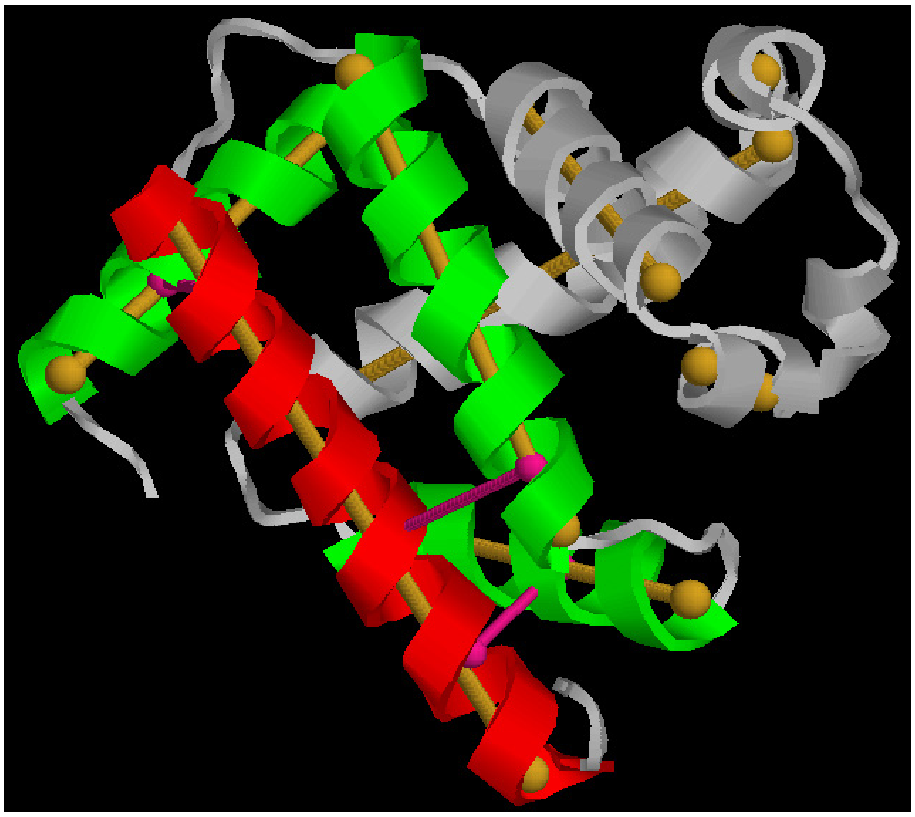


HelixFinder

A program for identifying and classifying protein α-helix pairs

Master of Science Thesis in Computer Science

MARKUS KOLLIND

Chalmers University of Technology
Department of Computer Science and Engineering
Gothenburg, Sweden, June 2015

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.

The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

HelixFinder
A program for identifying and classifying protein α-helix pairs

MARKUS KOLLIND

© MARKUS KOLLIND, June 2015

Examiner: GRAHAM KEMP

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover:
An image of the protein myoglobin (PDB id 1MBN) with helices and axes detected by the HelixFinder program. The green helices are detected as being close to the red one. The yellow lines are the calculated axes of the helices and the purple lines represent the closes distance between the helices.

Department of Computer Science and Engineering
Göteborg, Sweden, June 2015

Abstract

To gain understanding of proteins and increase knowledge of how they interact with and impact each other, studying larger sets of data on proteins is important. Being able to easily aggregate such data is essential for this type of analysis.

This work is investigating how to detect some specific sections of protein secondary structures, α-helices, and to, within a given protein, identify pairs of such helices that are close to each other. The identified pairs are categorized in four different categories, depending on how the distance is detected.

The outcome is a program, HelixFinder, that detects such helix pairs and gathers information about them. The program uses an implementation of the DSSP algorithm to identify the structures, and a rotational algorithm together with a number of distance tests to identify α-helix pairs and output data relevant for further analysis.

Acknowledgments

I would like to thank my supervisor Graham Kemp for steering me in the appropriate direction when it comes to what direction to take, both when it comes to the program as well as for the content of the report. I would also like to thank my mother for the support and review feedback, but especially I would like to thank my wife for the support and the encouragement to finalize my thesis work and to get my exam.

