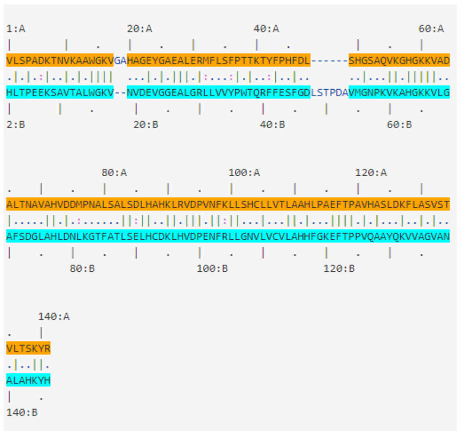
- The proteins are not similar in sequence but are in structure: once again, this case may appear when grouping proteins with higher evolutionary distances. As time passes, sequences become more prone to mutations whereas structure tends to be preserved.

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.

3.1.1 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 3.1.



```
1:A                20:A                40:A                      60:A
|        |    .    |    .    |    .    |    .          |    .    |
VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDL------SHGSAQVKGHGKKVAD
.|.|.:|..|.|.||||  ...|.|.|||.|:...:|.|..:|..|..      ..|...||.|||||..
HLTPEEKSAVTALWGKV--NVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLG
|       |    .      |    .    |    .    |    .    |    .    |    .
2:B                 20:B                40:B                60:B


               80:A                100:A               120:A
.    |    .    |    .    |    .    |    .    |    .    |    .    |
ALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVST
|.....||.|.:......||:||..||.|||.||:||...|...||.|...||||.|.|...|..|.|..
AFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVAN
|    .    |    .    |    .    |    .    |    .    |    .    |    .
          80:B                100:B               120:B


     140:A
.    |
VLTSKYR
.|..||.
ALAHKYH
|    .
140:B
```

Figure 3.1: 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.

So far, there are a lot of algorithms and software that have been developed for sequence alignment. A few examples are FASTA [14], BLAST [1] and PSI-BLAST [2] for database searches, PyMol [7] (supports commands for sequence alignments) for pairwise alignments and ClustalW [5] 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) [6] and BLOSUM (Blocks Substitution Matrix) [9]. 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.

3.1.1.1 Needleman-Wunsch

For global alignment, the Needleman-Wunsch [15] 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.

3.1.1.2 Smith-Waterman

For local alignment, the Smith-Waterman [**smith1981identification**] 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.

3.1.2 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 3.2 shows us an example of a structural alignment.

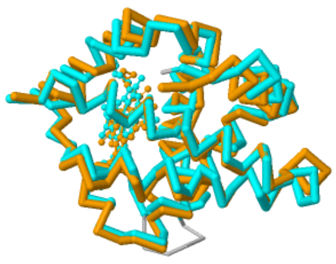


Figure 3.2: Example of a structure alignment between 4HHB.A and 4HHB.B

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 [**mosca2008alignment**], VAST [**gibrat1996surprising**], MASS [**dror2003mass**], SSM [**krissinel2003protein**] and DALI [**holm1993protein**]. Another possible approach that FAST [**zhu2005fast**] and SABERTOOTH [**teichert2007sabertooth**] explore, is obtaining a structural alignment based on pairwise residue distances. Furthermore, there are also multiple structural alignment methods such as: CBA [**ebert2006development**], POSA [**ye2005multiple**], MultiProt [**shatsky2004method**], MALECON [**ochagavia2004progressive**] and MUSTANG [**konagurthu2006mustang**].

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 on how they work.

3.1.2.1 Superposition based on Kabsch's algorithm

This algorithm allows us to compare two proteins by directly superimposing the structures [**kabsch1976solution**]. 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 [7].

The steps of the algorithm have a fairly complex mathematical component, which is more thoroughly explained in [4], 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 RMSD.

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

3.1.2.2 Sequential Structure Alignment Program -RESUMIR

The SSAP [**orengo199636**] 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 [**sillitoe2014cath**] 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.

3.1.2.3 TM-Align

One of the most standard algorithms for these types of alignments is TM-Align [**zhang2005tm**]. 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. The final assignment for its secondary structure is then obtained by merging and removing isolated states.

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.

3.1.2.4 MAMMOTH

Matching molecular models obtained from theory (MAMMOTH) [**ortiz2002mammoth**] 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 in the sections ahead.

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 of 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.

3.1.2.5 CE -variaveis -resumir

One other method of obtaining a structural alignment is using Combinatorial Extension (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 Protein Data Bank (PDB) [**berman2000protein**], 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 [**shindyalov1998protein**].

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.

Furthermore, 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.

3.1.2.6 Elastic Shape Analysis

Another different and unique approach is ESA [**liu2011mathematical**] 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 [**srivastava2016efficient**]. 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.

3.2 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 instances of these scores.

3.2.1 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.

3.2.2 Global Distance Test - Total Score

An alternative to using RMSD, is the more efficient Global Distance Test - Total Score (GDT-TS) [**zemla2003lga**]. 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-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 [**poleksic2009algorithms**]. 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 10Åto 20Å, whereas the values for related structures are much higher. [**herbert2008maxcluster**]

3.2.3 Global Distance Test - High Accuracy

As the name indicates, the Global Distance Test - High Accuracy (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:

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

3.2.4 Template Modeling Score

Another measure of similarity between two protein structures is the Template Modeling score (TM-score) [**zhang2004scoring**]. 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:

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

3.2.5 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.

3.3 Machine learning

Machine learning [**tommitchell**] [**ethemalpaydin2010**] 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.

3.3.1 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.

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.

3.3.1.1 Partitional clustering - CITE KMEDOIDS

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** [**ethemalpaydin2010**] 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.

Affinity propagation

In an attempt to solve the inconveniences of having to specify the number of clusters before runtime, the Affinity Propagation [**frey2007clustering**] 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:

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

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.

3.3.1.2 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 [**jain1999data**]. 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, which is the one used for this work, we need to select the two closest groups to be merged, using some distance measure. A few examples of possible distances are:

- **Single-linkage**: the shortest distance between two points in each cluster.
- **Complete-linkage**: the longest distance between two points in each cluster.

- **Average-linkage**: the average distance between each point in one cluster to every point in the other.

3.3.1.3 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 [**jain1999data**] 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.

3.3.1.4 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) [**ester1996density**] 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.

3.3.1.5 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 [**jarvis1973clustering**] 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 [**ertoz2003finding**], 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.

3.3.2 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 [**duda2012pattern**] [**rokach2005clustering**].

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

- **Calinski-Harabasz score:** this score represents the ratio between the within cluster dispersion and the between cluster dispersion. This score is given 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. [**calinski1974dendrite**] [**scikitlearn**]

- **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.
  - $b(i) = \min\limits_{C \neq A} d(i,C)$

  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. [**rousseeuw1987silhouettes**] [**scikitlearn**]

3.3.2.2 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 [**rokach2005clustering**]. 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.

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**: Similar to the previous metric, AMI also returns 1 if the two provided input clusterings are in complete accordance with each other and 0 or a negative value if its the complete opposite situation. The difference is that AMI accounts for chance and is biased towards clusterings with a higher amount of clusters.

  $$ARI(U,V) = \frac{MI(U,V) - E\{MI(U,V)\}}{max\{H(U), H(V)\} - E\{MI(U,V)\}}$$

  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. A more detailed description can be found in its original paper. [**vinh2010information**] [**scikitlearn**]

- **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 [**rosenberg2007v**] [**scikitlearn**]. 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:

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

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

- **Completeness**:

  Since homogeneity only focuses on assessing the quality of the data point clustering, we need a counterpart to it, in order to evaluate if all the available data points have been assigned to a cluster. This is essentially what completeness is: a measure that reflects the portion of the total data points that have correctly been assigned to a 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 [**rosenberg2007v**] [**scikitlearn**].

- **V-measure**:

  Is defined as the harmonic mean between the homogeneity and completeness measures:

  $$v = 2 \times \frac{h \times c}{h + c}$$

  Its value ranges from 0 to 1. [**rosenberg2007v**] [**scikitlearn**]

Chapter 4

Available data and tools

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

4.1 Alignment tools

4.1.1 BioPython

[**cock2009biopython**]

This is a library developed for the Python programming language which provides a wide range of utilities for protein data processing. It allows its users to parse PDB and SCOP files, for instance, and access more complex information regarding each protein, however, it seems this library is more oriented towards protein sequences. Nevertheless it may still be used as a tool for structural alignment and the subsequent RMSD calculation.

