



**Universidad
Zaragoza**

Trabajo Fin de Máster

Prototype of bioinformatic application for rational
stabilization of proteins

Prototipo de aplicación bioinformática para
estabilizar proteínas de manera racional

Autor

Héctor García Cebollada

Director

Javier Sancho Sanz

Facultad de Ciencias

Año 2017

List of programs

Program	Version	Function
Python	3.6.1	Main programming language interpreter
Biopython	1.7	Extension of Python with functions for biological data processing
scikit-learn	0.19.0	Python module for machine learning
scipy	0.19.1	Python module for scientific uses. Scikit-learn depends on it
matplotlib	2.0.0	Python module for graph representation.
ggplot	0.11.5	Python module for ROC curves representation
numpy	1.12.1	Python module for mathematic operations. Ggplot depends on it
pandas	0.20.3	Python module for mathematic operations. Ggplot depends on it
dateutil	2.6.0	Python module for registering date and time in programs
blastp	web	Sequence-based search of similar proteins
MUSCLE	3.8.31	Multiple sequence alignment
CD-HIT	4.6.8	Sequence clustering by similarity
PhyML	3.1	Fast calculation of phylogenetic trees by Maximum Likelihood
PAML	4.9e	Calculation of phylogenetic trees and ancestral sequences
DSSP	2.2.1	Calculation of secondary structure from a PDB file
HBPlus	3.06	Calculation of hydrogen bonds from a PDB file
Disulphide by Design	1.20	Prediction of mutations that stabilize a protein using disulphide bridges
SCWRL	4.0	Minimization of energy in protein structures using rotamers
BetaVoid	1.1	Calculation of internal cavities and their volumes

List of abbreviations

Along the text of this Master thesis, several abbreviations are found. All of them are collected in the following table for an easier reading.

Abbreviation	Full name
ASR	Ancestor Sequence Reconstruction
AUC	Area Under the Curve
DbD	Disulphide by Design
ELISA	Enzyme-Linked Immuno Sorbent Assay
ESST	Environment-Specific Substitution Table
JTT	Jones, Taylor and Thornton (Authors of the substitution model)
LUCA	Last Universal Common Ancestor
ML	Maximum Likelihood
ROC	Receiver Operating Characteristic (curve)
SD	Standard Deviation
SVM	Support Vector Machine
WT	Wild Type
wwPDB	Worldwide Protein Data Bank
$\Delta\Delta G$	Difference in Gibbs free energy

Also, aminoacids and residues are referred to along the text with their full name, one-letter code and three-letter code. The following table gathers all codes for all 20 proteinogenic aminoacids and 2 ambiguous positions (Asx and Glx):

Full name	Three-letter code	One-letter code
Alanine	Ala	A
Arginine	Arg	R
Asparagine	Asn	N
Aspartic acid	Asp	D
Asparagine or aspartic acid	Asx	B
Cysteine	Cys	C
Glutamic acid	Glu	E
Glutamine	Gln	Q
Glutamine or glutamic acid	Glx	Z
Glycine	Gly	G
Histidine	His	H
Isoleucine	Ile	I
Leucine	Leu	L
Lysine	Lys	K
Methionine	Met	M
Phenylalanine	Phe	F
Proline	Pro	P
Serine	Ser	S
Threonine	Thr	T
Tryptophan	Try	W
Tyrosine	Tyr	Y
Valine	Val	V

Glossary

Residue: Specific monomer forming part of a macromolecule. In the case of proteins, residues are aminoacids after reacting to form the peptide bond.

Homologous proteins: Homologous proteins are proteins that share a common part of their ancestry. This is usually studied using sequence alignments and comparison. Two sequences are **orthologous** if their descend from the same ancestral sequence, separated in two species after and speciation event, and **paralogous** if their origin resides in an event of duplication of the gene.

Protein family: Group of evolutionarily-related proteins whose ancestry is usually inferred with sequence alignment methods and have similar three-dimensional structures and functions. The biggest grouping of proteins with a common ancestor is a **protein superfamily**.

Extant sequence: Sequence that currently exists today in a living organism.

Consensus sequence: Sequence formed by the most common residue found at each position in a multiple sequence alignment, usually calculated for protein families or for proteins sharing a function/location in the cell, so that sequence motifs can be found.

Ancestral sequence: Sequence calculated for a common ancestor for a group of sequences, usually extant, using computational algorithms known as Ancestral Sequence Reconstruction (ASR).

Likelihood: Probability of a certain model of being true. In the case of phylogenetic trees, it is the probability of obtaining the extant sequences with that tree disposition using the specified evolutionary model.

Evolutionary time: Measure of time in phylogenetic trees. As trees are based on models, this models can calculate how much time ago an speciation process happened in evolutionary time units. However, as different branches of a tree can have different evolution rates and models can be more or less accurate, evolutionary time is not always the same as the usual concept of time.

Terminal: In Fedora (and other operative systems), a terminal is a way of giving commands to the computer using text instead of interacting with a graphical environment. It is useful for automating processes.

Index: In programming, the index is the position of an element in a bigger element, such as the position of an item in a list of items or of a character in a text string. In most programming languages (Python included), the index of the first element is 0.

Function: In programming, a function (in a general sense) is a section of a program that performs a specific task. These functions may require particular **arguments**, information necessary for performing that task. When a function is **called**, the specific task is performed with the given arguments. As a result, the function can **return** data to the program for further use, **print** some information on the screen or **execute** the designed operations, returning a None value (equivalent to returning no information at all).

Class: In programming, method for creating an object using defining **arguments**. It can also have **class-specific functions**. For example, sequence could be a class. To define a sequence, three arguments would be necessary: type of sequence (DNA, RNA or protein), Alphabet (One letter code or three letter code) and the sequence itself. With a sequence object, some class-specific functions can be used, such as performing a BLAST or a sequence alignment.

Module: Section of a program that can work independently from most of the rest of the program. It usually performs an important function for the program, from which only a result is needed. This function must be **imported** to the main program before its usage.

Segmentation fault: Error that occurs when a program attempts to access a memory location without the necessary permissions.

Training: In machine learning, iterative process to adjust the parameters of a model in order to predict something based on the characteristics of an input object. For training, a list of input objects with their respective result for the prediction (measured before training) is needed.

Computational time: Time elapsed by a computer performing an operation or executing a program. As computers can perform several processes at the same time, it is not the same as real time. For example, if a computer is performing 2 processes, computational time would be twice the real time.

Parallel computing: Type of computation in which many calculations or several processes are performed simultaneously, dividing a large problem into smaller ones.

Contents

List of programs	i
List of abbreviations	ii
Glossary	iii
Abstract.....	1
Background.....	2
Objectives	4
Theory behind the project.....	5
Proteins as three-dimensional structures	5
Hydrogen bonds.....	5
Disulphide bonds	6
Alpha-helices and secondary structure	7
Polarity and exposure of side chains	9
Steric clashes	10
Internal cavities.....	11
Proteins as sequences	12
Consensus sequence	12
Ancestral sequence	14
Methods	17
Global workflow of the proposing prototype	17
Modules	18
The PointMutation class	19
Retrieval	20
Sequence analysis	20
Alpha-helices	23
Disulphide bonds	26
Exposure	27
Acidic hydrogen-bonded residues	29
Cavities and steric clashes	31
Scoring model.....	32
Training and testing the programme.....	33
Database.....	33
Evaluation statistics and troubleshooting	35
Prediction training and testing.....	35
Proposal testing	36
Results	38
Alpha-helix scoring comparison.....	38
Running the program. Troubleshooting	38
Mutation database evaluation statistics	41

Prediction accuracy.....	41
Proposal statistics	43
Program input and output	44
Discussion.....	46
Alpha-helix scoring comparison.....	46
Program input and output	46
Running the program. Troubleshooting.	47
Mutation database evaluation statistics	49
Prediction accuracy.....	50
Proposal statistics	52
Outlook.....	54
Conclusions	55
Conclusiones.....	55
Bibliography	56
Appendix A. Code of the program	I
PointMutation class (mutation.py)	I
Retrieval (clean_retrieval.py)	II
Sequence analysis (clean_consensus.py).....	III
Alpha-helices (clean_alpha_helix.py)	XIII
Disulphide bonds (clean_disulphide.py)	XVIII
Exposure (clean_exposure.py)	XXI
Acidic hydrogen-bonded residues (clean_acid_hbonds.py).....	XXII
Cavities and steric clashes (clean_cavity.py)	XXIII
Scoring model (clean_scoring_model.py).....	XXV
Main program (prototype.py)	XXVII
Appendix B. Dataset for alpha-helix mutation evaluation	XXXVIII
Internal residues.....	XXXVIII
N-cap	XXXVIII
C-cap.....	XXXVIII
Average exposure for each residue.....	XXXIX
Appendix C. Parameter file for PAML	XL

Abstract

Increasing the stability of proteins is important for several applications, such as the production and storage of diagnostic kits and antibodies for therapeutic use or the industrial enzymatic catalysis in processes at high temperature. For this reason, several approaches have been used to predict the effects on stability of a mutation. However, all current programs can only predict the effect of a mutation given by the user. In this Master thesis, a method based on simple rational and empirical rules that both proposes mutations and predicts qualitatively their effect in stability is developed. Rules used for this program focus both on structural features (composition of alpha-helices, disulphide bonds, exposed acidic hydrogen-bonded residues, overexposed apolar residues, buried polar residues, steric clashes and internal cavities) and on the sequence of the protein (calculation of a consensus and an ancestral sequences). Mutations are then evaluated with a logistic regression model trained using machine learning techniques with a training group obtained from ProTherm database. The accuracy of the model is tested with another group of mutations obtained in ProTherm, reaching a higher accuracy than current methods. Finally, a user-friendly input and output format is developed for a general use of the program in research.

Aumentar la estabilidad proteica es importante para diversas aplicaciones, como la producción y almacenamiento de kits diagnósticos y anticuerpos con uso terapéutico o la catálisis enzimática en procesos industriales a alta temperatura. Por ello se han usado diversas aproximaciones para predecir los efectos en la estabilidad de una mutación. Sin embargo, los programas actuales solo predicen el efecto de una mutación introducida por el usuario. En este trabajo fin de Máster, se ha desarrollado un método basado en reglas empíricas y racionales simples que propone mutaciones y predice cualitativamente su efecto en la estabilidad. Las reglas utilizadas en el programa comprenden tanto propiedades estructurales (composición de hélices alfa, puentes disulfuro, residuos ácidos expuestos con puentes de hidrógeno, residuos apolares hiperexpuestos y polares enterrados, choques estéricos y cavidades internas) como de la secuencia (cálculo de las secuencias consenso y ancestral). Posteriormente, las mutaciones son evaluadas con un modelo de regresión logística entrenado mediante machine learning con un grupo de mutaciones obtenido de la base de datos ProTherm. La precisión del programa se calcula con otro grupo de mutaciones de ProTherm, alcanzando valores superiores a los de los métodos actuales. Finalmente, se han desarrollado formatos de entrada y salida de información sencillos para que cualquier usuario básico del campo de la investigación lo pueda usar.

Background

Proteins are macromolecules formed by amino acids. There are 20 different proteinogenic amino acids, all of which are L- α -amino acids, except for glycine, which is not chiral. These amino acids are sequentially linked by peptide bonds forming chains that have two ends, N- and C-terminus, with their amino and acid moieties not bonded to other amino acid, respectively [1]. With the exception of intrinsically disordered proteins or regions, each of those chains usually folds at physiological conditions in a precise conformation, which allows it to fulfil its function [1]. In a living organism, these functions range from structural ones, such as forming the cytoskeleton, to catalytic functions, as those performed by enzymes. [1]

However, technological developments allow for the usage of some proteins in environments different from the cell and with other purposes. Examples of this are the usage of antibodies or enzymes for the detection of small amounts of a substance (e.g., ELISA, flow cytometry with labelled antibodies) [2, 3], the production of kits for the amplification of nucleic acids [4] or the industrial usage of enzymes or cross-linked enzyme aggregates for the catalysis of beneficial chemical reactions [5]. Another important field are therapeutics, where enzymes and antibodies can be used as treatments for certain conditions [6].

One of the main hindrances for a wider development of these techniques is protein stability. Depending on the protein and on solution conditions (e.g., pH, temperature, salt and denaturant concentration), a larger or smaller fraction of protein molecules will fold correctly. The more different conditions are from physiological ones, the smaller the folded protein fraction will be [7]. It is particularly important that a higher stability results in a higher percentage of folded protein, which usually means a higher level of the desired activity or properties [1] and, in protein-based diagnostic kits or therapeutics, a longer shelf life [8]. Also, this allows for a higher operation temperature without a significant loss of activity in industrial reactors, so that a higher transformation rate is achieved.[9]

Because of all these reasons, efforts are being made to understand protein folding and how to increase protein stability by replacing specific amino acid residues in the sequence with others. Initially, some empirical observations were made and some improvements were proposed, such as engineering disulphide bonds or optimizing alpha helices. However, these simple rules do not always work, so more complex methods are

being developed. Currently, three main different approaches are being used for predicting the effects of mutations on protein stability: force fields [10], machine learning [11] and sequence-based analyses [12-14].

Force field based methods predict the difference between the stability of the native and mutant protein ($\Delta\Delta G$) using atom and torsion angle potentials, calculated statistically from a set of non-redundant protein structures, as well as electrostatic and van der Waals forces.[10]

Machine learning based methods rely on structure and/or on sequence analysis to predict if the mutation is stabilizing, destabilizing or neutral. In all cases, an algorithm known as support vector machine analyzes a training dataset to develop some rules and classify the mutations from the data provided to the program. Sequence-based machine learning methods use the nearest-sequence neighbours as input data, while structure-based machine learning methods use the residues found in a sphere surrounding the alpha carbon of the mutated amino acid as input data.[11]

Sequence-based methods, such as SIFT [12] or PROVEAN [13] use phylogenetically close homologous proteins' sequences to evaluate the probability of the mutation being deleterious by alignment analysis. Some of these methods, such as Polyphen 2 [14], also evaluate some simple structural characteristics, such as avoiding destabilizing steric effects or the presence of charged residues in the hydrophobic core of the protein.

All these methods have proven a high accuracy in validation tests, even reaching 80%.[10-14] However, they can only predict the effect of specified mutations, and do not aim at proposing the best options to improve stability for a given protein, because the analyses they perform are based on statistics and not in rational rules, so that the only way for them to identify the most stabilizing mutations would be to analyze all possible mutations, which would be nineteen times the sequence length, extraordinarily increasing computational cost and complicating output analysis.

A combination of the modern machine learning methods with rational and empirical rules proposed earlier could solve this problem, making a program able to both propose mutations (with rational rules) and ordering them according to their probability of being stabilizing, evaluated using information from both the sequence and the structure of the protein.

Objectives

Based on the hypothesis that the effect of mutations on protein conformational stability can be estimated from simple rules and data derived from the structure and sequence of the analyzed protein, the objectives of this Master thesis are:

- Developing a program that analyzes quickly several structural and sequential properties of a protein and proposes probably stabilizing mutations based on rational rules.
- Developing a machine learning based program to evaluate the probability of a mutation of being stabilizing using the same structural and sequential properties.
- Training the evaluating program with an experimental protein stability database.
- Evaluating the accuracy of the trained program with a dataset different from the training one.
- Reporting the obtained results for a given protein in an accurate, precise and easy to interpret format.

Theory behind the project

As proteins are macromolecules, a complex network of interactions between their thousands of atoms can explain their folding. However, such networks are so complex that the calculations required for a correct study of stability based on them take very long times of computation [15] and are unlikely to produce accurate enough results. Apart from carefully considering their three dimensional properties it is important to realise that proteins are products of evolution and that analysis of related protein sequences may provide valuable insight [16], so both structure and sequence approaches are worth considering.

Proteins as three-dimensional structures

Even though studying all interactions occurring in a protein is not feasible for a fast program, there are some desirable or undesirable properties to be found in a stabilizing mutation, which allow for a fast (but less accurate) analysis of the entire protein. Some of these properties concern the composition of secondary structure elements, such as alpha-helices, others are related to the physical-chemical properties of the protein and of the original and mutated residues, and some others are related to specific bonds that can be found in proteins, such as disulphide bonds or hydrogen bonds.

From all the possible features that can be studied in a protein structure, some have been selected to build this program due to their better understanding in current literature. The following sections will describe basic data of each of those selected properties for a better comprehension of the whole thesis.

Hydrogen bonds

A hydrogen bond is a type of electrostatic interaction between two polar groups: the donor, which is a hydrogen atom bonded to a highly electronegative small atom (i.e. oxygen, nitrogen or fluorine) and the acceptor, which is another electronegative atom located near the donor [17]. This kind of interaction is stronger than van der Waals interactions, but it is still a weak interaction, much weaker than covalent bonds. Its formation free energy depends on the atoms which are bonded, their environment and the geometry of the bond, so it covers a range of more than two factors of ten, from 0.2 to 40 kcal/mol.[18]

Hydrogen bonds are important in proteins as they contribute to the stabilization of certain elements of the secondary structure, such as alpha helices and beta sheets, and

they are involved in shaping the global folding of the protein, by stabilizing the tertiary structure too. [1]

Therefore, modifying the pattern of hydrogen bonds of a protein with mutations might have some impact on both the stability and structure of the mutant protein. As the objective of the program is making proteins more stable without changing its function, which is related to the structure, and there is no clear evidence that proves stabilization when introducing hydrogen bonds, this option is not considered.

However, if the strength of existing hydrogen bonds is increased, a stabilizing effect may be achieved without risking changing the structure. This approach has been successfully performed before on apoflavodoxin [19]. Also, checking for side chain hydrogen bonds not being eliminated when a given mutation is implemented will help proposing better mutations.