In the developed program the GNU Scientific Library (GSL) is used for some calculations. All illustrations of proteins are created using data files from the Protein Data bank provided by the Research Collaboratory for Structural Bioinformatics (RCSB), and the protein visualization software RasMol. In addition, the report is written in LaTeXusing a report template created by Jonas Einarsson.

Markus Kollind, Gothenburg, June 2015

Contents

Figures

1

Introduction

Proteins are vital for all life as we know it, and are part of every living creature. As macromolecules they are unique in that they are involved in every biological reaction [1]. The function varies depending on the type of protein, but proteins are active in almost all areas of life. Hormones, enzymes and antibodies are examples of groups of proteins, and keratin and gluten are examples of some of the well known proteins.

Proteins have an important role to fill in our lives as humans, but proteins losing their function or changing their behavior can also cause reduced functionality in life critical areas, or result in diseases like Alzheimer's, Parkinson's and Huntington's [2]. This means that proteins are important for our well being, as well as being the cause of some of our deceases.

To understand how living creates function is important for understanding how to improve life and prevent deceases, and a vital step in this is to understand how proteins work. This is not only about understanding how they function, but there is also a possibility to understand and find new, preventive methods or cures for diseases. This can be done both by understanding why proteins sometimes malfunction as well as trying to identify or even construct special customized proteins, like specially constructed antibodies, for certain diseases.

How a protein interacts depends on the form, or the structure, of the protein [3]. Experimentally determining the structures is cumbersome, and for some groups of proteins it is especially complex [4]. To learn more about proteins, and especially to be able to construct new proteins, more statistical information about how proteins are structured is needed.

1.1 Looking at α-helices

Several different areas can be investigated to gather statistics about protein structures, but one such area is to look at how the so called α-helices within proteins interact with each other as for instance done by Zhang at al. [5]. α-helices are special sections of protein structures, with a clearly defined formation and is one of the most common sub-section of proteins. For many proteins, interaction between such α-helices are critical for the structure of the protein [6]. And for many proteins they provide stability and function [7].

Several methods exists for predicting the form of proteins and several of these relies on knowledge on existing α-helices and statistics about helices interacting. Fleishman et al. have constructed a scoring function aimed to differentiate between helix pairs in membrane proteins that would closely interact from helix pairs that would not [8]. In this function, real data for helix pairs is used as a confirmation space for the scoring function.

Another method for trying to predict membrane protein structures is the *RosettaMembrane* tool, developed in increments by different teams. The increment done by Barth et al. uses a library of known protein structures with α-helix pairs, where local sequence matching is used to provide possible interaction points [9]. To allow for such tools to evolve, as well as for development of new tools, statistical gathering of known structures is important as a standpoint for the prediction tools to base the predictions on.

in general, statistical data for protein interaction could be used as an important source for data when constructing scoring functions for protein predictions, as well as for protein-protein docking predictions in relation to interaction of α-helices.

Although the interaction of α-helices is important for the structure of proteins, it has not been possible to define rules for how they impact the final structure [7]. This means that continued investigation of relations between such helices is needed, and to gain a better understanding of the process statistical analysis of larger sets of data would be one way to gain a better understanding.

Such analysis tools are available, with different approaches to what to evaluate as well as how interaction is defined. Burba et al. developed a web service, HIT, where analysis of α-helix interaction can be performed [7]. The method uses two approaches to evaluate if the helices interact, either by looking at the partitioning of space between helices by usage of the Voronoi method, or by measuring distance between atoms and if enough atom pairs are found fulfilling the closeness definition, the helices are defined as interacting.

Another tool, TMPad, for analyzing α-helix pairs is developed by Lo et al. [6]. They state that ”helix packing [...] is not governed by a single factor alone”, and to understand how helices are packing together in a protein one need to investigate and evaluate a number of different factors [6]. In their work the method for determining if helices are interacting is by looking at the distance between atoms of the helices, in a number of different criteria.

1.2 Aim

This thesis main focus is methods in the area of protein analysis. The aim is to investigate methods needed for detecting α-helix pairs in proteins and also for allowing to do analysis of the behavior of patterns in such interactions. In comparison to the work of Burba et al. [7] and Lo et al. [6], this method for identifying helix interaction is focusing on the whole helix, rather than looking at distance between single atoms within the helices.

The closeness detected is also categorized depending on how the distance is measured. The categories are based on the relation between the helices and their position in relation to each other.