4.1.2 BioJava

[**holland2008biojava**]

This is the Java library that powers the jCE tool available on the PDB. It is a complex and powerful library that is able to load protein files and calculate both sequence and structural alignments. As far as structure similarity measures are concerned, RMSD and TM-score are available.

4.1.3 Clusco

[**jamroz2013clusco**]

Clusco is one of the most complete tools in the context of this work. It provides an easy way for its users to calculate structure alignments and cluster proteins according to the results. It can compute RMSD, GDT, TM-score, MaxSub and CMO for a given structural alignment. Despite this, it was not chosen as the main tool since initial experiments have shown that the input for the all against all comparison cannot exceed 10 structures.

4.1.4 jCE

[**prlic2010pre**]

Available for download or to be used online on the PDB website, the jCE tool is powered by the BioJava library and provides us with protein alignments and visualization tools. Even though each alignment generates a great deal of information, as far as structural similarity measures are concerned, only the RMSD is displayed.

4.1.5 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 one single file. [**herbert2008maxcluster**]

4.1.6 TM-Align and TM-Score

The creators of TM-Align algorithm and TM-score measure have also programmed two tools, which are freely available online and can be used to obtain protein alignments and structure similarity measures. Although these are both fast tools that calculate alignments, they have their drawbacks regarding this work's objective.

The TM-score program is one that also calculates RMSD, TM-score, MaxSub, GDT-HA and GDT-TS. It was initially considered for this work, however it is not meant to be used on proteins whose sequences are very different since according to the authors, it is limited to comparing models based on their given and known residue equivalence.

TM-Align on the other hand, may be employed to align proteins with different sequences but it is also limited by the fact that it only calculates RMSD and TM-score.

4.2 Languages and libraries

4.2.1 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 others. 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 [**DEAP_JMLR2012**].

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. [**Hunter2007**]

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. [**walt2011numpy**]

Scikit-learn

Clustering algorithms are one of the main aspects of this work, which is why the Scikit-learn [**scikitlearn**] 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.

4.2.2 R

As opposed to Python, R serves a more specific role, that of statistical computing and data visualization. It is a high-level programming language whose abstractions facilitate what in other languages would be a more complex task, such as loading data and plotting graphics.

DBSCAN

This is a clustering library [**hahsler2017dbscan**] which implements a collection of popular algorithms, most of them based on the density clustering approach. The most relevant algorithms present are: DBSCAN, HDBSCAN, OPTICS and SNN Clustering.

ppclust

Is yet another clustering library [**ppclust2018**], however the algorithms it implements are focused on probabilistic clustering. It features, among others, the Fuzzy C-Means algorithm.

Factoextra

This a data visualization tool [**kassambara2016factoextra**], which provides functions that allow the user to easily extract algorithms' summaries as well as plot said information.

4.3 Protein data

CAPRI

The Critical Assessment of PRedicted Interactions (CAPRI) [**janin2005assessing**] 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.

CASP

The Critical Assessment of protein Structure Prediction (CASP) [**moult1995large**] 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.

CATH

The CATH [**sillitoe2014cath**] 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.

PDB

The PDB [**berman2000protein**] 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.

SCOP

The Structural Classification of Proteins database (SCOP) [**murzin1995scop**] 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 [**uniprot2014uniprot**] 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.

Chapter 5

SCOP vs SCOP2 vs SCOPe - CITE IMAGE -CITE PAPERS

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 different 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 5.1 illustrates the standard process and the mentioned classification factors.

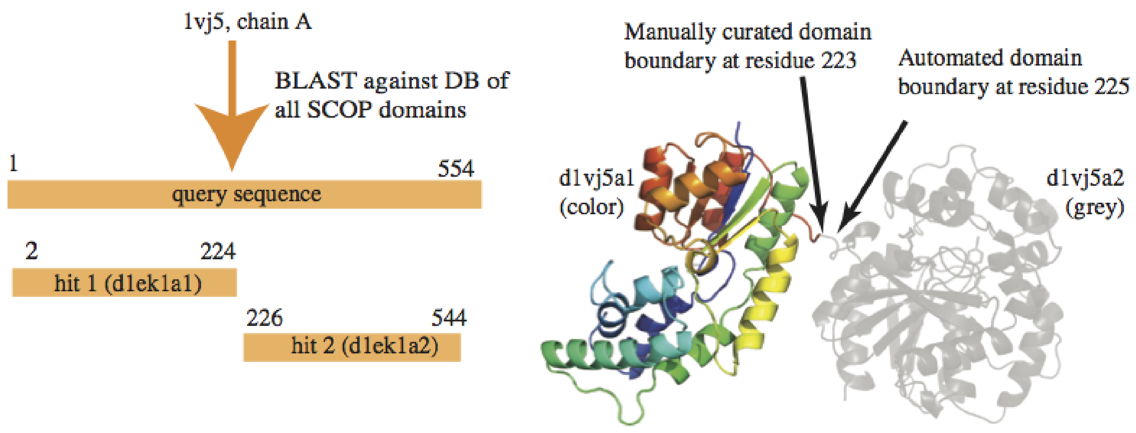

1vj5, chain A
BLAST against DB of all SCOP domains
1
554
query sequence
2
224
hit 1 (d1ek1a1)
226
544
hit 2 (d1ek1a2)
Manually curated domain boundary at residue 223
Automated domain boundary at residue 225
d1vj5a1 (color)
d1vj5a2 (grey)

Figure 5.1: Standard SCOPe classification process

Chapter 6

Sampling

Nowadays, the SCOPe database contains several thousands of domains (~274000) and thus it becomes an unfeasible task to obtain a complete similarity matrix, i.e. the matrix representation of the all-vs-all structure alignments, with the available hardware, since it would require the computation of millions of alignments. Given this limitation, there is the need to sample the data in a way that we can obtain a wide range of alignments in terms of their quality.

We want there to be a lot of different structures in our sample because this way the amount of matching residues between a given pair of proteins will vary a lot. This is a good thing because in extreme cases of similarity any single structure similarity measure suffices, since it will be fairly easy to identify residue correspondences, while in the case of extreme dissimilarity it will be harder for some of them to find any feint traces of homology, should they exist. It is the middle ground of these ranges that we want to study, since it is most likely where there will be relevant changes in the classification and also where the shortcomings of the measures will manifest themselves.

This is why the sample chosen must contain more intermediate values, because it is in these situations that the used similarity measures are most likely to misrepresent the differences between two structures and as such, it is where we will try to compensate with another measure.

With these restrictions in mind, a few samples were chosen from the SCOP hierarchy. The structures taken were chosen from specific classes and folds, for instance, a.1. would return every structure from every family and superfamily that was labeled with a class α and fold 1.

Chapter 7

Alignment computation

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. This piece of software is implemented in the C programming language which makes its runtime extremely low for single alignments. 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, once again, this feature is limited by hardware and as such the sample size must be restricted to a smaller amount.

Besides the structural alignments, the sequence alignments were also computed using ClustalW. Much like the previous program, there is also the possibility of calculating single and list alignments, however, the computation of the sequence alignments is much lighter than that of the structures.

Both types of files that were required, sequences and structures, needed to be parsed so that they could be used in the next phase. Specifically, most of the SCOP files contained duplicate atoms for some amino acids, which would cause Maxcluster to not be able to process the corresponding structure file. As far as sequences go, the amino acid chains needed to be processed into the FASTA format so that they could be loaded into ClustalW.

Chapter 8

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 input, and the pairwise similarities for each structure combination. Similarly, ClustalW also outputs the sequence identity percentage matrix, which has a different format than that of the Maxcluster files but needs to be processed all the same.

These resulting files were then processed in a way that allowed for the extraction of the mentioned matrices. This consisted of loading the data to 2D numpy arrays and symmetrizing them.

Once the symmetrical similarity matrix were obtained, the distance matrix followed. To accomplish this task, we assumed an Euclidean space which allowed us to calculate the distance between each pair of structures. From this, the distance matrix was obtained and the clustering phase began.

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 aim to combine different similarity measures through their respective distance matrices, there is the need to rescale the data.