Disulphide bonds

Disulphide bonds are covalent bonds formed between two sulphur atoms, usually from the oxidation of thiol groups. In proteins, thiol groups are only found in cysteines. Disulphide bonds are responsible for the folding of some proteins, predominantly secreted ones, as disulphide bonds are highly dependent on the redox environment and most cellular compartments are too reducing for cysteines to become cystines (two cysteines bonded through a disulphide). However, the endoplasmic reticulum offers an oxidizing environment, appropriate for the folding of such proteins [8]. Some proteins, as insulin, rely on these links to keep their constituting chains together.

However, not all disulphide bonds are stabilizing, due to the fact that the formation of a disulphide can introduce strain in the folded protein, destabilizing many other favourable interactions. Also, when two parts of the backbone of a protein are joined together by a disulphide bond, their mobility is reduced compared to the wild type protein, so it may alter the original function of the protein if it affects relevant residues. To avoid a big impact of these effects, several properties must be taken into consideration, such as the distance between sulphurs, and between the carbons they are linked to, the torsion angle of the link and the flexibility of the region of the protein where the disulphide is introduced [8]. As this approach is usually effective, simple software (Disulphide by Design [20, 21]) has been developed to predict stabilizing mutations and their thermodynamic effect by applying these rules. The usefulness of such software has been successfully demonstrated in several research papers [22, 23].

Alpha-helices and secondary structure

Secondary structure is the regular periodic conformation adopted by local segments of proteins. This structure is determined by phi and psi angles, which describe how the amino acid is rotating relative to its peptide bonds. Phi angle is the rotation of the bond between the alpha carbon and the nitrogen of the previous peptide bond (previous meaning closer to the N-terminus) and psi angle measures the rotation of the bond between the alpha carbon and the carbon of the next peptide bond.[24]

Certain combinations of phi and psi angles in consecutive residues lead to dispositions of the backbone and side chains that favour the formation of hydrogen bonds, such as alpha-helices and beta-strands, where patterns of hydrogen bonds and structural properties are well determined. [1]

Alpha-helices (figure 1) are formed by four or more residues with phi and psi angles around -60° and 45° , respectively. These angles generate a helical structure in which the carbonyl group of residue i forms a hydrogen bond with the hydrogen of the amine group of residue $i+4$. The ends of the helix are named as N-cap and C- cap, N-cap being the first residue without alpha-helix phi and psi angles that still establishes hydrogen bonds with the amines of the helix closest in sequence to the N-terminus, and C-cap the equivalent to N-cap but in the other end of the helix and interacting with carbonyl groups. Side chains of the residues conforming the helix go out of the centre of the helix, slightly tilted towards the N-end of the helix.[1]

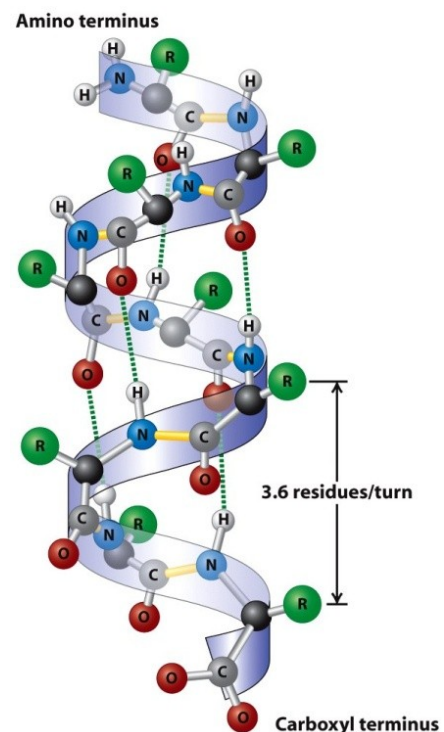


Figure 1. Structure of an alpha-helix. Peptide bonds highlighted in yellow, alpha carbon atoms in black, side chains shown as R, hydrogen bonds as dotted green lines. Taken from Molecular Cell Biology, Lodish et al. 6th edition[1]

Several studies have been performed to understand the propensity of a sequence of amino acids to form a helix, and how this affects helix and, hence, protein stability [25]. Software for this predictions has been developed, such as AGADIR [7, 26, 27]. However, studies on the abundance of each type of amino acid in alpha-helices have shown that the distribution of amino acid residues is not random, some residues being favoured over others. In initial experimental studies, all positions in the helix were

considered to be equivalent [28], but it was soon concluded that three different regions have to be distinguished, as their stability behaviour related to amino acid composition is totally different. They are the internal positions amino acids, the N-caps and the C-caps. [27]

Most of the initial studies of the helix formation propensity of amino acids were performed using small peptides [28]. However, this data didn't explain for the proportion of amino acids inside alpha-helices found in real proteins. This fact does not imply that data from small peptides was wrong, as the proportion of amino acids in alpha helices is an evolutionary result that does not necessarily correlate with stability, but it sparked interest in alpha helices in proteins and similar studies using statistics of the existing protein database and mutagenesis and thermodynamic quantifications of free energy in existing proteins were performed, obtaining more accurate estimates of the relative stabilization afforded by the presence of a given amino acid inside a helix relative to that of others [29]. This set of data was further improved with the implementation of AGADIR, where iterative algorithms were applied to find the best fitting set of values to accurately predict existing data [7, 26, 27]. However, this does not account for all the stability effects of the inner residues of a helix, as the side chains of residues i and $i+3$ or $i+4$ are close to each other (each turn of the helix takes 3.6 residues) and they can establish stabilizing interactions [29]. For inner residues, alanine is found to be the most stabilizing one.[7, 26-29]

N-caps do not behave as internal residues due to the fact that the amine groups of the closest residues inside the helix are not hydrogen-bonded to other residues. Because of this, a partial positive charge of the amine groups is found in the N-end of the helix, so that residues that can interact forming hydrogen bonds with the free amine groups or establishing electrostatic interactions with the positive charge, such as the negatively charged amino acids (aspartic and glutamic acid) or their amines (asparagine and glutamine, which form hydrogen bonds) are more stabilizing than others. Studies have been conducted with model peptides [30], proteins and their mutants [31] and further adjusted with iterative algorithms [7, 26, 27], as in the case of internal residues. The most stabilizing N-caps have been found to be asparagine and aspartic acid. As aspartic acid has acidic properties, it can be ionized and create an electrostatic charge. This charge can alter other interactions in the protein, especially electrostatic ones, so asparagine is preferred over aspartic acid as N-cap.

As N-caps, C-caps show a different behaviour from inner residues too, due to the fact that carbonyl groups at the C-end of the helix are not forming hydrogen bonds with the rest of the helix, so a partial negative charge is generated. Because of this, positively charged residues (lysine and arginine) were expected to be the most stabilizing ones for this position. Data from studies conducted in model peptides [30] backed this hypothesis, being arginine the most stabilizing C-cap, but data from proteins [31] and further adjustments [7, 26, 27] showed that the most stabilizing C-cap in proteins was glycine. Several hypothesis about the cause of this unexpected stabilization were developed. The two main hypothesis were that a small side chain allowed for a better solvation of the C-end of the helix and that glycine allowed for more phi and psi angles, making it possible to reduce the conformational stress of the structure. Recent data obtained with chemical protein synthesis and D-amino acids fit better with the conformational stress hypothesis. [32]

Polarity and exposure of side chains

Each of the twenty proteinogenic amino acids is different from the others due to its side chain. These side chains may possess similar properties, as in the case of leucine and isoleucine, or may be very different, such as glycine and tryptophan. There are different classifications of the amino acids according to these properties, such as charged and uncharged, small, aromatic, aliphatic, polar or hydrophobic. [1]

For the stability of a protein it is important to consider where the hydrophobic and hydrophilic residues are. As proteins are in aqueous solution (i.e. a polar environment), polar residues establish stabilizing interactions with water, while hydrophobic residues do not. This simple fact can determine whether a mutation will be stabilizing or not.

Comparing the structure of the folded and unfolded states of a protein, the stabilizing and destabilizing changes can be, to some extent, anticipated. As hydrophobic residues do not interact favourably with water and perturb water-water interactions, the burial of these residues at folding increases the entropy of water, stabilizing the folded form due to the hydrophobic effect. However, the burial of polar groups destabilizes the protein due to the loss of interactions between water and polar residues that take place in the unfolded state. Therefore, mutating polar buried residues into similarly sized hydrophobic ones can be a starting point for finding stabilizing mutations. This approach has been successfully applied in apoflavodoxin, allowing to develop equations to quantify this effect[33].

For the case of residues on the surface of the folded conformation, polar residues establish interactions with water, as they do in the unfolded state, but they may also interact with other polar groups forming stabilizing interactions. Hydrophobic groups on the surface are, in principle, neither stabilizing nor destabilizing, as they are creating the same reduction of water entropy as in the unfolded state. However, if a hydrophobic residue is more exposed in the folded than in the unfolded state, it will be destabilizing. Therefore, overexposed hydrophobic residues should be mutated into polar residues.[34]

To determine whether a residue is buried or overexposed, the folded state should be compared with the unfolded state. This is not a simple issue, as the unfolded state is a large ensemble of conformations. There are servers for the calculation of unfolded conformations (ProtSA[35]), but their usage would increase significantly the computation time, so the average exposures in the unfolded state that were calculated for each residue in the development of the server [35] will be used instead.

Steric clashes

Each atom of a protein occupies a certain volume. If two atoms (bonded or not) are forced to stay too close, repulsion forces due to their electron clouds become bigger than attraction, and the native conformation of the protein is destabilized.[1] This is to be taken into account when considering a mutation for a protein. As the 20 proteinogenic amino acids have different side chains, each occupies a different volume, as it can be deduced from their different molecular weights, ranging from 75 Da of glycine to 204 Da of tryptophan.[1]

Because of this, it is important to check that any residue introduced by mutation will not cause clashes in its surroundings. The most common way to analyze this is using visualization programs, such as Swiss PDBViewer [36] or VMD [37], but an automated way is needed for this work. For this purpose, some programs, such as SCWRL [38], can perform a minimization of the potential energy of the protein structure. If steric clashes appear, the repulsion forces will become bigger, increasing significantly the minimum potential energy, thus decreasing the conformational stability. Because of this, the minimum potential energy calculated by SCWRL is considered a good steric clashes detection property.

Internal cavities

There are several types of cavities in proteins that may facilitate their interaction with small molecules and play key roles in function and stability. For example, invaginations of the surface of a protein can be catalytic pockets where the substrate of a reaction fits in the optimal position for that reaction. Mutating there could well result in a change of stability, but the function of the protein will be altered. In this Master thesis, the following nomenclature will be used [39]:

Cavity is a general term for all spaces inside or close to the surface of a protein where a certain molecule fits. For this work, the considered molecule is a molecule of water.

Internal cavity (figure 2a) is a type of cavity from which a small molecule (e.g. water) cannot reach the surface of the protein.

Pocket (figure 2b) is a type of cavity located on the surface of the protein with

only one connection to the outside of the protein. An example of pocket can be the invagination mentioned at the beginning of this section. These pockets can be catalytic pockets, may have a role in molecular recognition or in regulation of the catalytic activity of the protein. [39]

Channel (figure 2c) is a type of cavity in contact with the surface of the protein with two connections to the outside of the protein. As its name indicates, it is a channel connecting two areas of the surface while going inside of the protein. Channels can be involved in molecular transport or catalysis.[39]

Some studies show that aliphatic deletions in the protein hydrophobic core, made by generating large-to-small mutations, result in less stable variants of the protein [40], so it can be concluded that internal cavities may have a negative effect on stability. The reverse approach was later used in apoflavodoxin, obtaining modest stabilizations by small-to-large mutations in internal cavities, with smaller effects than expected, due to the rearrangement required to avoid clashes with the new residue [41].

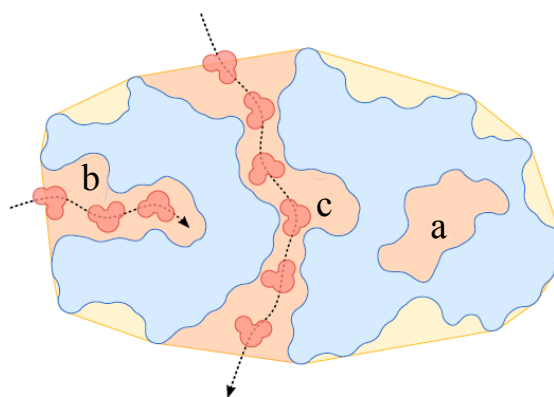


Figure 2. Different types of cavities.

Internal cavity (a), pocket (b) and channel(c).

This is a top view of a slice of a protein. In blue, the protein. In orange, cavities. In yellow, boundaries (volumes considered close to the surface of the protein. Two molecular paths (in b and c) are depicted for the studied molecule. Adapted from Krone et al.[39]

Internal cavities are good targets for stabilizing mutations, as they are not usually involved in the function of proteins and moderate stabilizations can be achieved by filling those cavities with larger aliphatic residues. However, current developments are mainly focused on the detection of external cavities and prediction of molecules that bind to them, in order to find new drugs, so few programs are focused on internal cavities and, according to a recent review [39], with current software it is only possible to detect the atoms surrounding cavities in a semi-automatic manner, using graphic visualizations for the user to decide which atoms are defining the cavity. What can be performed currently in an automated pipeline is the calculation of the volume of these internal cavities [42].

Even though programs for the detection of channels and pockets are more advanced than for internal cavities, they can be neither used in an automated manner. Even if they could, this wouldn't be a recommended approach to increase stability in proteins, as both channels and pockets are usually linked to the function of the protein, and this modification could change its function, which is usually an undesirable effect. Because of this, the study of external cavities is deemed unnecessary for this project.

Proteins as sequences

Apart from studying protein structure as the result of several forces combined, it is also important to consider protein sequence as the result of evolution, where changes in the sequence are positively selected if they give the organism an advantage relative to the original form. With the current computational methods, it is possible to study not only proteins from currently existing organisms (extant proteins) but also their phylogenetical precursor predicted sequences, from back in time, when the average temperature of Earth was higher than current [43]. For this project, both approaches will be used.

Consensus sequence

Homologous sequences are those that apparently share a common ancestor and, because of that, they have a high percentage of identity and analogy between them. If two close-related species are analyzed, most of their proteins have an equivalent homologous protein in the other species, with only some or no changes at all. The differences between two homologous sequences could be due to an improvement of the function of the protein for that specific species or to a change in function, which may later generate a new family of homologous proteins in further evolution.[44]

To find homologous sequences, a search for sequences with a high similarity or identity percentage can be performed. The problem of this approach is that it does not necessarily distinguish homologous sequences with different functions from the ones that keep the same as the original search sequence.[44] This is an inconvenient feature for this work. However, for a change of function, the required change in the sequence is usually bigger, so few sequences with a different function will be found using restrictive parameters while searching.[44]

Inside a family of homologous proteins, differences between sequences can be found. If all these variants are aligned using some of the currently available software (MUSCLE [45], ClustalW [46]), a consensus sequence for the whole family can be deduced. The consensus sequence is composed by the most common amino acid in all sequences for a given position. For some purposes, such as the program developed for this thesis, it is convenient to set a minimum threshold for the relative frequency of the amino acid. If the most common amino acid falls under that threshold, an ambiguous amino acid “X” is placed in the consensus sequence instead. This way, variants in positions with a high variation rate are not interpreted as relevant.

Consensus sequences are often used in the search of signalling patterns for specific functions or cell locations. However, they can also be interpreted as a reference for the most beneficial variants. They can be more beneficial because of a better regulation, better expression or better stability, among many other possibilities. Because of this, many prediction programs based on sequence consider the alignment of extant sequences and the evaluation of the consensus sequence as a way to evaluate the impact of a mutation in protein stability. [12-14].

Ancestral sequence

Another approach to find stabilizing mutations for a protein is using the extant sequences of a protein and its family to reconstruct the sequence of their ancestors through phylogenetics, in a process known as ancestral sequence reconstruction (ASR). Unlike consensus sequence, which compares only the sequences of the present, ASR tries to find the history of changes of the protein to form the current family of homologous proteins. Because of this, ASR is termed “vertical approach”, while consensus is referred as “horizontal approach”. There is no standard work pipeline for this purpose. However, there are some elements that tend to be common in most of the research conducted in this field. [16] A simple graphical explanation of phylogenetic terms used below is offered in figure 3.

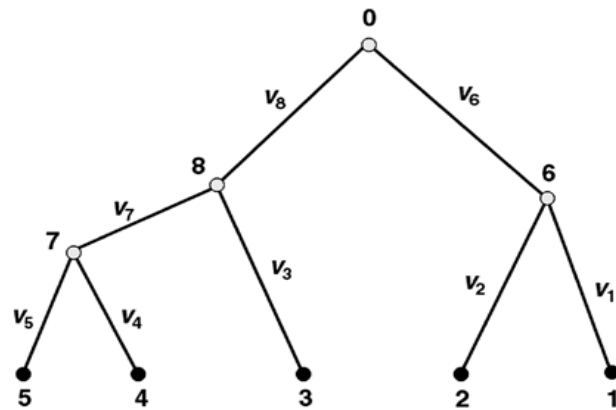


Figure 3. Example of a phylogenetic rooted tree.

Black dots (1-5) are the leaves of the tree. They represent extant proteins, proteins from currently existing organisms.

White dots (6-8) are the nodes of the tree. They represent the common ancestors of the other leaves or nodes they are connected to.

The top node (0) is the root, the common ancestor to the whole tree. Usually, an outgroup or information apart from the sequence is required for a correct placing of the root, but an incorrect rooting does not affect maximum likelihood calculations [16].