The result is a compilation of methods that can be used to find information on distance and angles between such helix pairs, as well as other characteristics for the identified pairs. These methods are then evaluated by the implementation of a program, called HelixFinder, that is using these methods.

The output of the program can be used for statistical analysis of groups of proteins, when it comes to α-helix pairs. The generated data could also be used for scoring functions in prediction methods.

1.3 Thesis outline

To be able to understand the work, as well as the methods presented, the biology behind proteins will be presented in chapter 2. Also, to set the methods in context as well as explaining the choices made, a review of different computational methods for analyzing proteins and their form will be presented in chapter 3.

The program will then be presented, with a focus on the major parts of the software, in chapter 4. This will be followed by an analysis of the result from the program as well as suggestions for potential improvements, in chapter 5.

2

Protein structure

To understand this work and the reasoning behind the program developed, there is a need to understand the concepts of proteins and therefore the biology behind. In addition, understanding how prediction of protein functionality is done provides some concepts needed to understand how the results fits into the goal of protein prediction.

2.1 Protein building blocks

The central dogma in molecular is that DNA is transcribed to mRNA, and mRNA is translated to protein. Proteins consist of chains of specific combinations of amino acids, where the same protein have the same sequence no matter where it occurs. There are a number of different amino acids, but all proteins are constructed from a set of 20 amino acids[1], each with different chemical properties [3].

Each amino acid consists of a carbon atom(called α-carbon and written C^α) bonded with an amino group (H_2N), a carboxyl group (COOH), a hydrogen atom (H) and a side-chain group (usually called R). For the different amino acids, it is the side chain that varies and it can vary from a single hydrogen atom (H) to more complex structures (as seen in figure 2.1).

The amino acids have different properties, depending on the structure of the side chain; they can be differently charged, or hydrophobic or they can be polar. This results in different interactions and behavior depending on how they are contained in the proteins, and help diversify the range of proteins that exists. Some do, for instance, have ionizable side chains, making it possible to donate or accept protons to facilitate reactions, which is an important trait for enzymes [10].

[1]There are a few exceptions where there are proteins containing other amino acids [1]. This can, however, be disregarded for this work since the focus on this work is on the C^α) in the backbones, and the amino acid differences are impacting the side chains.

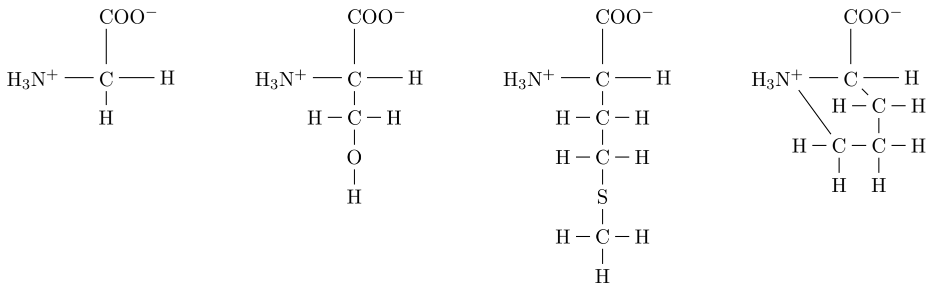


Figure 2.1: Amino acids vary in their side chains, where the size of the side chain can differ from a single atom to a more complex structure. Here, from left to right, Glycine, Serine, Methionine and Proline are shown in their predominant ionic forms as examples of amino acids, with all atoms of the side chains explicitly drawn to clearly show the difference in size.
Proline is a special case among the amino acids used in proteins, in that the side chain is also bonded to the main chain nitrogen atom.

2.2 Levels of protein structure

The amino acids of a protein are bonded together into chains, where the ordered sequence of the amino acids is called the *primary structure* of the protein. The chain is then formed into some common sub-structures shared between many proteins, that are called *secondary structure.* But the whole protein has a certain three dimensional form, its *conformation*, enabling its function; this is called the *tertiary structure.*

In some cases several amino acid chains come together to form a single protein; such a structure is called a *quaternary structure.*

Most proteins are in the size of 50 to 2000 amino acids, although there are proteins decoded with as many as 35 000 [11]. With the availability of 20 amino acids a protein with 100 residues would have 10^{130} possible combinations, which is much more than there are atoms in the universe[2]. Only very few of these structures become real proteins, and by natural selection, over time, only the best proteins have survived [3].