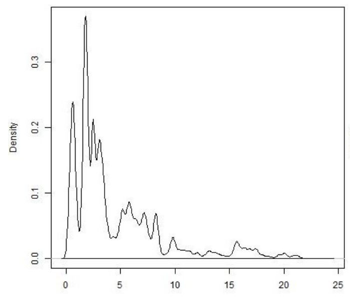
There are two main ways of achieving this, those being normalization and standardization, which are obtained by:

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

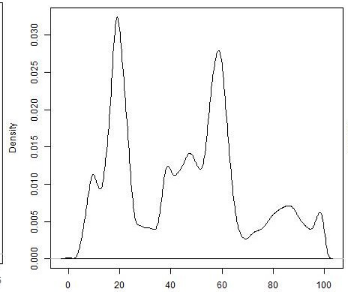
$Standardization: z = \frac{X - \mu}{\sigma}$

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

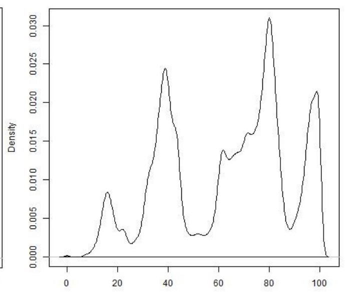
case, as we can see in the distributions for the *a*.1 sample, shown below in figure 8.1, normalization was chosen in order to rescale the data.



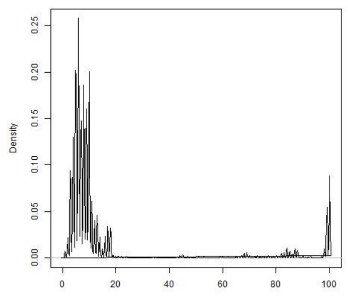
a RMSD distribution



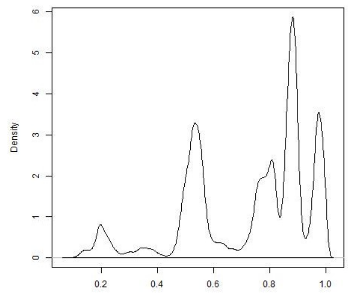
b GDT-HA distribution



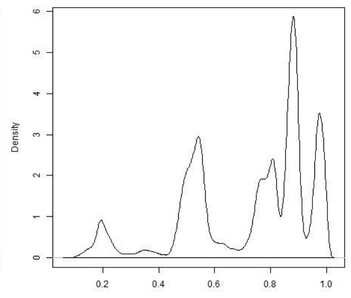
c GDT-TS distribution



d Sequence distribution



e TM-Score distribution



f Maxsub distribution

Figure 8.1: Kernel density estimation for the alignments of the *a*.1 sample

After obtaining the alignment matrices, we also needed to calculate the distances between points and thus, the Euclidean distance was chosen for this and it is calculated through the following equation:

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

Chapter 9

Structure clustering process

Having obtained the distance matrices, the clustering process was next. One of the algorithms that was considered was DBSCAN. Ultimately, it was not used as it was deemed too ineffective for the task at hand. This was due to the algorithm's approach to clustering, since its parameters are extremely hard to adjust to our data in the sense that it does not allow us to directly state how many clusters we wish to be formed. Besides the issue of estimating optimal parameters, there is also the issue that DBSCAN uses the density concept to cluster the data, which means that depending on its entry parameters there may be structures that are considered as noise and as such, will not be given a label which in turn implies that the final cluster evaluation will not take into account some of the input structures, thus hindering the experiment's results. The Affinity Propagation algorithm was also considered initially, however it was ruled out also due to its complex parameterization. This algorithm takes two numbers as input which are used to internally as rules in order to form clusters. Once again, the problem that comes from this type of approach is that the parameterization is attached to the shape of the data within each sample.

Given this, the studied algorithms were agglomerative and k-medoids. Both of these also receive distance matrices as input and one the most important aspects is that we can choose how many clusters we want the data to be split into. As opposed to DBSCAN, these algorithms also label every structure in our dataset. The main advantage of these algorithms in this work is that it allows us to cluster the data without minding its shape, and thus we can apply a broader approach to clustering that may be used regardless of the sample.

As for the experiment itself, we started by clustering the distance matrices of the mentioned similarity measures: RMSD, GDT-HA, GDT-TS, TM-Score and Max-sub. These provided us with the reference values (cluster evaluation metrics) that we

wished to optimize by combining said matrices with each other and them measuring their performance against the existing SCOP classification.

So far, we have established that there is a need to combine different structure similarity measures. However, this is not an easy task since theoretically we can have every number within the [0,1] range as weights for the considered measures. In an effort to mitigate this issue, only weights with two decimal numbers were considered. Having gotten past this, there was now a need to find the best combination of weights among the different measures, for which a genetic algorithm was used. This is a powerful tool that can be applied in optimization problems, as is the case.

The general steps of a genetic algorithm are the following:

1. Initialize population randomly
2. Evaluate the fitness of the population
3. Select the individuals from which the new generation will be formed
4. Crossover
5. Mutation
6. Repeat steps 2-5 for N generations or until convergence

For this particular problem, our individuals are made of decimal numbers, each of which corresponds to the weight that a given protein similarity measure will take during the clustering process. **As for fitness, the algorithm uses the AMI clustering evaluation metric, meaning that we are trying to optimize the weights with respect to the existing SCOPe classification in an attempt to obtain a set of labels as similar as possible to the existing one. After determining the individual whose weights generate the best AMI, we then calculate its internal clustering metrics. This way, we are guaranteed to obtain the weights that produce results that most closely resemble the existing classifications. It should be mentioned that it is to be expected that the external metrics should be lower than the base case, since we are using other similarity measures and comparing the results with others obtained with the previously mentioned process of automatic processes and manual curation. However, the internal metrics are expected to surpass those of the base classification, due to them not being attached to any labels and the potential that different structure similarity measures have of complementing each other's weaknesses.**

CHAPTER 10

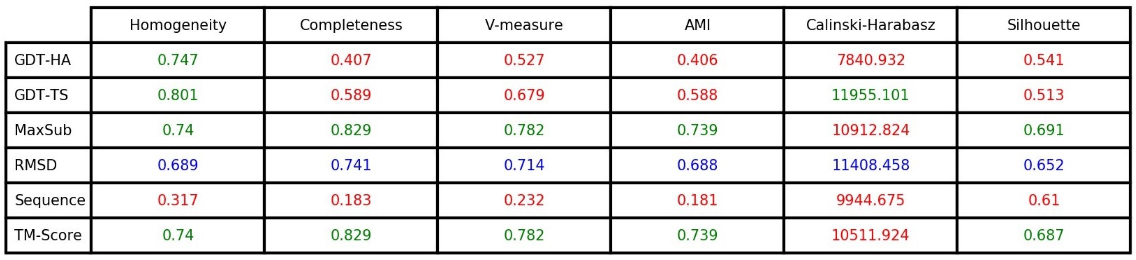
Results

In this chapter, the clustering results are presented along with a discussion regarding whether or not the results were as expected, if each algorithm is appropriate for this type of data and the which are structural similarity measures performed best.

10.1 Clustering with a single similarity measure

In the following sections, only the *a*.1 sample results are discussed and presented. Despite this, the results for the remaining samples are shown in the first annex of this document for which the following analysis and conclusions are also valid.

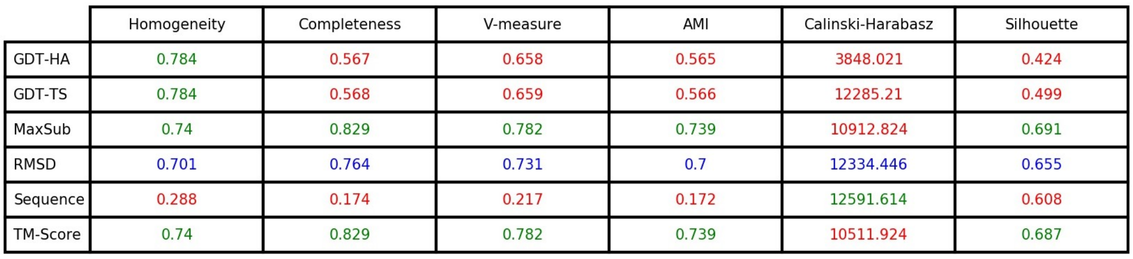
As was mentioned in the previous chapter, the clustering experiments began by obtaining the base clustering results, specifically, the metrics that originated from the different algorithms when applied to the individual distance matrices of the structure similarity measures.