The straight lines between nodes/leaves are branches. They represent an ancestral relationship, being the one closer to the root the ancestor of the other. The values close to these lines (v_1 -8) are the length of the branches. In rooted trees with a molecular clock, this length is a measure of the evolutionary time between ancestor and descendant.

Figure taken from Merkl et al. 2016 [16]

For the ancestral sequence reconstruction, four steps, involving different software, are necessary: selection of extant sequences, creation of a multiple alignment, computation of a phylogenetic tree and reconstruction of the ancestral sequence.[16]

First step is the selection of sequences for the analysis. Extant sequences are the leaves of the phylogenetic tree. The more difference between sequences, the bigger distance between the branches of the tree. If only a few similar sequences are considered, the root of the tree will be close in time to the present or even can cause execution errors in the phylogeny program due to a high level of identity. If there are few sequences but they are very different from each other, the resulting tree may go a very long time back in evolution, but it will provide an inaccurate prediction.[16] The computational time associated to the construction of a phylogenetic tree grows linearly with the length of the sequence and with the square of the number of sequences.[47] Because of this, it is not recommended to use a big number of sequences. To control the number of

sequences and to optimize the results, a frequent solution is the usage of clustering software, which groups sequences in clusters with more than a given percentage of identity to a representative sequence, reducing the number of sequences to one per cluster and ensuring the differences between clusters are significant, but not too big. [48]

Another fact to take into account is that recent studies show that, whereas common ancestor resuscitated proteins are usually thermophilic variants with higher stability at high temperature due to the conditions of life back in the age of the ancestor, the last universal common ancestor (LUCA) resuscitated proteins are usually mesophilic variants with similar stability to the extant ones [16, 49]. This fact points out the importance of not going too far back, not only because the longer the time predicted, the less accurate the prediction, but also because longer times do not necessarily imply a higher stabilization. As proteins with a low percentage of similarity are phylogenetically related further back in time than those with a higher percentage, minimum thresholds of similarity between clusters are imposed, in order to avoid getting an inaccurate ancestral sequence from too far back in time.

Even considering minimum thresholds and performing clusterization, the number of sequences considered can vary depending on protein-specific mutation rates and the time span of interest. Because of this, there is no optimal number of sequences, and datasets used may vary from 11 to over 200 sequences [16]

The second step is the alignment of all sequences for further construction of the tree. There is a wide range of heuristic software for this purpose with similar alignment quality, such as Clustal (X and W) [46] and MUSCLE[45], being the latter one of the most frequently used.[16]

The third step is the construction of a phylogenetic tree for the family of homologous proteins. There are different methods for the construction of a tree, however, not all of them are compatible with ASR. For example, distance-based algorithms such as neighbour joining lack an evolutionary model, which is necessary for ASR. There are different ASR-compatible families of algorithms, such as substitution (measuring the frequency of changes between two concrete nucleotides/amino acids), maximum likelihood (measuring the probability of the tree being true), Bayesian inference (iterative method to estimate both the topology of the tree and the parameters of the evolutionary model) and more complex evolutionary models. [16]

From all the available methods, maximum likelihood is the most frequently used type, due to its good ratio accuracy/computational cost. Bayesian methods can be more accurate, but are usually more computationally expensive. However, accuracy is highly dependent on the selected dataset. Likelihood is defined as the probability of obtaining the extant sequences using a given evolutionary model with the supposed topology of the tree. This definition can be applied to each branch of the tree, to find close ancestors, or to the whole tree, trying to maximize the likelihood of the common ancestor. Apart from this algorithm on sequences, some programs are also capable of using information provided from literature. [16]

The last step is the reconstruction of the ancestral sequence itself using the tree and the extant sequences, combined with an evolutionary substitution model, preferentially according to the one used in the construction of the tree. However, there is no consensus on the best substitution matrix for this purpose, so different ones have been used, with similar accuracy rates. For the common ancestors, their sequence is usually determined as the one with a highest likelihood score locally or inside the whole tree. [16]

Some programs are capable of both creating the tree and reconstructing the ancestral sequence. However, they are usually specialized in one of the two functions. For example, PAML is a program that can create trees and, from them, calculate their extant sequences, but its tree search and building algorithm are rather primitive, in a way that trees with more than ten sequences can take a really long time to be created [50]. Because of that, many of these programs accept trees in Newick format as input, in order to avoid this slow calculation and they perform only the reconstruction of the sequence. On the other hand, other programs allow for fast calculation of trees with maximum likelihood and a high number of sequences, but they cannot reconstruct ancestral sequences, such as PhyML [47].

Some of the reconstruction programs return as a result the ancestral sequence of the root and all nodes, the alignment of these with all extant sequences and the likelihood for each position of the sequence, allowing for a fast evaluation of the reliability of each amino acid in the reconstructed ancestral sequence[50]. This kind of approach has been used successfully for the study of past conditions, such as temperature or multisubstrate enzymes, finding some thermophilic variants, and for the study of the evolution of proteins or complexes of interest, such as in interactions between proteins and hormones [16].

Methods

Global workflow of the proposing prototype

The main objective of this work is the development of a prototype of computer program using existing software, empiric data and newly written code. The main programming language used in the program is Python 3.6.1 (<http://www.python.org>), with its extension Biopython 1.7 [51], which allows for an easier manipulation of biological sequences. The overall task of the program is proposing probable stabilizing mutations and returning them in descending order of probability of being stabilizing.

The program is composed of different modules, parts of the code focused on one particular characteristic of the protein. These modules can work almost as independent programs but, to reduce the required calculations, some modules can transfer information between them and they can also use files created by other modules. The way in which each module functions is explained below, as different characteristics require different approaches for its study.

As shown in figure 4, the program performs three consecutive tasks: Mutation proposing, mutation evaluation and mutation scoring. The expected input for the program are either PDB files or PDB codes for the retrieval of a PDB file from the online file exchange ftp server of the wwPDB (Worldwide Protein Data Bank: <ftp://ftp.wwpdb.org/pub/pdb/data/structures/divided/pdb/>). However, there is also a function in the program for testing and training purposes that allows for the usage of a mutation list and a PDB file.

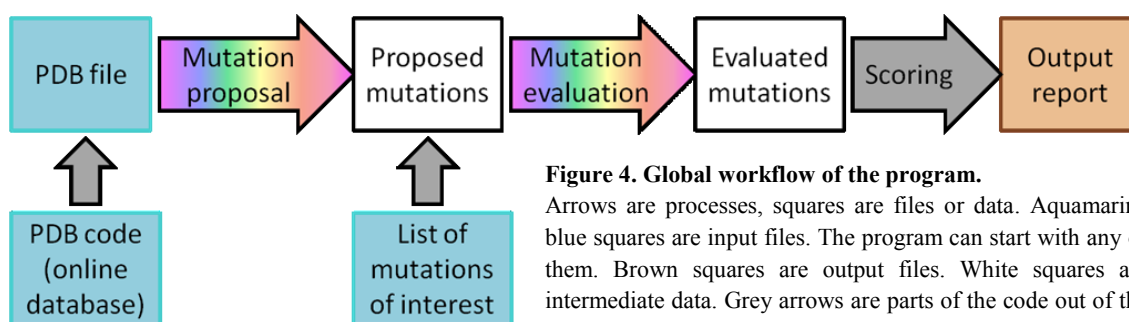


Figure 4. Global workflow of the program.

Arrows are processes, squares are files or data. Aquamarine blue squares are input files. The program can start with any of them. Brown squares are output files. White squares are intermediate data. Grey arrows are parts of the code out of the six main modules. Rainbow arrows are processes in which all or the majority of the six main modules are involved, representing each colour one module: Violet/purple for sequence analysis, blue for alpha-helices, green for disulphide bonds, yellow for exposure, orange for acidic hydrogen-bonded residues and red for cavities and steric clashes. This colour code will be further used in figures 5-10.

The first task is an initial analysis performed by each module to propose probably stabilizing mutations. All six modules propose mutations whenever possible. Proposed mutations are already evaluated by the proposing module. If the same mutation is

proposed by different modules, each module will return the mutation but the coincident proposals will be merged in one, with all the information from each proposing module. For a better explanation on this subject, go to section “the PointMutation class”.

The second task is evaluation. As the effect of a mutation is the combination of all its effects in different characteristics of the protein, proposed mutations are evaluated not only by the proposing module(s) but also by the rest of modules, to check if there are important destabilizing concomitant effects and to obtain a score for the impact of the mutation on all the studied protein characteristics. After this task, proposed mutations can be filtered according to different parameters, if it is considered necessary. However, this is not performed by default.

The third task is scoring, using a logistic regression machine learning model, trained with existing mutation data. These scores will be used to rank the most probably stabilizing mutations and returning them as output.

Modules

Six main modules have been developed for the study of the different characteristics of proteins that have been considered. Five of them (sequence analysis, alpha-helices, disulphide bonds, exposure and acidic hydrogen-bonded residues) play a role in both mutation proposal and evaluation. One of them (cavities and steric clashes) only evaluates mutations. Because of this double role of most of the modules, they have at least two relevant functions: one for the proposal of mutations according to the studied parameters and one for the evaluation of those parameters on mutations proposed by other modules. However, the main modules have additional functions, such as exporting data that can be used in further evaluations instead of calculating it for each evaluation.

Apart from the six main modules, there are other complementary modules with important functions for the program, such as the retrieval of the PDB structure from the PDB online database, the scoring of the mutations with a logistic regression model and other testing and training purposes that will not be present in the final version of the program. Also, there is a module for the definition of a special programming class for the generated mutations, the PointMutation class. All of these modules are explained below and all the code written is shown in Appendix A.

The PointMutation class

In Python programming, a “class” is a group of instructions on how to create and use an “instance”, with specific properties. It also can contain information on how to perform class-specific operations. For example, in geometry, an example of “class” would be a circle, determined by its radius. Two circles of 3 and 5 cm of radius would be different “instances”. A rectangle would also be a “class”, defined by its long and short sides. In either example of “class”, the area of the geometrical shape can be calculated. However, the formula for the area of the circle is different from that of the rectangle. These formulas are examples of class-specific functions, as they can be called with the same name (area), but they perform different operations.

For this project, several classes have been used, such as protein sequences, structures, chains and alignments. All of them were implemented in Biopython [51] except one used to describe proposed and evaluated mutations, named as PointMutation class, that has been created for this Master thesis.

The PointMutation class refers to a mutation of a single amino acid into a different one. Its arguments (information necessary to define an instance of this class) are the *protein code*, the mutated *chain* of the protein, the *original* and *mutated* amino acid and the *position* of the mutation. An additional argument has been created to solve the fact that not all of the annotated sequences in PDB files start in amino acid 1. This is the argument named as *start number*. If all these arguments are the same in two mutations, then they are the same mutation, no matter whether the other properties of both instances are the same or not. To define a mutation in structure 1A2P, that starts its sequence with number 1 (*start number*) from the glycine (G in one-letter code) in the position 34 of the chain A to a tryptophan (W in one-letter code), the line of code would be as it follows:

```
mutation = PointMutation("G", "W", 34, "A", "1a2p", 1)
```

Most of the rest of properties are defined as blank by default, to be filled by further evaluation by the modules. These properties are *ancestral*, *consensus*, *helix*, *disulphide*, *acid_bonds*, *cavities*, *exposure* and *energy*, which will be explained further in their respective module’s section. When a module proposes a mutation, it immediately fills the evaluated property field. If two mutations are the same, as mentioned before, they can differ on these properties, as they can be proposed by different modules, so both mutations merge their properties by filling the blank fields of the first mutation with the

data from the second one. Introducing new information for these fields in an existing PointMutation object is simple. Using the last example, if its consensus value is 0.5, it would be added using the following code:

```
mutation.consensus = 0.5
```

Also, there are properties intended for testing, such as *measured*, which is reserved for storing the results of empirically measured $\Delta\Delta G$ values on mutations for training and testing purposes. The properties *m_average* and *m_sd* are used in case there is more than one measured value and they have their own class-specific function to be calculated. The last two properties show which module proposed the function and the score of the mutation according to the logistic regression trained model.

Class-specific functions are the *representation* function, that shows the mutation in a simple way, and *write_full_mutation*, which returns all data of the mutation and its evaluated properties in an ordered format for further analysis. The latter is specially used for training and testing. Other functions are used to check if all properties have been evaluated and, if they haven't, to start the process to evaluate them. It can also be used to determine which of the modules proposed the mutation. All these functions contribute to an easier storage and manipulation of the information about a given mutation in an ordered way.

Retrieval

In order to make possible for the program to function with just a PDB code, a retrieval function must be implemented. Biopython has a retrieval function from the Worldwide Protein Data Bank (wwPDB) database. This complementary module uses this function to download the file in the right PDB format (.ent), saves it in a directory and prepares the necessary folders for further analysis.

Sequence analysis

The sequence analysis module performs both the consensus and ancestral sequence analysis, as both of them work with the homologous sequences of the protein. However, the way in which they use them is different, as shown in figure 5. As some PDB files contain multiple chains with the same sequence, this module will only be executed once for each different chain, avoiding the repetition of the process if there are two chains with the same sequence.

The common part for both approaches is finding the family of homologous protein sequences. For this purpose, a BLAST search is performed in the web server of the

NCBI, using the program blastp [52], in the non-redundant protein sequence database, with gap costs in the alignments and with a minimum percentage of identity of 70% and a maximum expect value of 10^{-4} . Expect value is the probability of the result being positive at random. These parameters are very restrictive. Nevertheless, if more than 10000 results are found, only the first 10000 will be used.

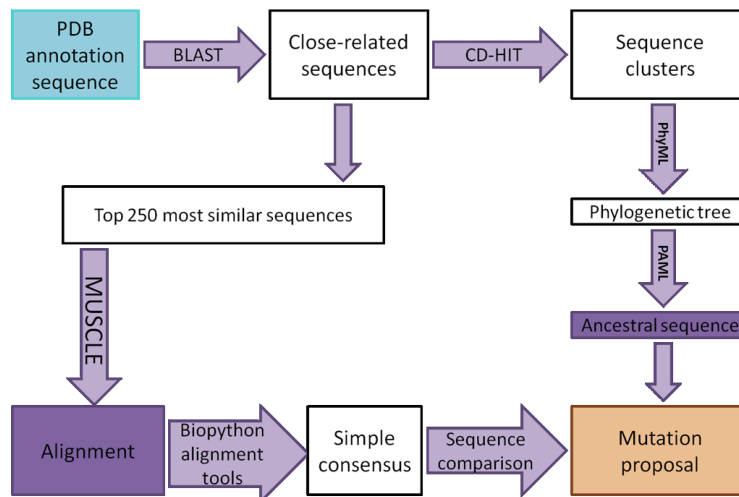


Figure 5. Workflow of the sequence analysis module.

The proposal function workflow is shown, as evaluation function can be thought of as an analogy of the final part of this process.

Same colour and shape meanings than in figure 4 are used, except for dark purple squares, which are the starting points for the evaluation functions. Because of this, the information related to the dark purple squares is exported and saved for a faster execution of the program Programs mentioned in the arrows will be described along the text.

Consensus sequence analysis takes only the first 250 results from the BLAST search, which are the 250 most similar sequences, to find any kind of consensus. As there is no evolutionary model behind this approach, a higher number of sequences could lead to a completely ambiguous consensus in some proteins with a small homologous family. These sequences are aligned using one of the most common multiple alignment programs, MUSCLE 3.8.31 [45] with the default options and saved as a text file in FASTA format.

After that, Biopython alignment analysis tools are used to read the file and generate a simple consensus with a minimum threshold of 0.3 (30%) as described in the consensus sequence theory section before. As a result, one consensus sequence is obtained. However, during multiple alignment, some gaps may have been introduced in the original protein sequence. Because of this, it is necessary to perform a pairwise alignment between the original sequence and the consensus to find differences between both sequences. This is achieved using Biopython global pairwise alignment tool with a BLOSUM62 alignment scoring matrix. Once both sequences are aligned, a comparison between them is performed and wherever in the sequence a difference is found, a mutation will be proposed, taking into consideration the number of gaps generated in

the original before that amino acid in the sequence and the number in which the annotated sequence starts.

Then, the mutation is scored for the consensus field. This is done using the multiple alignment file and follows the same process as the evaluation function. For the original sequence in the multiple alignment (with gaps), the program finds where the mutation is located and then counts in how many sequences that amino acid is the same than the original (*WT*), the mutated (*mut*) and in how many there is no gap in that position (*total*). With this data, a score is calculated according to the following function: (1)

$$Score = \frac{mut - WT}{total}$$

The evaluation function follows the same process, with only slight technical changes in the input and output of data, as the mutation is not proposed by this module.

Ancestral sequence analysis starts with all sequences retrieved from BLAST. As a high number of sequences is not good for the performance of phylogenetical programs, a first step is the sequence clustering using CD-HIT 4.6.8 [48], to reduce the number of sequences in use in a rational way, leaving sequences in a concrete range of identity percentage between them. For sequences with 100 residues or more, the minimum threshold for clustering is 90%, leaving all sequences in the range of 70-90% identity. However, for shorter sequences, this could be too restrictive, so the minimum threshold for shorter sequences is set to 80%. After the clustering, a multiple alignment of the sequences is performed with MUSCLE using default settings and its saved in a text file in PHYLIP sequential format, the required format in further phylogenetic programs.

The first phylogenetic program to use is PhyML 3.1 [47], for the fast construction of the tree. As discussed before, PAML [50] can both create the tree and generate the ancestral sequences, but it is very slow compared to most programs, so PhyML is preferred for tree building. First, slight changes in the names of the sequences of the multiple alignment are performed, to make the file compatible with the program. Then, PhyML is executed in amino acid sequence mode with the substitution model JTT [53] and the rest of parameters are set as default. PhyML exports a tree in a Newick-like format, compatible with most software except PAML, so another piece of code has been developed to convert the Newick-like tree into strict Newick format.

For its functioning, PAML 4.9e needs a multiple alignment file, a Newick format tree and a third file with all the parameters needed in the program. An example of this file is

provided in appendix C. Amino acids have been set as sequence type, the JTT substitution model has been set as rating model, all sequences are considered to be existing at the present and the program is running under a molecular clock hypothesis. The rest of the parameters are set as default. PAML returns a file with a multiple alignment between all extant and ancestral sequences, several analysis of the tree and an analysis for each position of the sequence of the probability of the ancestral sequence amino acid of being correct. For the proposal and evaluation of the mutation, these probabilities and amino acids for the common ancestor of all sequences (the root of the tree) are saved, as well as the original and ancestral sequence in the multiple alignment (with gaps). As the minimum percentage of identity used is 70%, it is improbable that the common ancestor is LUCA, given that most of the studies focused on LUCA use a 30% threshold [16].

As ancestral and original sequences are already aligned, direct comparison is possible. If a difference is found between them, the mutation from the original to the ancestral is proposed. As score of this mutation, the probability of this amino acid being correct in the ancestral sequence, reported by PAML, is used as score and assigned automatically to the ancestral field when proposing the mutation. This section of the module also considers gaps and the start number, as the consensus. The evaluation function works in an analogous way to the scoring process of the proposing function. With a given mutation, the evaluation function looks for it in the original sequences, taking gaps and the start number into account, and then looks for the same position in the ancestral sequence. If the mutated amino acid is the one in the ancestral sequence, the probability is assigned as score. If not, score is considered zero.

Alpha-helices

Alpha-helices module is a key module for the program, as it does not only evaluate and propose mutations for alpha helices, but also performs analyses of the structure that are further transferred to other modules, as shown in figure 6. Two main analysis paths are developed in this module: one for secondary structure and one for hydrogen bonds. Then, the information gathered by both modules is analyzed together for the proposal and evaluation of mutations in alpha-helices.

The secondary structure analysis is performed by DSSP 2.2.1 [54], which reads the PDB file and calculates different properties, such as exposure, phi and psi angles, estimated hydrogen and disulphide bonds and, using this information, determines the

secondary structure along the sequence. DSSP is executed with default settings and all residue relative exposures, tagged with their chain, one-letter code and position in the sequence, are exported for further use in the exposure module. Relative exposures are calculated as exposures returned from DSSP divided by the average exposure in the unfolded state for the corresponding type of residue, calculated by Estrada et al [35] using Alphasurf [55] and ProtSA [35]. As DSSP returns results that are 5 % higher than Alphasurf, relative exposure is divided by 1.05.

For the detection of helices, the same criteria as Leader et al [56] is used: a helix must have at least four consecutive residues predicted by DSSP as in alpha-helical conformation, apart from the N- and C-cap. Less than four would not be even a full turn of a helix. N- and C-cap are, respectively, the residue before and after the residues in alpha-helical conformation. After that, all helices, including N- and C-caps, are extracted for their analysis as individual entities. Each amino acid in a helix is tagged with its one letter code, chain, position and relative exposure, calculated as before.

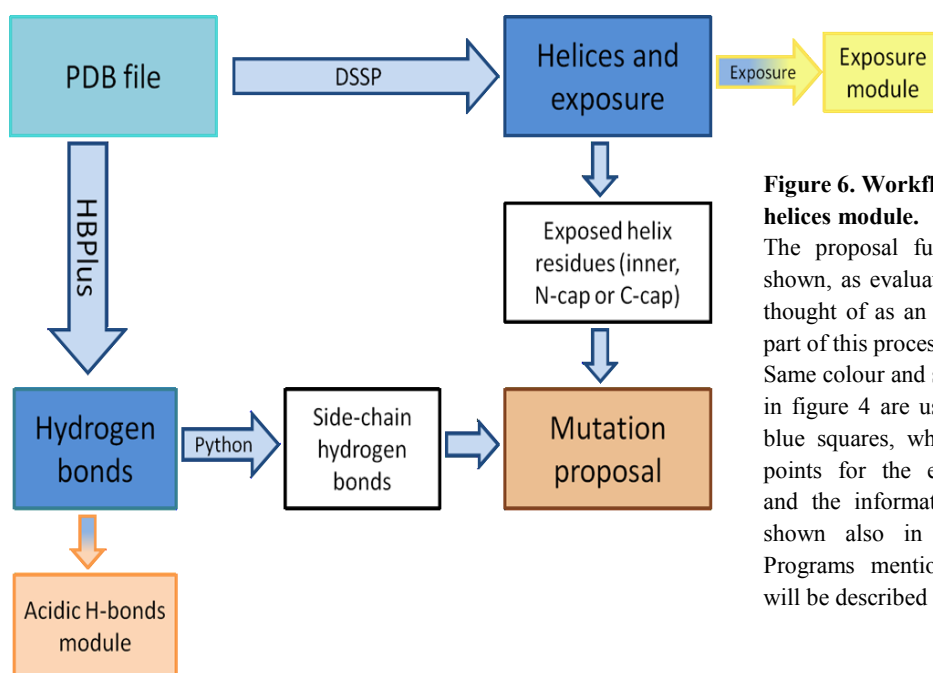


Figure 6. Workflow of the alpha-helices module.

The proposal function workflow is shown, as evaluation function can be thought of as an analogy of the final part of this process.

Same colour and shape meanings than in figure 4 are used, except for dark blue squares, which are the starting points for the evaluation functions and the information transfer points, shown also in figures 8 and 9. Programs mentioned in the arrows will be described along the text.

For the analysis of hydrogen bonds, HBPlus 3.06 [17] is used, with default settings. This program predicts hydrogen bonds using geometrical properties of the structure between important atoms for possible hydrogen bonds and returns a file which shows each possible hydrogen bond, its predicted strength and whether the hydrogen bond implies a side chain or a part of the backbone. As the backbone is not supposed to change much after the mutation, only the residues with hydrogen bonds on their side

chains are considered relevant and saved in a list. Also, in N-caps, it is frequent to find interactions between the N-cap side chain and the backbone of the helix. Because of this, only hydrogen bonds between the side chain of the N-cap or C-cap and other side chain are considered relevant, in order to preserve these relevant interactions.

Once this information is retrieved, mutations are proposed on the residues that fulfil all of the following criteria:

- The amino acid is not buried (relative exposure higher than 10%). If it is buried, the mutation may generate steric clashes or cavities.
- There are no hydrogen bonds affecting the side chain of the amino acid (for internal helical residues) or there are no side chain-side chain hydrogen bonds with the residue (for N- and C-caps).
- The amino acid is not one of the best options of amino acid for its position (Ala, Leu, Arg, Met, Lys for inner residues; Thr, Asp, Ser, Gly, Asn for N-cap or Gly, His, Asn for C-cap)

Depending on the location, the proposed mutated amino acid will be different, as the most stabilizing residue depends on the location. For inner residues, Ala is proposed; for N-cap, Asp, and for C-cap, Gly. To score the mutations, the effect on ΔG of each amino acid in each of the three locations is used. Three sets of data have been considered for each location. To determine which one is more reliable, comparison between sets has been done calculating correlation in a pairwise manner, using Pearson and Spearman coefficients, as both score and position matter for this program. Finally, Muñoz et al [27] dataset is chosen for all three positions. For an easier use, data has been represented relative to the most stabilizing residue for each position instead of Ala. The full dataset for each location is shown in appendix B.

After proposing the mutation, a score is assigned to it. Score is defined as the difference between effects on ΔG of the original and mutated residues, defined this way in order to have positive scores for more stabilizing mutations. As the most stabilizing residue is the mutated one in proposed mutations and all ΔG values are relative to the original residue, score is directly ΔG for the original residue. The score in this module is given in a list of three elements, being the first the type of position (N-, C-cap or inner); the second the calculated score, and the last whether there are hydrogen bonds involved or not. First and last elements are only for statistical purposes.

For the evaluation of the mutation, the program looks for the mutation in the list of helices. If the mutation is not found there, then a score of 0 is assigned for the calculated score and “X” is written for the other two elements. If the mutation is found, its location is determined (N-, C-cap or inner residue), and the score is calculated as stated before. The absence or presence of hydrogen bonds is also stated in the list of helices to transfer it to the mutation helix score field.

Disulphide bonds

Disulphide bonds module is based mainly in one program, Disulphide by Design 1.20 (DbD) [20]. This program is Windows-based and, as the rest of the modules are run in Fedora 26, the proteins have to be processed previously in this program, saved in a text file and transferred to a computer using Fedora 26. There is also a web server with Disulphide by Design 2 [21], which allows to deal with bigger proteins.

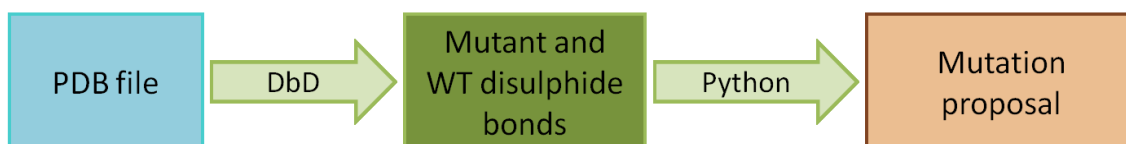


Figure 7. Workflow of the disulphide bonds module.

The proposal function workflow is shown, as evaluation function can be thought of as an analogy of the final part of this process. Same colour and shape meanings than in figure 4 are used, except for dark green squares, which are the starting points for the evaluation functions. Disulphide by Design (DbD) is executed in a Windows-based computer, different from the one where the rest of the program is executed. Programs mentioned in the arrows will be described along the text.

For the analysis of the structure, the PDB file is downloaded from the worldwide PDB database and analyzed by DbD with default settings. The results are saved as a text file and transferred to the computer using Fedora 26. There, some Python code allows for the interpretation of the results, distinguishing between disulphide bridges where a cysteine is already in the protein (single mutation needed) and where there are no cysteines in the original protein (double mutation).

For the proposal of mutations, the program firstly tries to propose single mutations. If there are no mutations found this way, then double mutations are proposed. For the proposal of single mutations, the program searches in the single mutation disulphide list and checks if the already present cysteine is part of a disulphide bond in the original protein. If it is, no mutation is proposed. If it is not, a mutation is proposed in the non-cysteine residue to change it into a cysteine. The score for this mutation in the disulphide field is the stabilizing effect calculated by DbD.

If no single mutations are proposed this way, then double mutations are proposed. As there are no cysteines involved in the original protein, there is no need to check whether they could change disulphide bonds, so all double mutations are proposed. As the responsible of the stabilization are both residues, disulphide score is half of the stabilization for each one and these mutations are evaluated by the other modules as if they were two different mutations, but their final score as a double mutation is the average between both scores and they are shown as only one proposal.

The first step of the evaluation function is checking whether a cysteine was in the original or mutated protein in that position. If not, the score is automatically 0, as no implication in disulphide bonds is possible without cysteines. Then, it analyzes if the mutation is involved in a disulphide bond. If it is, the score will be the stabilization effect calculated by DbD, being positive if the bond is being created with the mutation, or negative if destroyed. If there is no implication in disulphide bonds, the score is also 0.

Exposure

Exposure module starts its analysis with information transferred from alpha-helices module and filters it in order to propose and evaluate mutations, then it transfers data to the acidic hydrogen-bonded residues module, as shown in figure 8. It also uses the information of the amino acid type for this purpose.

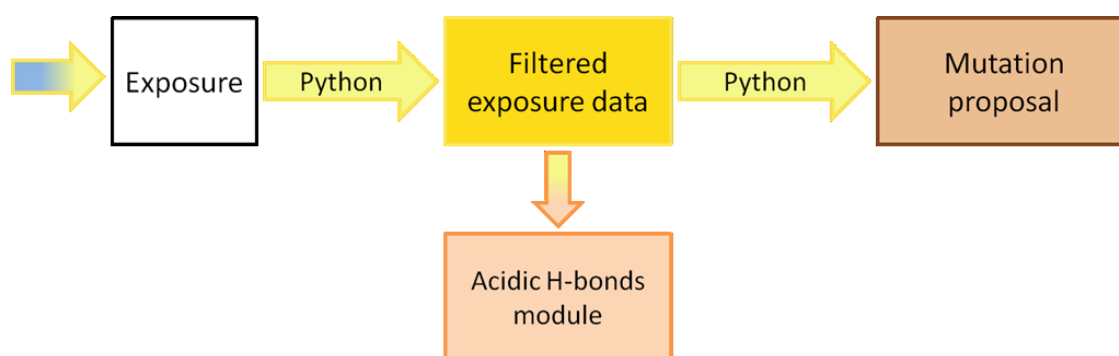


Figure 8. Workflow of the exposure module.

The proposal function workflow is shown, as evaluation function can be thought of as an analogy of the final part of this process. Same colour and shape meanings than in figure 4 are used, except for dark yellow squares, which are the starting points for the evaluation functions. The blue-yellow arrow is the transfer point shown in figure 6, the yellow-orange arrow is another transfer point, connecting with figure 9. All this module works with Python code.

First step is filtering exposure data. This step is necessary due to the process of calculation of the relative exposure. Relative exposure is calculated dividing absolute exposure by the average of exposure of that type of residue in an unfolded protein. Because of this, residues without bonds in both their amino and acid groups will be

more exposed, as one of their sides is not linked, making the calculation of the relative exposures inaccurate. This does not only happen for N- and C-terminus residues, but also for some segments that are intrinsically unfolded or too mobile, so that they do not appear in the PDB model. This way, these residues with inaccurate relative exposures are filtered out, and only the accurate ones are analyzed.

For the analysis, two different types of mutation are distinguished: overexposed apolar residues and buried polar residues. Buried residues are considered to be those with a relative exposure under 15%. This value has been set by calculating the relative exposure from the absolute exposure threshold used by Ayuso-Tejedor et al. [33] in their work. Only mutations shown as stabilizing in this article are proposed (i.e. mutations from buried Asn or Gln to Leu).

Overexposed residues are those with a relative exposure over 100%. For the proposal of these mutations, all apolar residues are considered except for Gly and Ala, due to its small size and higher flexibility (only in Gly), and Pro, due to its secondary amine for the amino group, which makes it more rigid. Cys is apolar unless ionized, but it can form disulphide bonds, so it is neither considered. Proposed mutations are from apolar overexposed residues to similar size and structure polar ones: Val to Asn; Phe to Tyr; Leu, Ile and Met to Gln.

For the score, some assumptions have to be made to be able to use the equation for $\Delta\Delta G$ estimation described by Ayuso-Tejedor et al. [33], which is the following:

$$\Delta\Delta G = 0.0276 \cdot \Delta\Delta ASA_{Apolar} + 0.0072\Delta\Delta ASA_{Polar} + 1.083 \quad (2)$$

Where $\Delta\Delta ASA$ is the buried accessible to the solvent area. As the exposure in DSSP is explored to a residue level instead of atomic level, some assumptions have been made to use data from DSSP with a simplified formula derived from (2). First assumption is that all the backbone is apolar and is responsible for a surface of the size of an alanine residue, as all residues (except glycine) are extensions of alanine. Second assumption is that all parts of the residue are buried in the same proportion, being that proportion the complementary of the relative exposure. That way, the buried area can be calculated as the average exposed area of a residue times a factor of one minus relative exposure. Third assumption is that side chains are completely polar or apolar, depending on the type of residue (polar or apolar, respectively). As these assumptions are not very accurate, the result is not going to be $\Delta\Delta G$, but can be used as a score. To simplify it

more, the independent term 1.083 is taken out of the formula. Two formulas are obtained, depending on the polarity of the original and mutated residue. If the original residue is polar and the mutated apolar, formula (3) is used. If it is the other way round, formula (4) is used.

$$Score = [0.0276 \cdot (ASA_{Ala} - ASA_{WT}) + 0.0072 \cdot (ASA_{Mut} - ASA_{Ala})] \cdot (1 - RE) \quad (3)$$

$$Score = [0.0276 \cdot (ASA_{Mut} - ASA_{Ala}) + 0.0072 \cdot (ASA_{Ala} - ASA_{WT})] \cdot (1 - RE) \quad (4)$$

Where ASA is the average exposure to the solvent in an unfolded protein of a type of residue, being this residue Ala (alanine), *WT* (the residue of the original protein) or *Mut* (the mutated residue), and RE is the relative exposure as a decimal. These functions are originally designed only for buried polar residues. However, as for overexposed apolar residues (1-RE) becomes negative, it can be used also as score for them, as the behaviour is the opposite than for buried residues both in the formula and in the empirical data and this solves the problem of the lack of quantitative studies for scoring mutations of overexposed apolar residues.

The evaluation function checks if the original and mutated residues are polar (Ser, Thr, Tyr, Asn, Gln) or apolar (Val, Leu, Ile, Phe, Met). If both are of the same kind, a value of 0 for exposure is assigned. If not, the relative exposure is obtained from the filtered data. If relative exposure for that residue is filtered out, a 75% relative exposure is assigned, so that in next step is given an exposure score of 0. If relative exposure is under 25% or over 100%, the formula is applied to calculate the exposure score. If not, a value of 0 is assigned.

Acidic hydrogen-bonded residues

Acidic hydrogen-bonded residues module takes information from both the alpha-helices and exposure modules to propose and evaluate mutations. From the alpha-helices module, hydrogen bonds implying acidic residues (Asp, Glu) or their isosteric neutral equivalents (Asn, Gln) is transferred. From the exposure module, the filtered list of relative exposures is used. A quick search with Python of both datasets is enough for proposing and evaluating mutations.