	Homogeneity	Completeness	V-measure	AMI	Calinski-Harabasz	Silhouette
GDT-HA	0.747	0.407	0.527	0.406	7840.932	0.541
GDT-TS	0.801	0.589	0.679	0.588	11955.101	0.513
MaxSub	0.74	0.829	0.782	0.739	10912.824	0.691
RMSD	0.689	0.741	0.714	0.688	11408.458	0.652
Sequence	0.317	0.183	0.232	0.181	9944.675	0.61
TM-Score	0.74	0.829	0.782	0.739	10511.924	0.687

Figure 10.1: Single matrix hierarchical clustering with average linkage

The tables displayed in figures 10.1 and 10.2 above shows us the results obtained when applying the hierarchical clustering algorithm with average and complete linkage, respectively.



	Homogeneity	Completeness	V-measure	AMI	Calinski-Harabasz	Silhouette
GDT-HA	0.784	0.567	0.658	0.565	3848.021	0.424
GDT-TS	0.784	0.568	0.659	0.566	12285.21	0.499
MaxSub	0.74	0.829	0.782	0.739	10912.824	0.691
RMSD	0.701	0.764	0.731	0.7	12334.446	0.655
Sequence	0.288	0.174	0.217	0.172	12591.614	0.608
TM-Score	0.74	0.829	0.782	0.739	10511.924	0.687

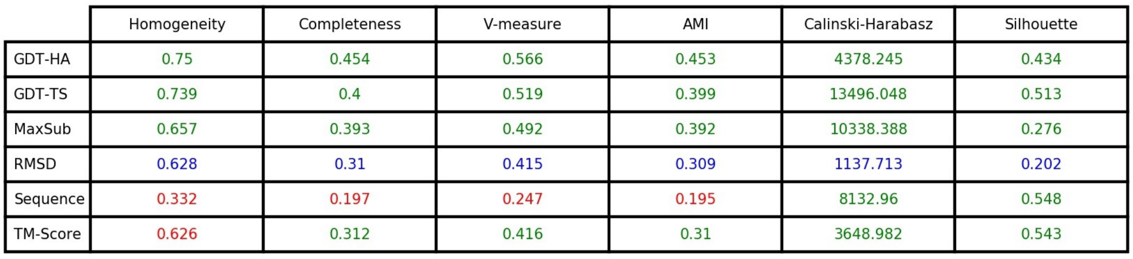
Figure 10.2: Single matrix hierarchical clustering with complete linkage

At this point, it was to be expected that at least the external metrics would perform worse for other similarity measures, since the base classification is through a set of automated processes and manual curation by experts. On the other hand, the internal metrics for the other measures could have performed better, since they are not tied to an already existing classification.

Nevertheless, for this particular experiment it seems that by themselves, Maxsub and TM-score are a cut above the rest since they are able to produce clusters with high similarities among their members while also maintaining a close resemblance to the SCOPe classification.

Furthermore, the clustering process was also applied to the sequences present in the sample. Though its results are poor according to the evaluation metrics, this is to be expected, because, as previously mentioned the samples were drawn at the superfamiliy level. This means that the proteins present in each sample share some commons ancestors among them, however the similarity traces that could indicate sequence homology have been lost during the evolutionary process. To illustrate this, we can check the kernel distribution for the sequence alignments in figure 8.1, which show us that most values range from 0% to 20%. It should be noted that even though the internal evaluation metrics are very favorable, due to the mentioned concentration, we must also consider the external ones, because it is not enough for the data points to be similar among themselves, they must also correctly be labeled.

On the other hand, the k-medoids algorithm produced some interesting results, as we can see in the table below:



	Homogeneity	Completeness	V-measure	AMI	Calinski-Harabasz	Silhouette
GDT-HA	0.75	0.454	0.566	0.453	4378.245	0.434
GDT-TS	0.739	0.4	0.519	0.399	13496.048	0.513
MaxSub	0.657	0.393	0.492	0.392	10338.388	0.276
RMSD	0.628	0.31	0.415	0.309	1137.713	0.202
Sequence	0.332	0.197	0.247	0.195	8132.96	0.548
TM-Score	0.626	0.312	0.416	0.31	3648.982	0.543

Figure 10.3: Single matrix *k*-medoids clustering

Although there are some favorable results in the cluster evaluation metrics, *k*-medoids seems to start to point out early on that it is not one of the best suited algorithms for this particular task. This is because even though it performs well when comes to the external metrics, internally it is a bit lackluster as evidenced by its low Silhouette and Calinski-Harabasz scores.

10.2 Clustering with a single similarity measure and sequences

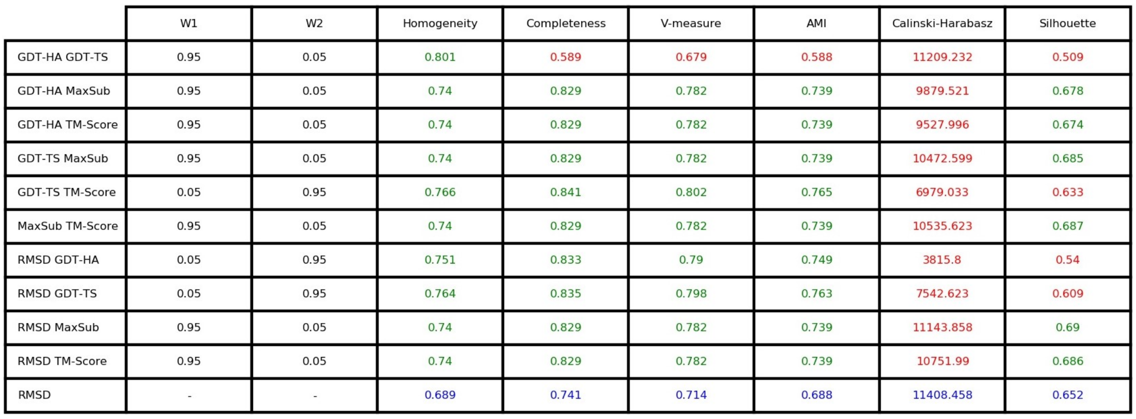
To reiterate, the SCOP database uses a mixture of both manual and automatic methods to group the present domains into its hierarchy which use RMSD. Not only does this classification use protein structures, it also uses their sequences to obtain the hierarchies' labels. In an attempt to replicate this process with the goal of improving the performance of the external clustering measures, each of the similarity measures were combined with the sequence alignments. In the table below, we can see the results of this process.

The obtained results by clustering with sequence were somewhat unexpected, as we hoped it would improve performance. The similarity measures' distance matrices were combined one by one with the sequence distance matrix by taking complementary percentages of both of them. After this process was finished, we went through the results in order to find which combination produced the highest cluster evaluation metrics. However, after analyzing the results we can clearly see that by taking any percentage of the sequence matrix into account only decreases performance, contrary to what was expected. Once again, this is mostly due to the fact that at the superfamily layer, proteins do not share many similarities sequence wise, hence, it is not enough to effectively cluster our data.

10.3 Clustering with two similarity measures

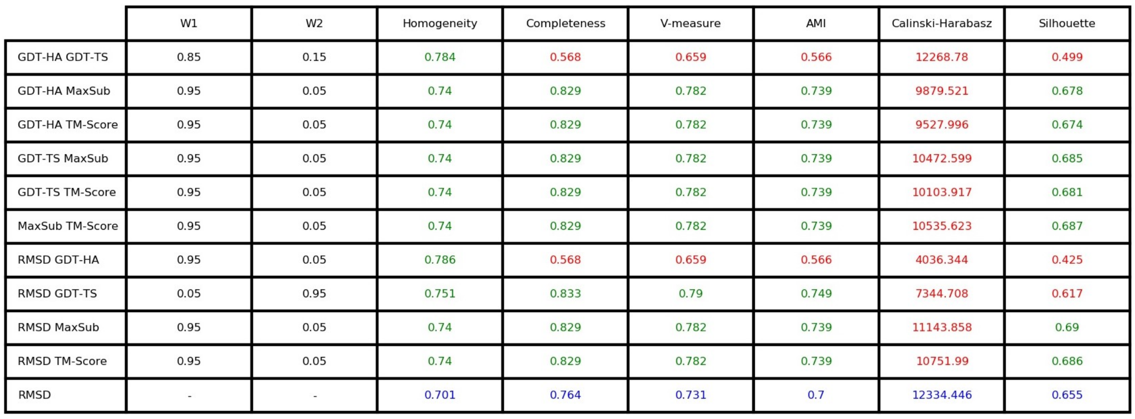
Since the use of sequences do not translate into an increase in performance, from this point on the experiences made were only based on the structure aspect of the proteins.

This next step of the experiment focused on one of the main goals of this work: to explore the combinations of similarity measures. In this phase of the experiments, it was expected that we could improve the internal clustering metrics, since we aim to counter the disadvantages of other similarity measures with other ones. At this stage, we were mainly hoping to improve RMSD performance, however, the results displayed in the following tables surpassed our expectations.



	W1	W2	Homogeneity	Completeness	V-measure	AMI	Calinski-Harabasz	Silhouette
GDT-HA GDT-TS	0.95	0.05	0.801	0.589	0.679	0.588	11209.232	0.509
GDT-HA MaxSub	0.95	0.05	0.74	0.829	0.782	0.739	9879.521	0.678
GDT-HA TM-Score	0.95	0.05	0.74	0.829	0.782	0.739	9527.996	0.674
GDT-TS MaxSub	0.95	0.05	0.74	0.829	0.782	0.739	10472.599	0.685
GDT-TS TM-Score	0.05	0.95	0.766	0.841	0.802	0.765	6979.033	0.633
MaxSub TM-Score	0.95	0.05	0.74	0.829	0.782	0.739	10535.623	0.687
RMSD GDT-HA	0.05	0.95	0.751	0.833	0.79	0.749	3815.8	0.54
RMSD GDT-TS	0.05	0.95	0.764	0.835	0.798	0.763	7542.623	0.609
RMSD MaxSub	0.95	0.05	0.74	0.829	0.782	0.739	11143.858	0.69
RMSD TM-Score	0.95	0.05	0.74	0.829	0.782	0.739	10751.99	0.686
RMSD	-	-	0.689	0.741	0.714	0.688	11408.458	0.652

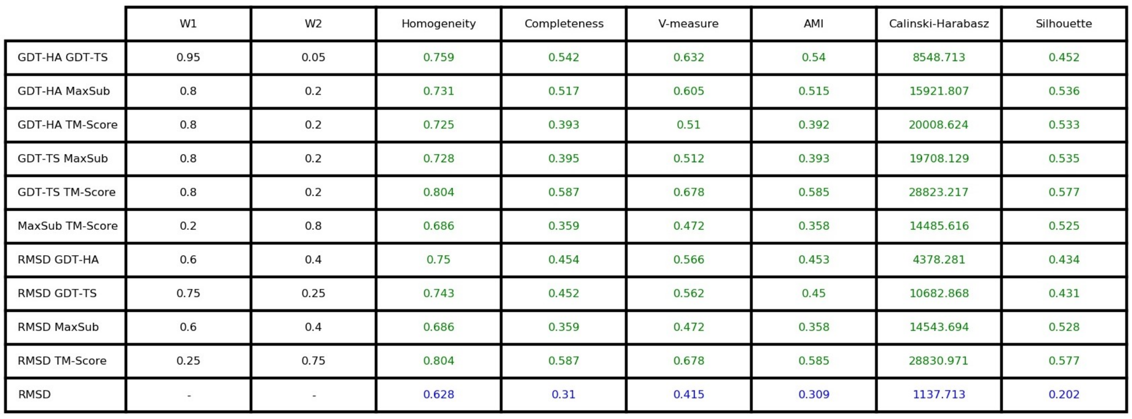
Figure 10.4: Hierarchical clustering results with average linkage for each combination of measures



	W1	W2	Homogeneity	Completeness	V-measure	AMI	Calinski-Harabasz	Silhouette
GDT-HA GDT-TS	0.85	0.15	0.784	0.568	0.659	0.566	12268.78	0.499
GDT-HA MaxSub	0.95	0.05	0.74	0.829	0.782	0.739	9879.521	0.678
GDT-HA TM-Score	0.95	0.05	0.74	0.829	0.782	0.739	9527.996	0.674
GDT-TS MaxSub	0.95	0.05	0.74	0.829	0.782	0.739	10472.599	0.685
GDT-TS TM-Score	0.95	0.05	0.74	0.829	0.782	0.739	10103.917	0.681
MaxSub TM-Score	0.95	0.05	0.74	0.829	0.782	0.739	10535.623	0.687
RMSD GDT-HA	0.95	0.05	0.786	0.568	0.659	0.566	4036.344	0.425
RMSD GDT-TS	0.05	0.95	0.751	0.833	0.79	0.749	7344.708	0.617
RMSD MaxSub	0.95	0.05	0.74	0.829	0.782	0.739	11143.858	0.69
RMSD TM-Score	0.95	0.05	0.74	0.829	0.782	0.739	10751.99	0.686
RMSD	-	-	0.701	0.764	0.731	0.7	12334.446	0.655

Figure 10.5: Hierarchical clustering results with complete linkage for each combination of measures

The results that were obtained with the hierarchical algorithm were very favorable. Even though, the weights for the different pairs of similarity measures vary across samples, we can see that we are able to complement RMSD with other measures in order to improve both internal and external clustering metrics. Besides this,



	W1	W2	Homogeneity	Completeness	V-measure	AMI	Calinski-Harabasz	Silhouette
GDT-HA GDT-TS	0.95	0.05	0.759	0.542	0.632	0.54	8548.713	0.452
GDT-HA MaxSub	0.8	0.2	0.731	0.517	0.605	0.515	15921.807	0.536
GDT-HA TM-Score	0.8	0.2	0.725	0.393	0.51	0.392	20008.624	0.533
GDT-TS MaxSub	0.8	0.2	0.728	0.395	0.512	0.393	19708.129	0.535
GDT-TS TM-Score	0.8	0.2	0.804	0.587	0.678	0.585	28823.217	0.577
MaxSub TM-Score	0.2	0.8	0.686	0.359	0.472	0.358	14485.616	0.525
RMSD GDT-HA	0.6	0.4	0.75	0.454	0.566	0.453	4378.281	0.434
RMSD GDT-TS	0.75	0.25	0.743	0.452	0.562	0.45	10682.868	0.431
RMSD MaxSub	0.6	0.4	0.686	0.359	0.472	0.358	14543.694	0.528
RMSD TM-Score	0.25	0.75	0.804	0.587	0.678	0.585	28830.971	0.577
RMSD	-	-	0.628	0.31	0.415	0.309	1137.713	0.202

Figure 10.6: *k*-medoids clustering results for each combination of measures

10.4 Clustering multiple similarity matrices

The final experiment was made using the several similarity distance matrices. In order to determine what percentage to take from each of them, a genetic algorithm was used, which saved us a lot of time since it would not be necessary to cluster the matrices with every single combination of weights.

Chapter 11

Conclusion

In this chapter, the results from the experiences are summarized. Furthermore, we also mention the limitations encountered and discuss future prospects regarding this work.

11.1 Final remarks

11.2 Detected limitations

As was mentioned before, the biggest limitation that was present throughout the experiments was the size 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, as Maxcluster would run out of memory and not output any results.

One other limitation was the fact that this work was based on the SCOP hierarchy, which is obtained with a mixture of manual and automatic analysis. Since this work aims to explore the automatic methods and their possible improvements, manual inspection of the structures was not a factor in the experiments. Given this, we could only replicate the SCOP classifications correctly up to a certain degree, as the remaining wrong classifications were most likely affected by the lack of manual inspection and the use of different algorithms to align the structures.

11.3 Future work

In future prospects, it would be interesting to see the protein databases mentioned in this document would be willing to use their resources to recompute their alignments in order to see if the current labels and cluster representatives would either change or remain the same when represented by complementing similarity measures. This of course implies that there would be no restrictions in data or sample size and therefore should show the most accurate results using the methodology that was studied during the course of this work.

Furthermore, classifications for new entries in the databases could also be experimentally obtained in order to verify if this clustering methodology for unknown label prediction may be applied in practice.

In conclusion, the improvements were not as big as expected but hopefully the community could make use of this methodology in order to extract more meaningful information from protein data.

11.4 Source code

The source code that was written to accomplish the different tasks this work required is available for consultation at `https://github.com/PArguelles/protein-clustering`

Bibliography

[1] 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.

[2] 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.

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

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

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

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

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

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

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

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

[15] 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.