For the proposal function, the program searches in the list of acidic and equivalent hydrogen-bonded residues for the acidic ones (Asn, Gln). Once they are found, a search for these residues is performed in the exposure list. If they have been filtered out by the exposure module, a relative exposure value of 0 is assigned. If their relative exposure

value is over 85%, a mutation to their equivalent is proposed (i.e. Asp to Asn, Glu to Gln) as performed by Irún et al [19].

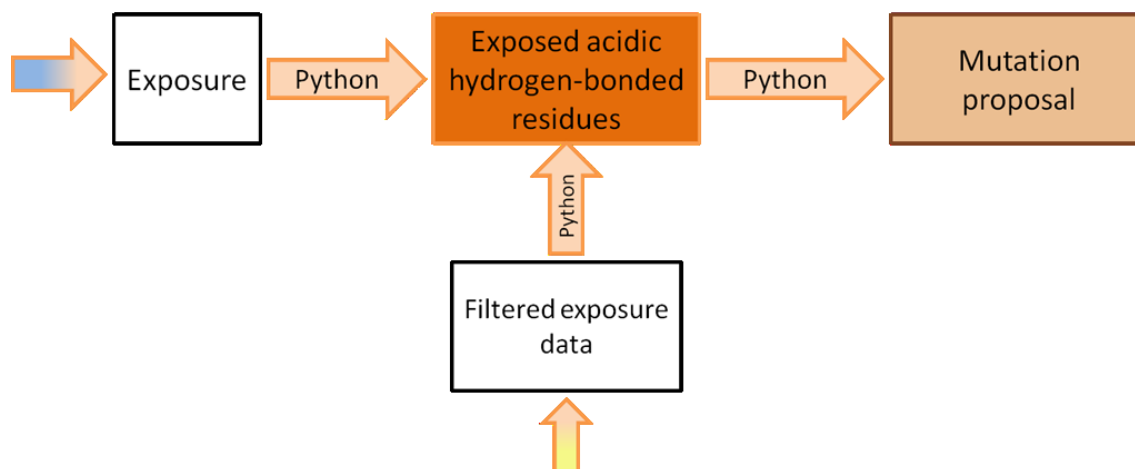


Figure 9. Workflow of the acidic hydrogen-bonded residues module.

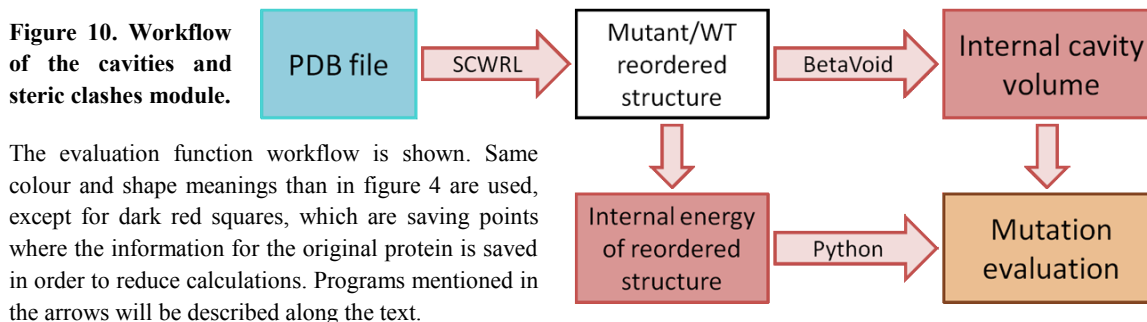
The proposal function workflow is shown, as evaluation function can be thought of as an analogy of the final part of this process. Same colour and shape meanings than in figure 4 are used, except for dark orange squares, which are the starting points for the evaluation functions. The blue-orange arrow is the transfer point shown in figure 6 and the yellow-orange arrow is the transfer point shown in figure 8. All this module works with Python code.

For the score, only three values are used: 1, 0 and -1. This is due to the fact that only three mutations on the same protein have been studied with this approach by Irún et al [Irún]. This is not enough to develop scoring formulas or to find differences between stabilizations on both types of mutation, so a score of 1 is a change from an acidic exposed hydrogen-bonded residue to a neutral amide and a score of -1 is a mutation in the opposite way. A score of 0 means no change in these terms. Because of this, the proposal function automatically gives an acid_bonds score of 1 to the proposed mutations.

The evaluation function searches for the mutation in the list of acidic or equivalent amide hydrogen-bonded residues. If it is there, it searches for the mutation in the exposure list. If there is a exposure value over 85 %, it checks whether the mutation corresponds to a value of 1, 0 or -1. For the cases that don't have some of the values described before, its assigned value is 0.

Cavities and steric clashes

The cavities and steric clashes module is the only one of the main six modules that does not have a mutation proposal function, due to the difficulties in automating cavity analysis programs. Two different scores are returned, one for cavities and another one for steric clashes, but the programs are executed sequentially.



The evaluation function workflow is shown. Same colour and shape meanings than in figure 4 are used, except for dark red squares, which are saving points where the information for the original protein is saved in order to reduce calculations. Programs mentioned in the arrows will be described along the text.

SCWRL 4.0 [38] is a program that allows for the reorganization of the backbone and side chains of a molecule in order to minimize the energy of the structure, so that lower energy models are created. It also can be used to introduce mutations in a model by introducing the mutated sequence. For the original protein, SCWRL is used as default, fixing the whole backbone of the protein. For the mutated sequence, flexibility is allowed only for the mutated residue. Apart from the model, the output of the program gives the minimal energy achieved for the structure. This energy can be retrieved with Python and the score for the energy field for a mutation is the difference between the energy for the mutated model and the original protein. As the energy for the original protein is used for all mutations, it is saved to reduce computation time.

Betavoid 1.1 [42] can calculate the total volume of cavities in the protein from a PDB file. However, the result for the model of the original protein before and after SCWRL without introducing mutations can vary greatly. Because of that, BetaVoid is applied on the model of the original protein after SCWRL and in the mutated model, obtained by SCWRL. As before, the score for the cavities field is the difference of the total cavity volume of the mutated model and the original protein (after SCWRL) and data for the original protein is saved in order to reduce computation time.

Scoring model

The scoring module is based on the Logistic regression model, trained by machine learning methods. This model is based on a linear multiparametric equation that calculates a global score from partial scores. In this work, the equation is like this:

$$\begin{aligned} \text{Score} = & \lambda_{anc} \cdot Sc_{anc} + \lambda_{cons} \cdot Sc_{cons} + \lambda_{hel} \cdot Sc_{hel} + \lambda_{dis} \cdot Sc_{dis} + \lambda_{acb} \cdot Sc_{acb} \\ & + \lambda_{exp} \cdot Sc_{exp} + \lambda_{ene} \cdot Sc_{ene} + \lambda_{cav} \cdot Sc_{cav} + \text{intercept} \end{aligned} \quad (5)$$

Where Sc is the partial score for each property field in the PointMutation object, and λ and intercept, parameters adjusted using machine learning. *Anc* is for *ancestral*, *cons*, for *consensus*; *hel*, for *helix*; *dis*, for *disulphide*; *acb*, for *acid bonds*; *exp*, for *exposure*; *ene*, for *energy* and *cav* for *cavities*.

After scoring the mutation, the program can work using two approaches: binary prediction or probability calculation. For the binary prediction, a threshold for the score is set, usually corresponding to 50% probability. Then, the mutations are evaluated. If the mutation is over the threshold, it will be considered as stabilizing. If not, it will be considered non-stabilizing.

The probability calculation goes a step further and returns a quantitative measure of the probability of the mutation being stabilizing, a relative score (*r-score*). To calculate the *r-score*, the logistic function, that gives its name to the method, is used. This function takes as input any real number (in this work, the score) and returns a value between zero and one (the probability in this case). The *r-score* for a mutation A to be stabilizing is defined with the following function:

$$r_A = \frac{1}{1 + e^{-\text{Score}_A}}$$

Where Score_A is the score for mutation A, as calculated in the equation (5) and being $P(\Delta\Delta G > 1|A)$ the probability of mutation A being stabilizing.

As the probability calculation approach offers a more informative result, this is the approach implemented in the program, offering as a result the *r-score* as a percentage rounded to an integer.

The module first uses a dataset to train the model and adjust all λ and the intercept, as described below in the prediction training and testing section and builds a logistic regression model with this parameters. Then, for each proposed mutation, after it is evaluated by all modules, takes its values for all property fields and calculates the probability of that mutation being stabilizing.

This module also offers the functionality of ordering all mutations from higher to lower probability, in order to give an ordered output.

Training and testing the programme

Database

The scoring function is based on machine learning, a group of algorithms that adjust their parameters to different functions using already known data in a process known as training. To assess the accuracy and good functioning of these methods, several approaches can be used, but the most straightforward is checking for the accuracy of predictions in a test group.

For both training and testing, a group of mutations whose effect on stability has been previously determined is needed. For this purpose, two groups are extracted from ProTherm [57], a collection of numerical data of thermodynamic parameters for wild type and mutant proteins. This work was already done by Capriotti et al [11], but the database has been expanded since then. A first filtering process is performed to retrieve all mutations before distributing them in two groups. Criteria for this process is:

- The protein must have only one amino acid-change mutation.
- Data for $\Delta\Delta G$ must be provided, as well as the PDB code for the protein.
- Temperature for the experiment must be between 15 and 35 °C (288 and 308 K).
- pH value for the experiment must be between 6 and 8.

After that, data is distributed in two groups: Mutations used by Capriotti (both in their testing and training groups, with data until April 2007) and not used by Capriotti (New, with data until the latest release, February 2013). Statistics from this datasets are shown in table 1. As some mutations are measured several times in different conditions, the average and standard deviation has been calculated in order to have only one value for each unique mutation.

Database	Unique proteins	Unique mutations	Mutations	Multiplicity
Capriotti no filter	68	1446	2067	1,429
Capriotti	57	865	1109	1,282
New no filter	150	3387	5495	1,622
New	108	1950	2667	1,368

Table 1. Statistics of the database

Statistics are shown for both groups before (named as “no filter”) and after the pH and temperature filtering. Unique proteins shows the number of different proteins in each group; unique mutations, the number of different mutations; mutations, the total number of mutations in a group and multiplicity is the ratio between mutations and unique mutations. As it can be seen, filtering reduces the multiplicity of measurements for a single mutation

For the classification of the mutations, same criteria as Capriotti [11] will be applied. $\Delta\Delta G$ in the range of -1 to 1 kcal/mol (both included) are considered to be derived from neutral mutations, due to the experimental noise of the measurements. $\Delta\Delta G$ higher than 1 kcal/mol are from stabilizing mutations and lower than -1 kcal/mol, from destabilizing. In the context of this work, only two groups are formed: stabilizing mutations and non-stabilizing mutations (comprising neutral and destabilizing).

Due to the multiplicity of values, some mutations can have values both over and under the threshold of 1 kcal/mol, so further filtering is needed. For mutations with an average over 1 kcal/mol, they must fulfil that $\overline{\Delta\Delta G} - SD(\Delta\Delta G) > 0$, where $\overline{\Delta\Delta G}$ is the average and $SD(\Delta\Delta G)$ is the standard deviation. For mutations under 1 kcal/mol, they must fulfil that $\overline{\Delta\Delta G} + SD(\Delta\Delta G) < 1$. Mutations that don't fulfil these rules are discarded.

If the distribution of the measured $\Delta\Delta G$ is studied for the whole filtered database, it can be seen that there is a clear predominance of destabilizing mutations, as shown in figure 11A. This can induce bias in the training and testing process, so Capriotti et al developed a method to make a symmetrical database, based on the assumption that if a mutation from a residue A to other residue B has a $\Delta\Delta G_{A\rightarrow B}$, the mutation from B to A would have a $\Delta\Delta G_{B\rightarrow A}$ equal to $-\Delta\Delta G_{A\rightarrow B}$. This makes an almost symmetrical database, as shown in figure 11B. Most of the scores of the program are also reversible, in the sense that the score for A \rightarrow B is the opposite of B \rightarrow A, except for ancestral score, where it must be recalculated.

However, this solution can be creating a new bias too. As the sequence analysis module finds the family of homologous proteins from the original sequence, many of the new

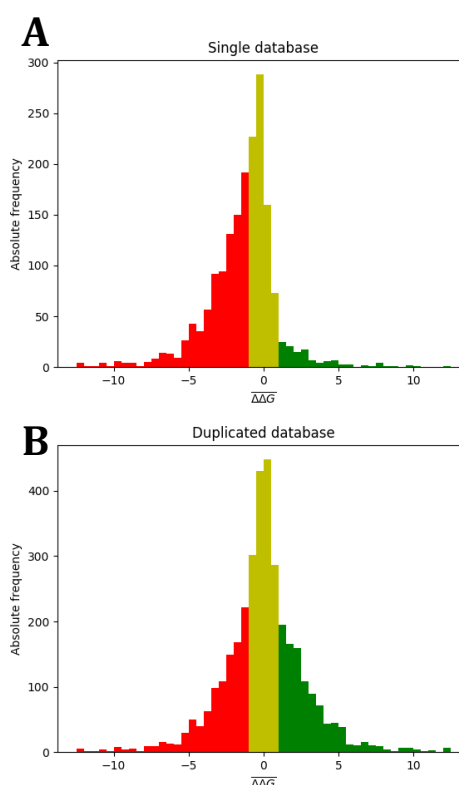


Figure 11. $\Delta\Delta G$ distribution in the database. Data for databases after all filters and after discarding mutations for proteins unable to work in the program (see results section for information on this). In red, destabilizing mutations ($\Delta\Delta G < -1$), in yellow, neutral mutations ($-1 \leq \Delta\Delta G \leq 1$), in green, stabilizing mutations ($\Delta\Delta G > 1$) A) Distribution for the full database before creating reversed mutations. B) Distribution with reversed mutations. No significant differences were found between Capriotti and New databases (data not shown)

reversed mutations can have a positive score due to the return to the original sequence instead of having a real evolutionary meaning. Because of this, two databases with original mutations (single, “s” in short) and both original and reversed mutations (duplicated, “d” in short) are saved for each initial group: Capriotti (“C” in short) or New (“N” in short). This generates four groups to be used in further steps: sC, sN, dC and dN.

Evaluation statistics and troubleshooting

Before training, a first statistical analysis of the mutations is performed to find out if there is a score for a certain module that is positive for all stabilizing mutations, to stop evaluating mutations with a negative partial score for that module. For this purpose, the program is run in mutation evaluation mode for all mutations of the full database (i.e., of dC and dN) and the results are saved in a text file for further analysis. At this point, some unforeseen errors appeared for some proteins. Troubleshooting for this cases was performed, preparing the program for a wider range of input files. However, some errors could not be avoided, and 12 proteins were discarded from the database. Further information about errors and troubleshooting can be found in the results section.

Data saved in this process contains the pdb code of the protein, the chain with the mutation, the original and mutated residue one-letter code, its position in the chain, the score for the ancestral, consensus, helix, disulphide, acid_h_bonds, exposure, energy and cavities field, and the average and standard deviation of the empirical measurements of $\Delta\Delta G$. From this file, data of mutations considered as stabilizing are extracted and saved in an Excel (.xlsx) file for statistical analysis of the relative frequency of positive and negative values for each parameter in order to find a filtering rule, if possible.

Prediction training and testing

Training and testing starts also with the files created in the last section, with the four databases described in the database section. Usually, the group with a higher number of elements is used as training group, while the other is kept as testing group. However, in this work a slightly different approach is going to be used to test if the results from training can be extrapolated to other groups of proteins. For this purpose, several training and testing processes will be performed and the obtained parameters of the model will be compared among them.

Four training and testing processes will be run, each one being trained with one of the four databases and then tested with their opposite database with the same type of mutations (simple-trained models test with simple databases, duplicated-trained models test with duplicated databases). Then, the parameters for each model are extracted and saved for further use. This process will be performed with the Python package scikit-learn [58].

For assessing the accuracy of each model, a ROC (Receiver Operating Characteristic) curve is calculated for each trained model with their corresponding test group. This curve shows the ability of the binary classification model to predict at different thresholds. This curve plots the true positive rate against the false positive rate. At each false positive rate value, a different threshold is calculated, and with this threshold, the true positive rate is calculated. This function is also present at the scikit-learn package, and it can be plotted with ggplot [59]. A diagonal line with the same value for true and false positive rate is included in all plots, as this would be the behaviour of a random predictor. A better-than-random predictor would be the whole plot over this line.

Visual inspection of the curve is the most informative method for comparison between curves. However, sometimes it can be difficult to determine which one is better. Because of this, a numerical measure of how good the prediction is can be derived from the ROC curve. This measure is the area under the curve (AUC) and has a value of 0.5 for random predictors and of 1 for a perfect predictor. This is not a perfect indicator, as is assessing the whole false positive space, even in regions that are not going to be used in normal studies [60], but it can be used as a comparison between models.

To ensure that the differences between groups are due to the training group and not to the testing group, another training and testing process is performed. From the best training/testing pair (dN/dC in this work), the training and testing groups are taken and used with other testing and training groups, respectively. In this case, dN/sC and sN/dC models are evaluated.

Proposal testing

Once the best model is selected, it can be used for the scoring of mutations and the calculation of the probability of each mutation to be stabilizing. Nevertheless, the accuracy of the whole program can vary from the prediction accuracy, as the program is already using a subset of the whole mutational space, the proposed mutations.

To correctly estimate the accuracy of the whole program, the program is run with the 10 proteins with most mutations in the database. Mutations are proposed, evaluated, scored and ordered from higher to lower score. Then, the database with only original mutations is searched for the proposed mutations to get their measured $\Delta\Delta G$ empirical value, if available. As the percentage of stabilizing mutations in the database is low, the ratio of percentages of mutations with positive $\Delta\Delta G$ in the proposed mutation list (or in subsets of this list) and in the original database will be also calculated.

Some other statistics will be collected also from this process, as the number of mutations proposed for each structure and by each module, the number of proposed mutations over certain probability thresholds (10, 25 and 50 %) and the maximum probability value achieved for a proposed mutation in each structure. Also, the time the program takes to analyze a structure and all proposed mutations (execution time) is shown. This time was measured using the Python `dateutil 2.6.0` module in an Asus F540LA-XX030T laptop with an Intel® Core™ i3-4005U processor (2 Cores, 3M Cache, 1.7GHz) using Fedora 26 as operative system.

Results

Alpha-helix scoring comparison

Three different sets of data for each type of position have been considered, published by different researchers. Pairwise comparison has been performed. First, a linear correlation has been found for all pairs, as shown in figure 12 for one example. The rest of graph are not shown. For further study of the correlation, Pearson and Spearman correlation coefficients have been calculated. While Pearson coefficient measures correlation between numerical values, Spearman coefficient measures correlation between ranks of the ordered values, in case a set of data was badly scored but well ordered. This results are shown in table 2.

According to this data, the most appropriate scoring model for N-cap and C-cap is the dataset from Muñoz et al [27] and Fernández-Recio [29] for internal positions, as they show a higher correlation with the other two datasets that both between them and has scores for all residues in all positions. Further discussion on the election of datasets can be found in the discussion section below.

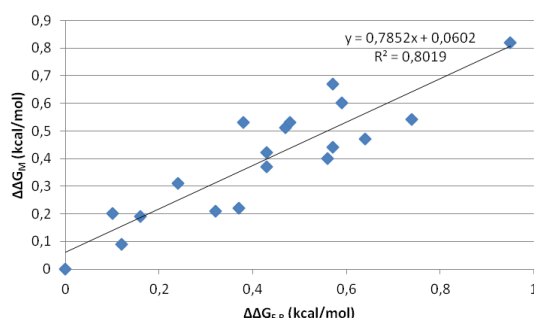


Figure 12. Comparison of alpha-helix scoring datasets. $\Delta\Delta G$ is the change in ΔG from a protein or peptide with a given residue to its homolog with alanine. This value is represented on both axes, but each one is from a different dataset. Each spot is the value of $\Delta\Delta G$ for a given type of residue in both datasets. By definition, $\Delta\Delta G$ of alanine is (0,0). In this example, the values for the internal residues of the helix by Fernández-Recio et al. [29] are on x-axis, and the values for internal residues by Muñoz et al [27] are on y-axis. Linear regression, its equation and R^2 are shown on the plot.

A	Pair	Pearson	Spearman
	Fernández/Muñoz	0,8955	0,8792
	Fernández/Pace	0,9588	0,9183
	Pace/Muñoz	0,9004	0,8274

B	Pair	Pearson	Spearman
	Serrano/Muñoz	0,8443	0,8354
	Serrano/Doig	0,6456	0,7201
	Doig/Muñoz	0,8806	0,8669

C	Pair	Pearson	Spearman
	Serrano/Muñoz	0,7068	0,8975
	Serrano/Doig	0,1845	0,3000
	Doig/Muñoz	0,2955	0,1122

Table 2. Correlation between dataset pairs.

A) Internal residues of the helix. B) N-cap. C) C-cap. Both Pearson and Spearman correlation coefficients are shown. Each dataset is named after its first author's surname. Data taken from Fernández-Recio et al. [29], Muñoz et al. [27], Pace et al. [28] Serrano et al. [31] and Doig et al.[30]

Running the program. Troubleshooting.

In the first run of the program for evaluation of the mutations in the database, only 58 out of 108 PDB files were analyzed correctly without stopping or showing any error message. All causes of error are gathered in table 3 and further description of them can

be found below. These are the main causes of trouble, from the most frequent to the least:

BetaVoid segmentation fault. This error is caused by BetaVoid, in the cavities and steric clashes module. No explanation of this error can be found in the manual or webpage, but it seems to be related with the input file name. The input files were previously named as “pdbXXXXSCWRL.ent”, where XXXX is the PDB, and this error only happened for PDB codes starting with a number other than 1. Also, when re-running the program, this error appeared for some files with an already existing output file. Because of this, two changes have been introduced for the final program: files are now saved with the name “pdbSCWRL.ent” in a folder named as the PDB code and this folder is automatically deleted after executing the program.

Incomplete backbone structure. This error is returned by SCWRL, in the cavities and steric clashes module, when some atom of the backbone of the protein is not defined. The only solution for this problem is introducing a complete structure, as it has been performed for the protein with PDB code 1BNI, using another structure of the same protein (PDB code 1A2P)

Repeated name. Error returned by MUSCLE, when saving the alignment in Phylip format. It happens when CD-HIT returns two different sequences for the same homolog sequence. A solution to this has not been found yet.

Mutation index out of range. Error returned by the consensus mutation evaluation function. This error is returned for PDB where the starting annotated sequence amino acid is not numbered as 1. This causes that, in a structure like 1FNF, in which the first annotated amino acid is 1142, the first version of the program tried to reach position 1142 in the sequence, that does not exist, returning this error. This is solved by adding a new property to the PointMutation class, the start number, and calculating the position of the mutation in the annotated sequence using this piece of code:

```
self.seq_pos = self.position - int(start_number) + 1
```

Ambiguous amino acid in family. Error returned by CD-HIT, in the sequence analysis module, when one of the sequences retrieved by BLAST contains an ambiguous amino acid such as “B” or “Z”. To solve this, a step of filtering after the retrieval is performed using the following piece of code, where hsp.sbjct is the sequence:

```
if "Z" not in hsp.sbjct and "B" not in hsp.sbjct:  
    sequences.append(hsp.sbjct)
```

Type of error	Number of files	PDB codes
BetaVoid segmentation fault	19	1CYC, 2ADA, 2AFG, 2AKY, 2CI2, 2HMB, 2HPR, 2IFB, 2LZM, 2TRX, 2TS1, 2WSY, 2ZTA, 3BLS, 3D2A, 3MBP, 3PGK, 4BLM, 5AZU
Incomplete backbone structure	7	1BNI, 1FC1, 1H7M, 1LRP, 1QGV, 1ZNJ, 2BRD
Repeated name	6	1A43, 1AXB, 1LVE, 2CRK, 2IMM, 3HHR
Mutation index out of range	6	1AG2, 1AMQ, 1FNF, 1KDX, 1MBG, 2Q98
Ambiguous amino acid in family	4	1BVC, 1I5T, 1RTB, 1SHG
Multiple model	4	1AJ3, 1CEY, 1LS4, 1UWO
PhyML error.	2	1IR3, 1ONC
Too big for DbD	1	1AON
PAML	1	1ACB

Table 3. Causes of error in the first run of the program.

All causes of error are listed in the table. The causes in the first column are further described in the text, as well as ways to solve them. The number of files from which each error has been returned and their PDB codes are also shown in the table. PDB codes in bold are those whose problem has been correctly solved by applying the solutions described in the text.

Multiple model. This warning is shown by Disulphide by Design. It appears for structures obtained using NMR or other techniques in which multiple models are obtained. However, this warning does not stop the program from functioning correctly, only using the first model, so no further action is required.

PhyML error. This error is returned by PhyML when less than three sequences are introduced as input. This error is common after a CD-HIT clustering process, so it was solved before the first testing, by making a conditional path. If CD-HIT forms less than 3 clusters, CD-HIT will not be used. This solved the problem for most proteins except for the two shown in table 3. In these cases, they retrieved the maximum number of sequences, but they were all in the same cluster. To solve the problem, the maximum number was raised up to 40000 sequences for these special cases, but it only worked for one of them.

Too big for DbD. This error is returned by the program Disulphide by Design when using 1AON PDB file as input. This file contains a chaperonin complex with 14 chains of 547 residues and 7 chains of 97 residues. This is a very big structure and it cannot be processed by the Windows-based computer program, but it can be processed with the web server Disulphide by Design 2.0. However, the output format is different and a special option for processing this data has been develop in the program.

PAML. This is a specific unknown error returned for chain B of structure 1ACB. As no mutations in the database were found in chain B, the evaluation program skipped this chain to avoid stopping the workflow.

After solving all this issues, 96 proteins (88.9%) were analyzed. This 96 proteins covered in total 93.4 % of the mutations in the database for further training and testing.

Mutation database evaluation statistics

From the 3640 mutations evaluated, only 1030 fulfil the conditions to be considered stabilizing. The evaluation scores for each field of these stabilizing mutations are classified according to their values in positive, negative or zero. Statistics are shown in table 4.

Score	% positive	% zero	% negative
Ancestral	77,38%	22,62%	0,00%
Consensus	86,89%	1,94%	11,17%
Helix	16,12%	66,99%	16,89%
Disulphide	0,00%	100,00%	0,00%
Acid_bonds	0,10%	99,90%	0,00%
Exposure	5,73%	91,46%	2,82%
Energy	50,78%	24,76%	24,47%
Cavities	10,97%	65,63%	23,40%

Table 4. Statistics for measured stabilizing mutations. All 1030 evaluated mutations with average measured $\Delta\Delta G$ higher than 1 and standard deviation lower than $\Delta\Delta G$ have been analysed for this table. As all scores (except ancestral) can be positive or negative, classification has been done according to this, making three classes: positive, negative and zero. All results are shown as percentage of the total stabilizing mutations.

Prediction accuracy

Six different ROC curves have been calculated. The first four curves (figures 13A-D) are calculated to propose the best training method and the other two (figures 13E-F), to check to which amount the observed effect is due to the training group and not to the testing group. The adjusted parameters for each of the training groups, corresponding to figures 13A-D, are collected in table 5.

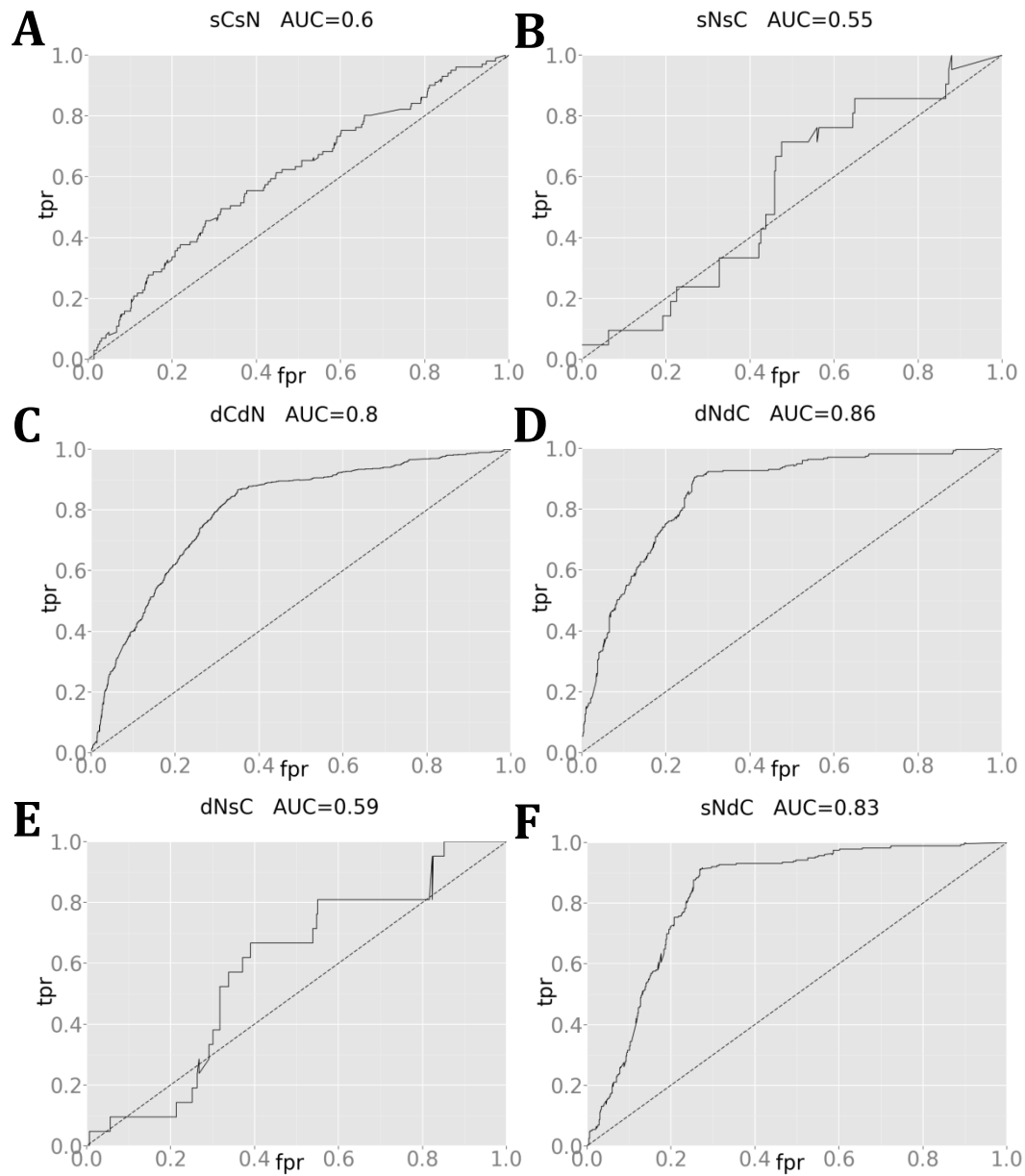


Figure 13. ROC curves for different training/testing combinations.

ROC curves are written as training/testing groups, being “s” for single datasets and “d” for duplicated, “C” for the groups taken from Capriotti et al [Capriotti] and “N” for the new ones. Groups used are the following: A) sCsN, B) sNsC, C) dCdN, D) dNdC, E) dNsC and F) sNdC. In this graph, the false positive rate (fpr, x-axis) and the true positive rate (tpr, y-axis) are plotted. The diagonal dashed line is the ROC curve for a random prediction. The continuous black line is the ROC curve for the model. The Area Under the Curve (AUC) is shown in the title of each curve.

Training group	sC	dC	sN	dN
Ancestral	0,46013	0,26838	0,06220	0,21999
Consensus	0,57365	1,71494	1,22157	1,29430
Helix	0,11718	-0,44716	0,47529	-0,06069
Disulphide	0,00000	0,00000	0,00000	0,00000
Acid_bonds	0,00000	0,00000	0,26497	0,16040
Exposure	0,95738	0,96742	0,50931	0,56254
Energy	-0,00161	-0,00328	0,00419	-0,00283
Cavities	-0,62560	-0,51837	-0,01029	-0,03692
Intercept	-2,44787	-1,41340	-1,44959	-1,37161

Table 5. Parameters for the trained model.

Parameters shown in this table are the adjusted λ for each partial score and the intercept described in equation (5) after using each of the groups of the database as testing group. As the scores of the modules, they are defined in arbitrary units. The result from substituting partial scores in this model can then be used in a logistic curve to calculate the probability of a mutation being stabilizing.

Proposal statistics

The ten structures with more mutations in the database have been analyzed without any execution error. Statistics obtained from the results and time spent running the program are summarized in table 6, only for single mutations. 24 double mutations were proposed, with only 3 with a probability higher than 10%, being the maximum 12,5%. None of this double mutations appeared in the database nor in Protherm.

To assess the effects of mutation proposal, proposed mutations are searched in the database and classified as positive or negative $\Delta\Delta G$. As few mutations are in the database, results can be inaccurate and all mutations have to be analyzed together instead of making a breakdown for each protein structure. The percentage of mutations that are positive is calculated for both the original database and the proposed mutations in it, in order to compare. As a measure of improvement, the ratio between percentages in proposed mutations and the original database is calculated. This results are shown in table 7.

PDB	1STN	1A2P	1RX4	1VQB	2CI2	1ARR	1WQ5	1QLP	1RN1	1FKJ	TOTAL
Execution time (s)	8,076	5,616	15,492	2,385	7,430	1,608	99,999	47,368	4,313	18,290	209,577
Number of mutations	39	36	54	5	19	11	119	145	24	24	476
Max r-score	29,80%	71,64%	51,36%	13,01%	49,68%	33,17%	19,05%	34,27%	29,66%	40,43%	71,64%
Mutations over 50%	0	2	1	0	0	0	0	0	0	0	3
Mutations over 25%	1	2	1	0	3	1	0	6	1	7	22
Mutations over 10%	5	24	5	2	9	1	10	104	20	20	200
Proposed by:											
Ancestral	26	24	36	0	14	0	67	91	15	11	284
Consensus	0	2	0	0	3	1	0	2	1	1	10
Helix	10	7	15	0	3	10	43	34	6	2	130
Disulphide	0	0	0	0	0	0	0	0	0	0	0
Acid bonds	0	2	0	0	0	0	1	0	0	2	5
Exposure	2	1	2	5	2	0	2	10	2	1	27

Table 6. Summary of the statistics obtained for the chosen structures in single mutations.

The table includes for each structure the time that the program was running for that structure (in minutes), the number of mutations proposed, the maximum probability for a mutation, as calculated by the scoring trained model, and a breakdown of the mutations according to their probability and to the module they have been proposed by. The total column contains the addition of all the structure, except for the maximum probability, which is the maximum of the values. The addition of the “proposed by:” values for all modules may be higher than the total number of mutations, as a mutation can be proposed by several modules.

	Positives	% positives	Ratio vs database
Database	323	18,07%	1,000
Proposed	16	32,65%	1,807
Proposed over 10%	10	58,82%	3,254
Proposed over 25%	2	100,00%	5,533

Table 7. Assessment of results of the program.