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

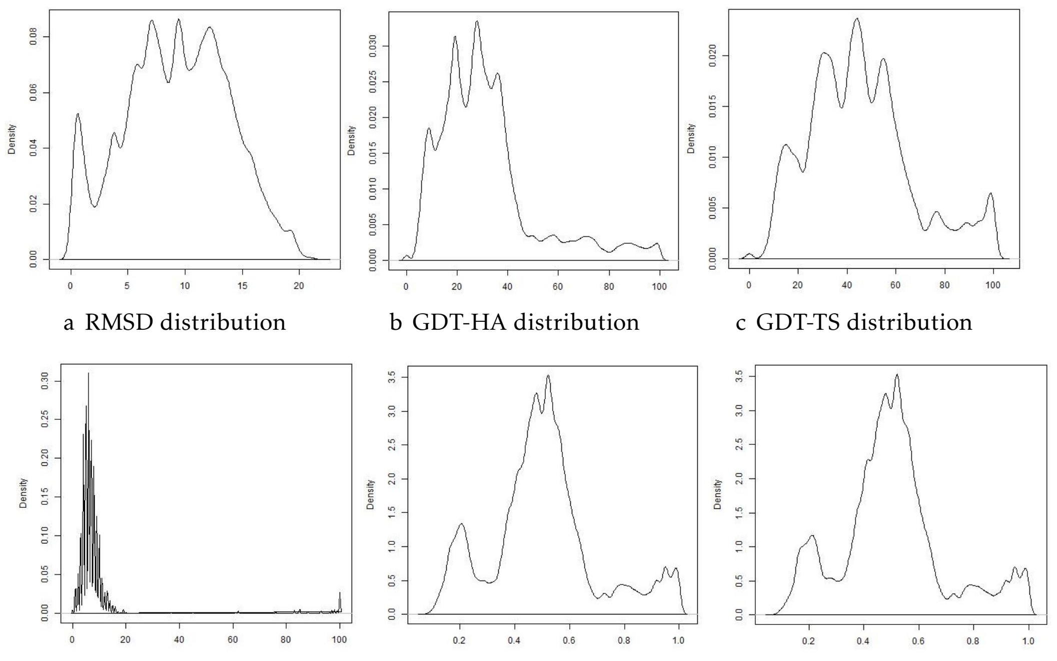
[17] C. P. Ponting and R. R. Russell. “The natural history of protein domains.” In: *Annual review of biophysics and biomolecular structure* 31.1 (2002), pp. 45–71.

ANNEX I

DATA

I.1 Results for the *a*.3 sample

I.1.1 Data distribution



a RMSD distribution

b GDT-HA distribution

c GDT-TS distribution

d Sequence distribution

e TM-Score distribution

f Maxsub distribution

Figure I.1: Kernel density estimation for the alignments of the *a*.3 sample

I.1.2 Single matrix clustering results

	Homogeneity	Completeness	V-measure	AMI	Calinski-Harabasz	Silhouette
GDT-HA	0.64	0.511	0.568	0.497	360.889	0.339
GDT-TS	0.573	0.494	0.531	0.478	374.098	0.297
MaxSub	0.565	0.498	0.529	0.481	559.408	0.398
RMSD	0.434	0.458	0.445	0.413	223.302	0.341
Sequence	0.311	0.305	0.308	0.279	930.057	0.489
TM-Score	0.548	0.491	0.517	0.473	546.655	0.399

Figure I.2: Single matrix hierarchical clustering with average linkage for sample *a*.3

	Homogeneity	Completeness	V-measure	AMI	Calinski-Harabasz	Silhouette
GDT-HA	0.571	0.419	0.483	0.402	641.764	0.275
GDT-TS	0.586	0.458	0.514	0.442	404.446	0.262
MaxSub	0.586	0.48	0.528	0.464	576.91	0.355
RMSD	0.522	0.432	0.473	0.415	532.519	0.419
Sequence	0.327	0.303	0.314	0.277	942.491	0.478
TM-Score	0.614	0.512	0.558	0.496	561.367	0.362

Figure I.3: Single matrix hierarchical clustering with complete linkage for sample *a*.3

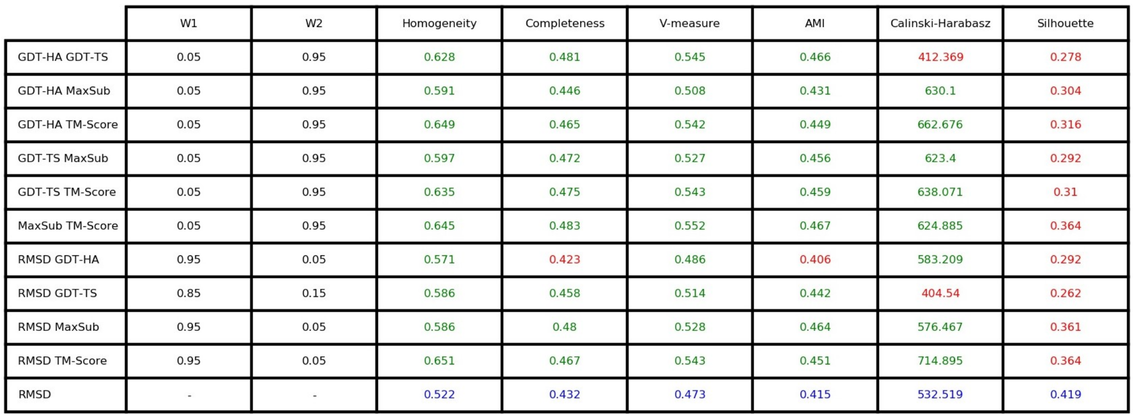
	Homogeneity	Completeness	V-measure	AMI	Calinski-Harabasz	Silhouette
GDT-HA	0.569	0.411	0.477	0.394	619.734	0.229
GDT-TS	0.503	0.377	0.431	0.358	463.653	0.215
MaxSub	0.494	0.368	0.422	0.349	481.43	0.158
RMSD	0.471	0.338	0.394	0.319	586.474	0.249
Sequence	0.191	0.222	0.205	0.161	200.983	0.225
TM-Score	0.56	0.41	0.473	0.392	558.246	0.182

Figure I.4: Single *k*-medoids clustering for sample *a*.3

I.1.3 Combined matrix clustering results

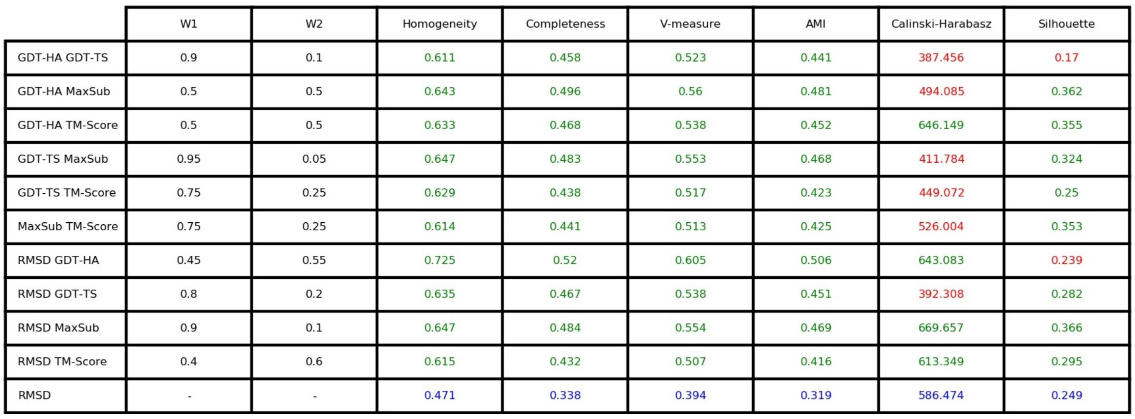
	W1	W2	Homogeneity	Completeness	V-measure	AMI	Calinski-Harabasz	Silhouette
GDT-HA GDT-TS	0.05	0.95	0.593	0.486	0.534	0.47	363.646	0.306
GDT-HA MaxSub	0.05	0.95	0.636	0.528	0.577	0.514	463.202	0.356
GDT-HA TM-Score	0.05	0.95	0.63	0.505	0.561	0.489	513.622	0.373
GDT-TS MaxSub	0.05	0.95	0.569	0.492	0.528	0.475	494.777	0.344
GDT-TS TM-Score	0.05	0.95	0.569	0.492	0.528	0.475	504.655	0.347
MaxSub TM-Score	0.95	0.05	0.548	0.491	0.517	0.473	546.352	0.399
RMSD GDT-HA	0.85	0.15	0.64	0.511	0.568	0.497	361.084	0.339
RMSD GDT-TS	0.75	0.25	0.573	0.494	0.531	0.478	374.113	0.297
RMSD MaxSub	0.95	0.05	0.566	0.493	0.527	0.477	569.649	0.41
RMSD TM-Score	0.05	0.95	0.565	0.497	0.528	0.48	515.267	0.443
RMSD	-	-	0.434	0.458	0.445	0.413	223.302	0.341

Figure I.5: Combined matrix hierarchical clustering with average linkage for sample *a*.3



	W1	W2	Homogeneity	Completeness	V-measure	AMI	Calinski-Harabasz	Silhouette
GDT-HA GDT-TS	0.05	0.95	0.628	0.481	0.545	0.466	412.369	0.278
GDT-HA MaxSub	0.05	0.95	0.591	0.446	0.508	0.431	630.1	0.304
GDT-HA TM-Score	0.05	0.95	0.649	0.465	0.542	0.449	662.676	0.316
GDT-TS MaxSub	0.05	0.95	0.597	0.472	0.527	0.456	623.4	0.292
GDT-TS TM-Score	0.05	0.95	0.635	0.475	0.543	0.459	638.071	0.31
MaxSub TM-Score	0.05	0.95	0.645	0.483	0.552	0.467	624.885	0.364
RMSD GDT-HA	0.95	0.05	0.571	0.423	0.486	0.406	583.209	0.292
RMSD GDT-TS	0.85	0.15	0.586	0.458	0.514	0.442	404.54	0.262
RMSD MaxSub	0.95	0.05	0.586	0.48	0.528	0.464	576.467	0.361
RMSD TM-Score	0.95	0.05	0.651	0.467	0.543	0.451	714.895	0.364
RMSD	-	-	0.522	0.432	0.473	0.415	532.519	0.419

Figure I.6: Combined matrix hierarchical clustering with complete linkage for sample *a*.3



	W1	W2	Homogeneity	Completeness	V-measure	AMI	Calinski-Harabasz	Silhouette
GDT-HA GDT-TS	0.9	0.1	0.611	0.458	0.523	0.441	387.456	0.17
GDT-HA MaxSub	0.5	0.5	0.643	0.496	0.56	0.481	494.085	0.362
GDT-HA TM-Score	0.5	0.5	0.633	0.468	0.538	0.452	646.149	0.355
GDT-TS MaxSub	0.95	0.05	0.647	0.483	0.553	0.468	411.784	0.324
GDT-TS TM-Score	0.75	0.25	0.629	0.438	0.517	0.423	449.072	0.25
MaxSub TM-Score	0.75	0.25	0.614	0.441	0.513	0.425	526.004	0.353
RMSD GDT-HA	0.45	0.55	0.725	0.52	0.605	0.506	643.083	0.239
RMSD GDT-TS	0.8	0.2	0.635	0.467	0.538	0.451	392.308	0.282
RMSD MaxSub	0.9	0.1	0.647	0.484	0.554	0.469	669.657	0.366
RMSD TM-Score	0.4	0.6	0.615	0.432	0.507	0.416	613.349	0.295
RMSD	-	-	0.471	0.338	0.394	0.319	586.474	0.249

Figure I.7: Combined *k*-medoids clustering for sample *a*.3

I.2 Results for the *b*.2 sample

I.2.1 Data distribution

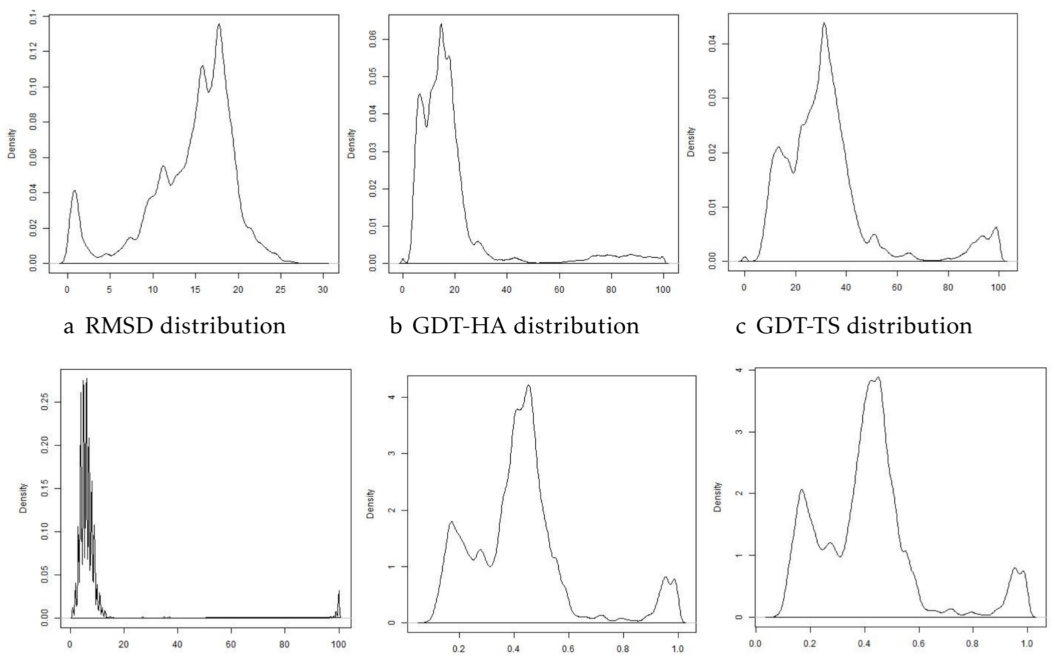


Figure I.8: Kernel density estimation for the alignments of the *b*.2 sample

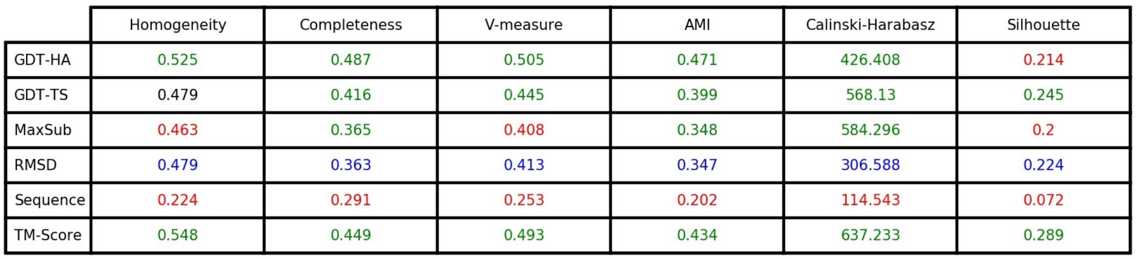
I.2.2 Single matrix clustering results

	Homogeneity	Completeness	V-measure	AMI	Calinski-Harabasz	Silhouette
GDT-HA	0.568	0.495	0.529	0.48	1409.267	0.453
GDT-TS	0.584	0.498	0.538	0.485	964.839	0.371
MaxSub	0.55	0.451	0.495	0.436	691.098	0.373
RMSD	0.565	0.459	0.507	0.445	554.976	0.382
Sequence	0.303	0.348	0.324	0.28	664.28	0.533
TM-Score	0.561	0.456	0.503	0.441	668.019	0.382

Figure I.9: Single matrix hierarchical clustering with average linkage for sample *b*.2

	Homogeneity	Completeness	V-measure	AMI	Calinski-Harabasz	Silhouette
GDT-HA	0.574	0.461	0.511	0.446	1410.208	0.427
GDT-TS	0.543	0.474	0.506	0.459	796.31	0.365
MaxSub	0.636	0.496	0.557	0.483	799.958	0.349
RMSD	0.492	0.395	0.438	0.379	403.583	0.339
Sequence	0.313	0.352	0.331	0.29	601.574	0.53
TM-Score	0.555	0.433	0.487	0.418	752.562	0.36

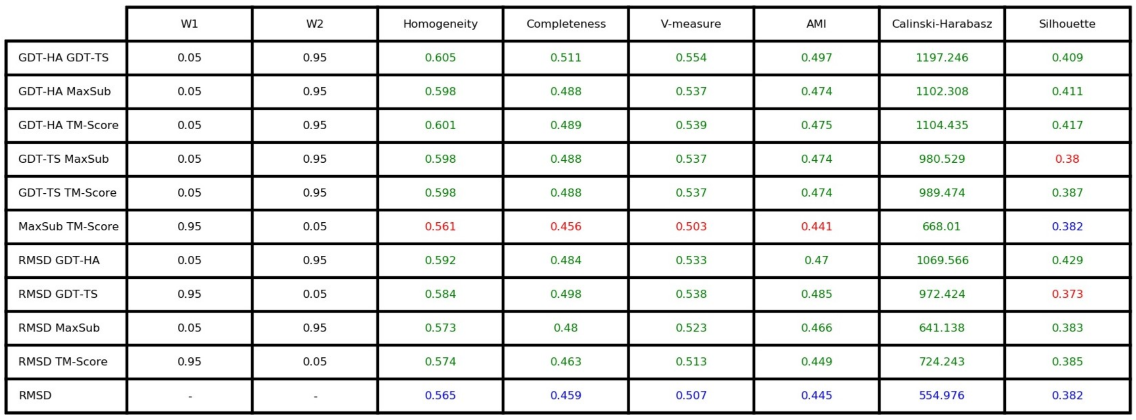
Figure I.10: Single matrix hierarchical clustering with complete linkage for sample *b*.2