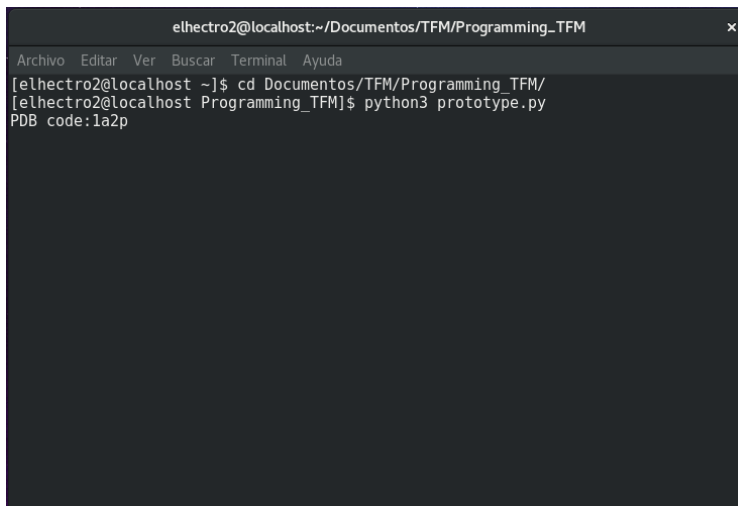
For each group, the number of positive measured values, the percentage of positives from the total measured values in the group and the ratio between that percentage and the percentage of the full original database is shown.

Program input and output

The program is started from the Fedora terminal. The only necessary input is the PDB code. An example of how to run the program and the necessary input is shown in figure 14.

Figure 14. Starting the program.

First, navigation to the folder with the program is performed (first line). Then, the program is started using Python 3.6 (second line). Then the program shows a message (“PDB code:”) and the user inputs the code of the protein to analyze. During the execution of the program, several messages appear on the terminal indicating the progress of the analysis.



```
elhectro2@localhost:~/Documentos/TFM/Programming_TFM
Archivo Editar Ver Buscar Terminal Ayuda
[elhectro2@localhost ~]$ cd Documentos/TFM/Programming_TFM/
[elhectro2@localhost Programming_TFM]$ python3 prototype.py
PDB code:1a2p
```

Currently, the output the program returns is a text file with all mutations, their probability of being stabilizing, the function each mutation has been proposed by and the partial scores for all fields. Also, a simplified html file is generated for easy interpretation of the results, as shown in figure 15.

The text file contains 11 different columns, separated by spaces. The first column describe the mutation as in the following example: “I109M_chain_A_PDB_1A2P”. In this case, I (isoleucine), the residue in position 109 of the chain A of structure 1A2P, has been mutated to M (methionine). The 2nd column is the probability of the mutation to be stabilizing, the 3rd is a list of numbers meaning which of the modules has proposed the mutation and the rest of columns are the scores for the evaluating functions of the module, shown in this order: ancestral, consensus, alpha-helix, disulphide, acid hydrogen bonds, exposure, energy and cavities. This order is also used for the numbers in the 3rd column, being 0 for the ancestral module and 5 for the exposure module (as energy and cavities module cannot propose mutations, they are not assigned any number).

The html file reduces the information from numerical values of the scores to qualitative information, being positive, negative or neutral. It also shows all the mutation description in only one column with spaces instead of underscores and uses a colour

code for a faster visual interpretation of the results, apart from a legend for the colour code and the module abbreviated names.

Single mutation results

Mutation	r-score	Anc	Cons	Hel	DiS	AcB	Exp	Ene	Cav
R16K chain A PDB 1ARR	33%	0	*	-	0	0	0	-	0
Q39A chain A PDB 1ARR	9%	0	-	*	0	0	0	-	0
F45G chain A PDB 1ARR	9%	0	-	*	0	0	0	0	-
V18A chain A PDB 1ARR	8%	0	-	*	0	0	0	-	-
V33A chain A PDB 1ARR	6%	0	-	*	0	0	0	0	0
E27A chain A PDB 1ARR	6%	0	-	*	0	0	0	0	0
E28A chain A PDB 1ARR	6%	0	-	*	0	0	0	0	0
S35A chain A PDB 1ARR	6%	0	-	*	0	0	0	*	0
D20A chain A PDB 1ARR	6%	0	-	*	0	0	0	0	0
Y38A chain A PDB 1ARR	6%	0	-	*	0	0	0	+	-
P15D chain A PDB 1ARR	5%	0	-	*	0	0	0	-	0

Double mutation results

Mutation1	Mutation2	r-score	Anc1	Anc2	Cons1	Cons2	Hel1	Hel2	DiS1	DiS2	AcB1	AcB2	Exp1	Exp2	Ene1	Ene2	Cav1	Cav2
A26C chain A PDB 1ARR	E36C chain A PDB 1ARR	6%	0	0	-	-	-	-	*	*	0	0	0	0	-	-	0	*

Color code legend

Code	Meaning
*	Positive score for this module
0	Neutral score for this module
-	Negative score for this module
*	Proposed by this module

Modules legend

Module	Full name
Anc	Reconstruction of the ancestral sequence
Cons	Prediction of the consensus sequence
Hel	Study of alpha-helix stability
DiS	Study of disulphide bonds
AcB	Study of exposed acidic hydrogen-bonded residues
Exp	Study of the exposure and polarity of the residues
Ene	Study of the minimum energy of the protein structure
Cav	Study of the volume of internal cavities in the protein

Figure 15. Full simplified results page in html format.

This is the result page displayed and saved by the program at the end of the analysis of a structure. It includes the proposal of single mutations and double mutations with the qualitative result for each module. The legends at the end of the page make this report self-explanatory.

Discussion

Alpha-helix scoring comparison

Comparison between the three datasets show a good correlation on the internal residues of the helix (table 2a). Fernández [29] and Pace [28] are the most similar datasets. However, Muñoz [27] is not significantly different. However, significant differences are found for N- and C-cap datasets (tables 2b and 2c, respectively). For N-cap, Serrano [31] and Doig [30] datasets show a good correlation with Muñoz dataset, but the correlation between them is not so good. For C-cap, all correlations with Doig dataset are poor, specially for the Serrano dataset.

This is probably a consequence of the different methods used for each dataset. Pace and Doig datasets are measured in model peptides, Fernández and Serrano are measured in model proteins and Muñoz use a model to refine empirical values measured before. As peptides are not complete proteins, some effects of the rest of the protein may be lost, which are especially important in the caps, as they are the ends of the helix and establish less interactions with the helix than the internal residues. Apparently, these effects are more important in the C-cap than in the N-cap, as the change in correlation is bigger. Some recent studies suggest that the effects on the C-cap can be mainly due to the change of mechanical tension of the protein structure with some residues as C-caps [32], a fact that may explain this results.

Also, as Muñoz dataset is obtained from a refined model, data for all types of residue in all positions is obtained, while some residues are not present in the other datasets. Because of this and for the simplicity of using only one dataset instead of a combination of different datasets, Muñoz dataset is preferred over the rest.

Program input and output

Starting the program is simple for a basic user of Fedora or Linux-based operative systems if all dependencies of the program are already installed, such as Python extensions or other programs (SCWRL, BetaVoid, etc.). The input method for using structures from the worldwide PDB database is user-friendly, as it allows for capital and lowercase letters in the PDB code. However, there is not an easy way for the user to analyze its own files yet, but it will be implemented soon. Also, an error introducing the PDB code results in the exit from the program with a generic error message. Some code in Python should also be implemented in order to check for misspelled PDB codes and allow the user for another chance to write it correctly.

The output html file offers a summary that easily allows to distinguish between positive, negative and neutral scores at first sight, even for colour-blind people, as different symbols are used apart from the colours. However, this value for the cavities module can be misleading, as a negative value (decrease of the volume of the cavities) is stabilizing. In future versions of the program, the definition of this score will be reversed for a more intuitive interpretation of the results.

The output text file (.txt) offers a more detailed summary, with quantitative values for the score in each module. However, some format compatibility problems have to be taken into account. The main problem is with line breaks. Windows Notepad does not detect line breaks correctly and shows the full document as one continuous line. Exporting the content of the document to a spreadsheet, like in Excel, solves this problem and allows for further analysis of the data.

Currently, all proposed mutations are being returned, but it could be useful for the user to have the possibility of using a filter so that only mutations with a probability over a concrete threshold are returned, or only up to a concrete number of results. This functionality is also being developed for future versions of the program.

Running the program. Troubleshooting.

In the first run, only 54% of the structures could be analyzed. This is a very low value, as current programs with the same aim can analyze all complete structures [10, 11] and some can even generate a less reliable prediction based only on the sequence, without the need of a structure [11]. This fact highlights the importance of making a robust program that can analyze almost all structures.

After the improvements, almost 89% of proteins could be analyzed. The causes of the failures in analysis were several (table 3). Half of the proteins that couldn't be analyzed had an incomplete backbone structure. This is not an easy problem to solve, as it can be incomplete in different ways and it should be completed differently depending on which part of the backbone it is lacking, but a fast way to detect this incompleteness should be developed to return the error message in the first seconds of execution of the program and not after half of the analysis is already performed.

The second cause of error in importance is repeated name. In most of cases, it comes from the output of CD-HIT, that makes two different sequences for a single input sequence. Further study of this problem should be made to find why this happens and

change the parameters of the program in order to avoid it. Other way of solving this could be a post CD-HIT analysis to check for sequences with the same name and, if found, select the sequence that is most similar to the rest of representative sequences, given the fact that usually one of the returned sequences is totally different from the rest.

PhyML error and BetaVoid segmentation fault only cause one error each, but they should also be handled. PhyML error happens only with proteins with a very big homologous family or a very small one, which in both cases creates an output for CD-HIT of one sequence, causing the error. For the cases of small family, this step of the analysis should be skipped. For the cases of a big family, either the number of retrieved sequences or the percentage of similarity in CD-HIT should be increased to generate more groups (even though these groups will be more similar between them than for other sequences in the latter case).

BetaVoid is a program that assesses the volume of all internal cavities of a protein, but a lot of errors are returned from it and it does not allow for the detection of the atoms in the limit of a cavity, so the prediction of cavity-filling mutation is not possible as originally thought. These two facts together suggest that BetaVoid is not the best option for this program and further software should be tested and integrated in the workflow for a better robustness and functionality.

Other conflictive point is Disulphide by Design, even though it is not the cause of any failure. The Windows-based program cannot analyze very big structures (such as 1AON) and requires pre-processing of the structure in a different operative system than the rest of the program, which is inconvenient for applications like a web server. On the other hand, the web-server Disulphide by Design 2.0 (DbD2) offers compatibility with all operative systems but makes the program dependent on this server. If this server is down, the program will not work. This server has been online for four years [DbD2] without major instabilities, so this should not be a problem in the near future. Because of this, a piece of code should be made to enable the program to communicate with the DbD2 server.

The modular structure of the program also offers another possibility to increase the robustness, with some costs in the reliability of the results. As all modules can work in a parallel way, with only minor dependences between them, if a module fails due to some of the already mentioned errors or new ones, the rest of modules can still work without

any problem. For this purpose, a piece of code should be developed to make the program continue with the other modules in case of an error and mark the module with the error as not analyzed in the final output, to indicate the possible partial loss of reliability of the results when not using that module. The development of the code and the assessment of the effects of not using each module remain as an open field for future work.

With all these modifications, especially with the last of them, the program would be able to analyze all structures and return a result. However, for the current work, a complete database of the evaluation results was needed for testing and training, so the last approach was useless, as it would leave some fields without evaluation.

Mutation database evaluation statistics

These statistics for the stabilizing subset of mutations in the database (table 4) serve as a first approach to detect which modules are more important for evaluating mutations. All disulphide scores are 0, which means that, in this database, no mutations involving disulphide bonds are present. This will make the logistic regression model ignore the disulphide scores, so more data with this type of mutations should be included in the database for a more complete model. Only one mutation involves acidic exposed hydrogen-bonded residues (0.1% of the 1030 evaluated stabilizing mutations), so the value for this score in the model can be inaccurate.

For the rest of the scores, all of them are enriched in the positive values, except for cavities and helix values. Scores are defined in a way that a positive value is expected in stabilizing mutations, so this general trend is what is expected. The definition of the cavities score is the only one that does not follow this rule. As this score is the difference between the volume of the cavities in the mutated and wild type structures and a smaller volume of cavities is usually correlated to stabilization, if the volume of cavities in the mutated protein is smaller (stabilizing mutation), the score for cavities will be negative. Thus, the enrichment in negative cavities score also happens as expected.

The only unexpected result is found for the helix module, where more positive values were expected, but it is slightly enriched in negative values. This could be due to many reasons. First of all is that residues in a helix can establish many different interactions with their side chains, so the effect of the mutation may depend more on the other

interactions than in the effect of the mutation on the stability of the helix. Also, it can be that some of the scores are not accurate. As three different positions (internal, N-cap and C-cap) exist, a breakdown of this values according to the position could be useful to detect in future work if some of these positions is less reliable on the results for the score value. If this was proven true, then three scores (one for each position) should be used instead of one for all residues in a helix, regardless their position.

The high percentage of positive values for the ancestral and consensus modules could be due to the use of reversed mutations. As this mutations use the wild type residue as the mutated residue, the probability of this residue being present in the consensus or the ancestral sequence is higher than for other residues. Because of that, special attention to these scores will be paid on further discussion.

Prediction accuracy

For single databases (figure 13a and b), the prediction accuracy is better than random predictions, but it is not especially good, with AUC values around 0.55 and 0.6. In the sNsC ROC curve, it is even worse than random for certain false positive rate values. In this graph, discretization of the true positive rate values can be observed, as some defined straight steps in the graph instead of a curve like in dNdC graph. This is probably due to the low number of stabilizing mutations in the testing group (sC), so testing with another group has been performed.

For duplicated databases (figure 13c and d), the results are significantly better, with an AUC value around 0.8 and 0.85, which is considered to be a very good value for ROC curves in binary prediction. Previous methods, such as the developed by Capriotti [11], reached a maximum AUC value of 0.76 for structure-based prediction methods and 0.73 for sequence-based methods. Other methods with quantitative predictions, such as CUPSAT [10] cannot be directly compared to binary prediction methods.

dNsC and sNdC graphs (figure 13e and f) are useful to check if the discretization effect in sNsC graph is caused as an effect of the low number of stabilizing mutations in the test group and how big that effect is. The effect is really significant, as the dN trained model AUC decreases to 0.58 with the sC test group and the sN trained model reaches an AUC of 0.83 with the dC test group, an AUC almost as big as for the dN trained model. This means that both models are almost equally good. However, the parameters can be different, especially in the consensus and ancestral functions, as mentioned

before. Discussion for training groups from now on will be focused on the New databases (N), as both their single and duplicated database generate better models after training than the Capriotti databases (C).

When comparing the parameters calculated for each model (table 5), no special biases can be observed in the dN trained model for the ancestral and consensus values. The ancestral lambda value for dN trained mode is close to the average of the sC and sN trained models lambda values, as if it was an intermediate value between both databases, thus not being as biased as the single databases' values. For the consensus values, the difference between the dN and sN values is insignificant and close to the average of sC and dC trained models.

When comparing the rest of values between sN and dN trained models, big changes can only be found in helix and energy values. The helix change can be due to the reasons discussed in the mutation database evaluation statistics section: other interactions more important than the stabilization of the helix or inaccurate scores for the mutations in different positions. This should also be studied by making a breakdown of the three types of position in a helix. The change in energy is small, but can be significant as lambda values are both small too. No clear explanation has been found to this fact, and it is not on the way suggested by the analysis of the statistics for stabilizing mutations in the database (table 4).

Having a look at the sign of the lambda values of the sN and dN trained models, all are positive except for cavities in both models and energy and helix in dN trained model. A value of 0 is obtained for disulphide, as no mutations of this type were in the database. As explained before for the scores, in all of them except for cavities, a positive value is expected to be stabilizing, so their lambda values should be positive too. In the case of cavities, as a negative value is expected to be stabilizing, its lambda value is expected to be negative, to account for this change of sign. Thus, all results are according to the expected hypothesis, except for the helix and energy values for dN, probably because of the aforementioned causes.

As a whole, both sN and dN trained models seem to be good and no bias for the ancestral and consensus values has been found. However, differences in the helix and energy values should be further researched to find their real cause and determine if a bias is generated in the duplication of the database with the reversed mutations. If it is, as the sC is a bad group for testing, the "leave-one-out" approach for testing can be

used, that consists on training the model with all the training group (sN) except for one element of the group (one mutation) and predicting the r-score of that element. This is repeated for all mutations, so that all r-scores needed for the ROC curve are obtained this way.

Comparing both databases (N and C), some significant changes are found. This means that not all evaluated elements are equally important in both databases, so it could be different in a third group of proteins, so probably a bigger group is required for reaching a better model.

Proposal statistics

In the first runs of the full program (table 6), it has proved a fast execution with the processing power of a standard laptop, with an average time of 20 minutes. However, the average is not the best measure of the time that the program takes for a full run, as the execution time depends on the length of the protein sequence, the size of its family and the number of proposed mutations, so a big variation between proteins can be found. In this set of 10 proteins, the fastest structure to be analyzed was 1ARR, with less than one minute and a half, and the slowest, 1WQ5, with over an hour and a half. This shows the wide range of execution times that can be found. Most of this effect is due to the phylogenetic programs, SCWRL and BetaVoid. As they are external programs, no changes can be made in them to make them faster, so this problem is only solvable with the use of faster programs. However, the execution time is not bad.

Using the modular structure of the program, parallel processing can be used to make the program faster in more complex computational environments. For example, in a computer with three different processing cores, one of the cores can be dedicated to the sequence analysis module, with the phylogenetic programs, other can be used for the calculation of cavities and steric clashes with SCWRL and BetaVoid and the third one can use the rest of modules. This way, the three operations are being performed in parallel instead of in a sequential manner, avoiding queuing on faster modules. However, the execution time would be limited by the slowest of the three cores.