	Homogeneity	Completeness	V-measure	AMI	Calinski-Harabasz	Silhouette
GDT-HA	0.525	0.487	0.505	0.471	426.408	0.214
GDT-TS	0.479	0.416	0.445	0.399	568.13	0.245
MaxSub	0.463	0.365	0.408	0.348	584.296	0.2
RMSD	0.479	0.363	0.413	0.347	306.588	0.224
Sequence	0.224	0.291	0.253	0.202	114.543	0.072
TM-Score	0.548	0.449	0.493	0.434	637.233	0.289

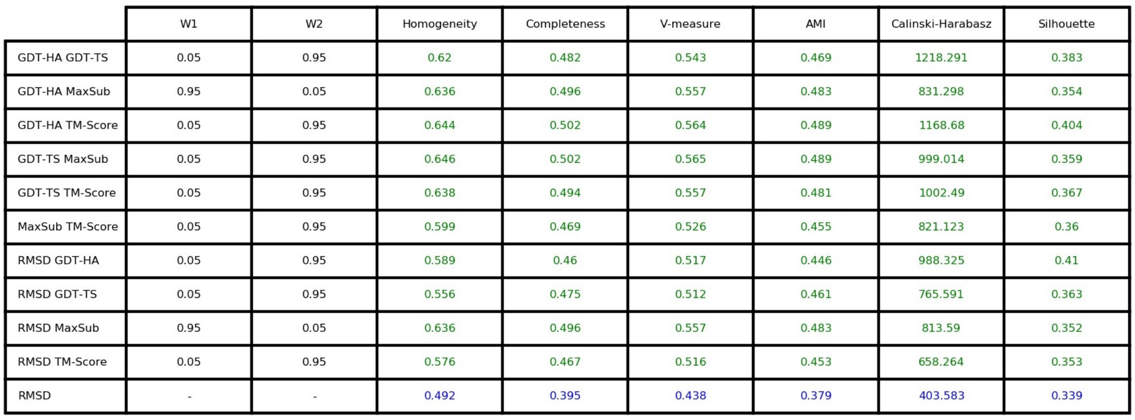
Figure I.11: Single *k*-medoids clustering for sample *b*.2

I.2.3 Combined matrix clustering results



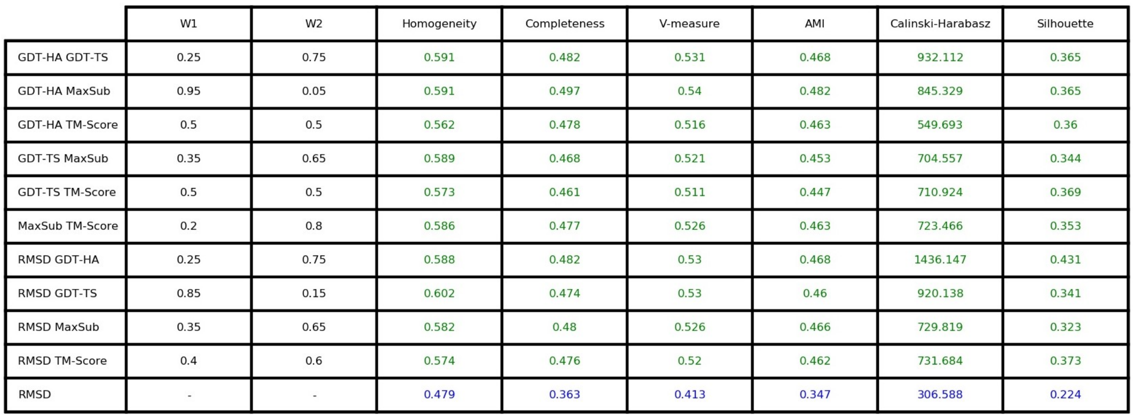
	W1	W2	Homogeneity	Completeness	V-measure	AMI	Calinski-Harabasz	Silhouette
GDT-HA GDT-TS	0.05	0.95	0.605	0.511	0.554	0.497	1197.246	0.409
GDT-HA MaxSub	0.05	0.95	0.598	0.488	0.537	0.474	1102.308	0.411
GDT-HA TM-Score	0.05	0.95	0.601	0.489	0.539	0.475	1104.435	0.417
GDT-TS MaxSub	0.05	0.95	0.598	0.488	0.537	0.474	980.529	0.38
GDT-TS TM-Score	0.05	0.95	0.598	0.488	0.537	0.474	989.474	0.387
MaxSub TM-Score	0.95	0.05	0.561	0.456	0.503	0.441	668.01	0.382
RMSD GDT-HA	0.05	0.95	0.592	0.484	0.533	0.47	1069.566	0.429
RMSD GDT-TS	0.95	0.05	0.584	0.498	0.538	0.485	972.424	0.373
RMSD MaxSub	0.05	0.95	0.573	0.48	0.523	0.466	641.138	0.383
RMSD TM-Score	0.95	0.05	0.574	0.463	0.513	0.449	724.243	0.385
RMSD	-	-	0.565	0.459	0.507	0.445	554.976	0.382

Figure I.12: Combined matrix hierarchical clustering with average linkage for sample *b*.2



	W1	W2	Homogeneity	Completeness	V-measure	AMI	Calinski-Harabasz	Silhouette
GDT-HA GDT-TS	0.05	0.95	0.62	0.482	0.543	0.469	1218.291	0.383
GDT-HA MaxSub	0.95	0.05	0.636	0.496	0.557	0.483	831.298	0.354
GDT-HA TM-Score	0.05	0.95	0.644	0.502	0.564	0.489	1168.68	0.404
GDT-TS MaxSub	0.05	0.95	0.646	0.502	0.565	0.489	999.014	0.359
GDT-TS TM-Score	0.05	0.95	0.638	0.494	0.557	0.481	1002.49	0.367
MaxSub TM-Score	0.05	0.95	0.599	0.469	0.526	0.455	821.123	0.36
RMSD GDT-HA	0.05	0.95	0.589	0.46	0.517	0.446	988.325	0.41
RMSD GDT-TS	0.05	0.95	0.556	0.475	0.512	0.461	765.591	0.363
RMSD MaxSub	0.95	0.05	0.636	0.496	0.557	0.483	813.59	0.352
RMSD TM-Score	0.05	0.95	0.576	0.467	0.516	0.453	658.264	0.353
RMSD	-	-	0.492	0.395	0.438	0.379	403.583	0.339

Figure I.13: Combined matrix hierarchical clustering with complete linkage for sample *b*.2



	W1	W2	Homogeneity	Completeness	V-measure	AMI	Calinski-Harabasz	Silhouette
GDT-HA GDT-TS	0.25	0.75	0.591	0.482	0.531	0.468	932.112	0.365
GDT-HA MaxSub	0.95	0.05	0.591	0.497	0.54	0.482	845.329	0.365
GDT-HA TM-Score	0.5	0.5	0.562	0.478	0.516	0.463	549.693	0.36
GDT-TS MaxSub	0.35	0.65	0.589	0.468	0.521	0.453	704.557	0.344
GDT-TS TM-Score	0.5	0.5	0.573	0.461	0.511	0.447	710.924	0.369
MaxSub TM-Score	0.2	0.8	0.586	0.477	0.526	0.463	723.466	0.353
RMSD GDT-HA	0.25	0.75	0.588	0.482	0.53	0.468	1436.147	0.431
RMSD GDT-TS	0.85	0.15	0.602	0.474	0.53	0.46	920.138	0.341
RMSD MaxSub	0.35	0.65	0.582	0.48	0.526	0.466	729.819	0.323
RMSD TM-Score	0.4	0.6	0.574	0.476	0.52	0.462	731.684	0.373
RMSD	-	-	0.479	0.363	0.413	0.347	306.588	0.224

Figure I.14: Combined k-medoids clustering for sample $b.2$