Regarding which modules have proposed mutations, the ancestral function is clearly the most prolific one, followed by the helix module. The difference in the number of proposed mutations by the ancestral and by the consensus functions is very significant, even though both functions are based on sequence comparison. However, as the

ancestral approach goes back to the past of the proteins and uses more sequences, it is a more extreme approach, proposing more mutations than the consensus approach, but being worse at a predictive level, as it can be deduced from the data in table 5, where the lambda values for consensus are usually one order of magnitude over the ancestral lambda values, being scores for both modules in the same order of magnitude.

The disulphide module has proposed no single mutations, but it has proposed 24 double mutations. This lack of single mutations involving disulphides can be the cause of the lack of this kind of mutations in the original database. There is also a low number of mutations proposed by the analysis of exposed acidic hydrogen-bonded residue, which also correspond to 1 mutation in the original database. Searching in current literature for articles using these approaches can be an effective way to enlarge the mutation database, as ProTherm hasn't been updated since February 2013 and research in these fields is currently being conducted. This will allow for the calculation of more accurate lambda values for these fields.

Another important fact is that no structures have been left without any proposed mutation. However, 1VQB has only 5 proposed mutations. To increase the number of proposed mutations, efforts should be made in functions such as consensus, acid bonds or exposure to make the parameters less restrictive, in order to propose more mutations. Also, new modules can be implemented if they are developed in the future, like the analysis of the distribution of charges in a protein or the introduction of cavity-filling mutations, which would also increase the number of proposed mutations.

To serve as a comparison value, the average r-score for the original database, with both stabilizing and non-stabilizing mutations (but predominantly non-stabilizing) is around 7 %, reaching a maximum of 57%. For the proposed mutations, three are over 50% r-score values and almost half (42%) of the proposed mutations are over 10%, still over the average of the full original database. These values can be further increased introducing more empirically deduced rules for protein stability (see outlook section below)

From the 476 proposed mutations, only 49 were present in the database. None of the 49 proposed mutations in the database was over a 50 % r-score value, which would be really significant. This fact makes the analysis of the quality of the results of the program more difficult and less reliable.. Because of this, the percentage of mutations

with positive $\Delta\Delta G$ is used to measure the quality of the results instead of the percentage of stabilizing mutations.

Comparing the subset of proposed mutations with the full original database, a clear improvement is found, the percentage of mutations with positive $\Delta\Delta G$ is almost twice (1.8 times) the percentage in the full database. If the r-score is over 10%, this percentage is over 50%, being more than three times the database value. Only two proposed mutations with the probability over 25% are present in the database. Both of them have a positive $\Delta\Delta G$, one of them being stabilizing (higher than 1 kcal/mol) and the other one with a value of 1 kcal/mol, in the limit between neutral and stabilizing, but neutral by definition. However, due to the low number of samples, this results are not significant, but suggest an improvement in the proposal process versus random selection from the database.

Outlook

As a whole, this program is a new method to propose and evaluate mutations for the majority of protein structures using rational and empirical rules instead of only statistics or simulations with a better performance than other programs. However, optimization of the workflow and the parameters of each module is still necessary, as discussed before. Also, the creation of a larger database of mutations, especially with disulphide bonds, could be beneficial for the evaluating model.

Some new modules can be developed in the near future after this work, such a module for cavity-filling mutations or a module to study the distribution of charges in a protein in order to optimize it, as described by Estrada et al [61]. Also, some new approaches using benchmark mutation databases and Environment Specific Substitution Tables (ESSTs) [62], developed at the same time than this Master thesis, can be taken into account both for the use of ESSTs in a new module and for the databases, in order to enlarge the training mutation dataset.

Another point to address in future work is the existence of proteins with an unfolding equilibrium intermediate, known as three-state proteins. In this type of proteins, only a part of the protein unfolds at first to become the intermediate form, and then the intermediate fully unfolds. Because of this two step denaturing process, it is important to focus on the part of the protein that unfolds first to avoid the loss of biological

activity [63, 64]. This can be done using prediction programs such as ProteinLIPS [65] together with the probability obtained in this program to focus mainly on these areas.

In few words, the developed prototype of the program allows for the proposal of mutations with a high probability (59% in the studied set of proteins) of having a positive $\Delta\Delta G$ if using a 10% r-score threshold. Further work on the fields discussed before can increase this value to reach the 80 % accuracy described for some current evaluation methods [10, 11] that are, therefore, more accurate but can only offer predictions for a few user-determined mutations and, unlike this program, do not attempt to propose mutations on their own.

Conclusions

- Combination of rational simple rules based on structural and sequential properties of a protein allow for the successful proposal and prediction of stabilizing mutations.
- Combination of rational rules in a fast, user-friendly program allows to propose probable stabilizing mutations with a better performance than the already existing methods.
- It is possible to use the same rational-rule based approach for the evaluation of given mutations with better accuracy than the already existing methods.
- The mutation dataset for training can generate biases in the evaluating model. A bigger mutation dataset is required to ensure this model apply to most proteins.
- The results from this program can be conveyed to scientists outside of the computational structural biology field using a simple report template.
- Further work is required for the optimization of the program, the implementation of new functionalities and new mutation-proposing modules.

Conclusiones

- La combinación de reglas racionales simples basadas en las propiedades estructurales y de la secuencia de una proteína permiten la correcta propuesta y predicción de mutaciones estabilizantes.
- La combinación de reglas racionales en un programa rápido y fácil de usar permite proponer mutaciones probablemente estabilizantes con un mayor rendimiento que los métodos ya existentes.

- Es posible utilizar el mismo enfoque basado en reglas racionales para evaluar mutaciones concretas con mayor precisión que los métodos actuales.
- El conjunto de mutaciones para el entrenamiento puede generar sesgos en el modelo de evaluación. Un conjunto mayor es necesario para asegurar que el modelo sea aplicable a la mayoría de proteínas.
- Los resultados de este programa se pueden transmitir a científicos fuera del campo de la biología computacional estructural usando una plantilla simple de informe.
- Más trabajo es necesario para la optimización del programa, la implementación de nuevas funcionalidades y nuevos módulos de propuesta de mutaciones.

Bibliography

1. Lodish, H. *et al.* *Molecular Cell Biology*. 6th edition (W. H. Freeman, 2007).
2. Li, Y.-F. *et al.* Immunochemical techniques for multianalyte analysis of chemical residues in food and the environment: A review. *Trends Anal. Chem.* **88**, 25–40 (2017).
3. Kanegane, H. *et al.* Flow cytometry-based diagnosis of primary immunodeficiency diseases. *Allergol. Int.* **In Press**, (2017).
4. Saiki, R. K. *et al.* Primer-Directed Enzymatic Amplification of DNA with a Thermostable DNA Polymerase. *Science*. **239**, 487–491 (1988).
5. Littlechild, J. A. Improving the ‘tool box’ for robust industrial enzymes. *J. Ind. Microbiol. Biotechnol.* **44**, 711–721 (2017).
6. Madsen, C. *et al.* Echocardiographic and clinical findings in patients with Fabry disease during long-term enzyme replacement therapy: a nationwide Danish cohort study. *Scand. Cardiovasc. J.* **51**, 207–216 (2017).
7. Muñoz, V. & Serrano, L. Elucidating the folding problem of helical peptides using empirical parameters. III. Temperature and pH dependence. *J. Mol. Biol.* **245**, 297–308 (1995).
8. Dombkowski, A. A., Sultana, K. Z. & Craig, D. B. Protein disulfide engineering. *FEBS Lett.* **588**, 206–212 (2014).
9. Zore, O. V *et al.* Nanoarmoring: Strategies for preparation of multi-catalytic enzyme polymer conjugates and enhancement of high temperature biocatalysis. *RSC Adv.* **7**, 29563–29574 (2017).

10. Parthiban, V., Gromiha, M. M. & Schomburg, D. CUPSAT: Prediction of protein stability upon point mutations. *Nucleic Acids Res.* **34**, 239–242 (2006).
11. Capriotti, E., Fariselli, P., Rossi, I. & Casadio, R. A three-state prediction of single point mutations on protein stability changes. *BMC Bioinformatics* **9**, S6 (2008).
12. Ng, P. C. & Henikoff, S. Predicting deleterious amino acid substitutions. *Genome Res.* **11**, 863–874 (2001).
13. Choi, Y. & Chan, A. P. PROVEAN web server: a tool to predict the functional effect of amino acid substitutions and indels. *Bioinformatics* **31**, 2745–2747 (2015).
14. Adzhubei, I. A. *et al.* A method and server for predicting damaging missense mutations. *Nat. Methods* **7**, 248–249 (2010).
15. Doshi, U. & Hamelberg, D. Towards fast, rigorous and efficient conformational sampling of biomolecules: Advances in accelerated molecular dynamics. *Biochim. Biophys. Acta - Gen. Subj.* **1850**, 878–888 (2015).
16. Merkl, R. & Sterner, R. Ancestral protein reconstruction: Techniques and applications. *Biol. Chem.* **397**, 1–21 (2016).
17. McDonald, I. K. & Thornton, J. M. Satisfying Hydrogen Bonding Potential in Proteins. *J. Mol. Biol.* **238**, 777–793 (1994).
18. Steiner, T. The hydrogen bond in the solid state. *Angew. Chem. Int. Ed.* **41**, 49–76 (2002).
19. Irun, M. P., Maldonado, S. & Sancho, J. Stabilization of apoflavodoxin by replacing hydrogen-bonded charged Asp or Glu residues by the neutral isosteric Asn or Gln. *Protein Eng* **14**, 173–181 (2001).
20. Dombkowski, A. A. Disulfide by Design: A Computational Method for the Rational Design of Disulfide Bonds in Proteins. *Bioinformatics* **19**, (2003).
21. Craig, D. B. & Dombkowski, A. A. Disulfide by Design 2.0: a web-based tool for disulfide engineering in proteins. *BMC Bioinformatics* **14**, 346 (2013).
22. Touw, W. G. *et al.* A series of PDB-related databanks for everyday needs. *Nucleic Acids Res.* **43**, 364–368 (2015).
23. Yin, X. *et al.* Contribution of Disulfide Bridges to the Thermostability of a Type A Feruloyl Esterase from *Aspergillus usamii*. *PLoS One* **10**, (2015).
24. Richardson, J. S. The Anatomy and Taxonomy of Protein Structure. *Adv. Protein Chem.* **34**, 167–339 (1981).
25. Doig, A. J. Recent advances in helix – coil theory. *Biophys. Chem.* **102**, 281–293 (2002).

26. Lacroix, E., Viguera, A. R. & Serrano, L. Elucidating the folding problem of α -helices: local motifs, long-range electrostatics, ionic-strength dependence and prediction of NMR parameters. *J. Mol. Biol.* **284**, 173–191 (1998).
27. Muñoz, V. & Serrano, L. Elucidating the Folding Problem of Helical Peptides Using Empirical Parameters. *Nat. Struct. Biol.* **1**, 399–409 (1994).
28. Pace, C. N. & Scholtz, J. M. A helix propensity scale based on experimental studies of peptides and proteins. *Biophys. J.* **75**, 422–427 (1998).
29. Fernández-Recio, J. & Sancho, J. Intrahelical side chain interactions in alpha-helices: Poor correlation between energetics and frequency. *FEBS Lett.* **429**, 99–103 (1998).
30. Doig, A. J. & Baldwin, R. L. N- and C- capping preferences for all 20 amino acids in alpha-helical peptides. *Protein Sci.* **4**, 1325–1336 (1995).
31. Serrano, L., Sancho, J., Hirshberg, M. & Fersht, A. R. Alpha-Helix stability in proteins. I. Empirical correlations concerning substitution of side-chains at the N and C-caps and the replacement of alanine by glycine or serine at solvent-exposed surfaces. *J. Mol. Biol.* **227**, 544–559 (1992).
32. Bang, D. *et al.* Dissecting the energetics of protein alpha-helix C-cap termination through chemical protein synthesis. *Nat. Chem. Biol.* **2**, 139–143 (2006).
33. Ayuso-Tejedor, S., Abián, O. & Sancho, J. Underexposed polar residues and protein stabilization. *Protein Eng. Des. Sel.* **24**, 171–177 (2011).
34. Strub, C. *et al.* Mutation of exposed hydrophobic amino acids to arginine to increase protein stability. *BMC Biochem.* **5**, 9 (2004).
35. Estrada, J., Bernadó, P., Blackledge, M. & Sancho, J. ProtSA: a web application for calculating sequence specific protein solvent accessibilities in the unfolded ensemble. *BMC Bioinformatics* **10**, 104 (2009).
36. Kaplan, W. & Littlejohn, T. G. Swiss-PDB Viewer (Deep View). *Brief. Bioinform.* **2**, 195–197 (2001).
37. Humphrey, W., Dalke, A. & Schulten, K. VMD - Visual Molecular Dynamics. *J. Mol. Graph.* **14**, 33–38 (1996).
38. Krivov, G. G., Shapovalov, M. V. & Dunbrack, R. L. Improved prediction of protein side-chain conformations with SCWRL4. *Proteins Struct. Funct. Bioinforma.* **77**, 778–795 (2009).
39. Krone, M. *et al.* Visual Analysis of Biomolecular Cavities: State of the Art. *Comput. Graph. Forum* **35**, 527–551 (2016).

40. Bueno, M., Campos, L. a, Estrada, J. & Sancho, J. Energetics of aliphatic deletions in protein cores. *Protein Sci.* **15**, 1858–72 (2006).
41. Bueno, M., Cremades, N., Neira, J. L. & Sancho, J. Filling Small, Empty Protein Cavities: Structural and Energetic Consequences. *J. Mol. Biol.* **358**, 701–712 (2006).
42. Kim, J.-K. *et al.* BetaVoid: Molecular voids via beta-complexes and Voronoi diagrams. *Proteins Struct. Funct. Bioinforma.* **82**, 1829–1849 (2014).
43. McInerney, F. A. & Wing, S. L. The Paleocene-Eocene Thermal Maximum: A Perturbation of Carbon Cycle, Climate, and Biosphere with Implications for the Future. *Annu. Rev. Earth Planet. Sci.* **39**, 489–516 (2011).
44. Koonin, E. V. Orthologs, Paralogs, and Evolutionary Genomics. *Annu. Rev. Genet.* **39**, 309–338 (2005).
45. Edgar, R. C. MUSCLE: a multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics* **5**, 113 (2004).
46. Larkin, M. A. *et al.* Clustal W and Clustal X version 2.0. *Bioinformatics* **23**, 2947–2948 (2007).
47. Guindon, S. *et al.* New Algorithms and Methods to Estimate Maximum-Likelihood Phylogenies: Assessing the Performance of PhyML 3.0. *Syst. Biol.* **59**, 307–321 (2010).
48. Li, W. & Godzik, A. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics* **22**, 1658–1659 (2006).
49. Boussau, B., Blanquart, S., Necsulea, A., Lartillot, N. & Gouy, M. Parallel adaptations to high temperatures in the Archaean eon. *Nature* **456**, 942–945 (2008).
50. Yang, Z. PAML: a program package for phylogenetic analysis by maximum likelihood. *Bioinformatics* **13**, 555–556 (1997).
51. Cock, P. J. A. *et al.* Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics* **25**, 1422–1423 (2009).
52. Madden, T. L., Tatusov, R. L. & Zhang, J. Applications of network BLAST server. *Methods Enzymol.* **266**, 131–141 (1996).
53. Jones, D. T., Taylor, W. R. & Thornton, J. M. The rapid generation of mutation data matrices from protein sequences. *Comput Appl Biosci* **8**, 275–282 (1992).
54. Kabsch, W. & Sander, C. Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers* **22**, 2577–2637 (1983).

55. Edelsbrunner, H. & Koehl, P. The weighted-volume derivative of a space-filling diagram. *Proc. Natl. Acad. Sci.* **100**, 2203–2208 (2003).
56. Leader, D. P. & Milner-White, E. J. The structure of the ends of α -helices in globular proteins: Effect of additional hydrogen bonds and implications for helix formation. *Proteins Struct. Funct. Bioinforma.* **79**, 1010–1019 (2011).
57. Kumar, M. D. S. *et al.* ProTherm and ProNIT: thermodynamic databases for proteins and protein–nucleic acid interactions. *Nucleic Acids Res.* **34**, 204–206 (2006).
58. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V. & Thirion, B. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **12**, 2825–2830 (2011).
59. Wickham, H. *ggplot2: Elegant Graphics for Data Analysis*. (Springer-Verlag New York, 2009).
60. Lobo, J. M., Jiménez-valverde, A. & Real, R. AUC: A misleading measure of the performance of predictive distribution models. *Glob. Ecol. Biogeogr.* **17**, 145–151 (2008).
61. Estrada, J., Echenique, P. & Sancho, J. Predicting stabilizing mutations in proteins using Poisson–Boltzmann based models: study of unfolded state ensemble models and development of a successful binary classifier based on residue interaction energies. *Phys. Chem. Chem. Phys.* **17**, 31044–31054 (2015).
62. Pandurangan, A. P., Ochoa-Montaña, B., Ascher, D. B. & Blundell, T. L. SDM: a server for predicting effects of mutations on protein stability. *Nucleic Acids Res.* **45**, 229–235 (2017).
63. Sancho, J. *et al.* The ‘relevant’ stability of proteins with equilibrium intermediates. *ScientificWorldJournal.* **2**, 1209–15 (2002).
64. Lamazares, E., Clemente, I., Bueno, M., Velázquez-Campoy, A. & Sancho, J. Rational stabilization of complex proteins: a divide and combine approach. *Sci. Rep.* **5**, 9129 (2015).
65. Angarica, V. E. & Sancho, J. Protein Dynamics Governed by Interfaces of High Polarity and Low Packing Density. *PLoS One* **7**, (2012).