2.2.1 Primary structure

Amino acids are linked together between the amino and the carboxyl group by a stable bond with double bond characteristics [1], called a peptide bond. A number of amino acids bonded together like this is called a *polypeptide chain*, and each amino acid in such a chain is called a *residue* [10]. The repetitive, part of the chain, is called the *backbone* of the protein (as seen in figure 2.2).

Because of the double bond characters of the peptide bonds, the backbone is fixed in one of two ways in the bond connection: *trans* or *cis.* The trans form means that the α-carbons on the side of the bond are on opposite sides, while for the cis form they are

[2]The estimated mass of the universe is below 10^{53} kg [12], and with the assumption that all atoms are hydrogen atoms the total number of atoms in the universe would be approximately 10^{80}.

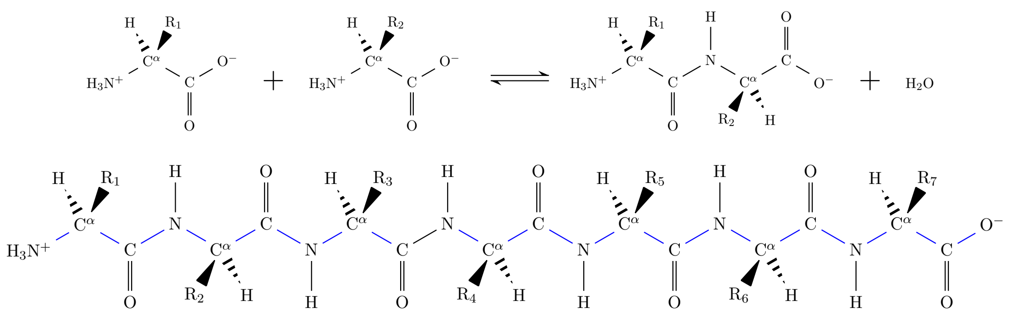


Figure 2.2: Amino acids link to each other by a peptide bond, resulting in chains of amino acids. The repeating main part of the chain is called the backbone, and is colored blue in the figure.

on the same side. Because of steric clashes between the R groups for the cis form, the trans form is dominating and almost all peptide bonds in proteins are actually of that form [10]. The ratio between cis and trans is 1:1000 for most peptide bonds[3] [1].

While the nature of the peptide bond means that there is no flexibility around that bond, the other bonds in the backbone can rotate. Although the possible positions are limited, and only 25% of all possible angles are actually available in reality in natural proteins [1], this does alllow the chain to change in form and shape, and thus result in the 3 dimensional structure of the protein.

Each polypeptide chain has a predetermined 3 dimensional structure it will retain, after it has been assembled. This process of going from an irregular extended structure to a functional structure is called *folding*.

2.2.2 Secondary structure

Looking at the 3 dimensional structure of a protein, it is possible to identify a number of structural sections that are reoccurring in proteins throughout. Those common structures are called secondary structures, and the most common ones are *α-helices*, *β-sheets* and *turns*.

When a chain folds into its final structure, it strives for a structure with low energy [3]. What holds it together varies between a number of different factors, but for the secondary structures the main force is the hydrogen bond, occurring between the NH and the CO group of the main chain.

α-helices are the most common structure found in proteins [1], and are the result of several hydrogen bonds between some consecutive residues and the fourth residue following each residue in the same chain. This means that all NH and CO groups are bonded by hydrogen bonds in the structure.

Some amino acids will not accommodate perfectly into an α-helix, because of steric clashes, and the amino acid proline lacks an NH group, as can be seen in figure 2.1, and

[3]The main exception here is proline, where a ratio is 1:4 when a residue is followed by proline.

therefore does disrupt a helix [10]. When it comes to peptide bonds that adopt the cis form, such a bond would break the helix.

A β-sheet is created by multiple straight polypeptide chain sections, so called β-strands, that are linked together by hydrogen bonds. One β-sheet can consist of a pair of β-strands or more than 10 strands [1].

A turn is simply a small section of the chain that makes it change direction. There are different types of turns, but the basic behavior of changing the direction is the same. The main difference is how many residues there are between the hydrogen bond resulting in the turn. The most common one is the β-turn that has a hydrogen bond between a residue and the following fourth one[4] [1].

Other structures exist as well, like 3_{10} helix and π helix. But those are quite rare in comparison [1]. In figure 2.3 a small protein with β-strands and an α-helix is shown.

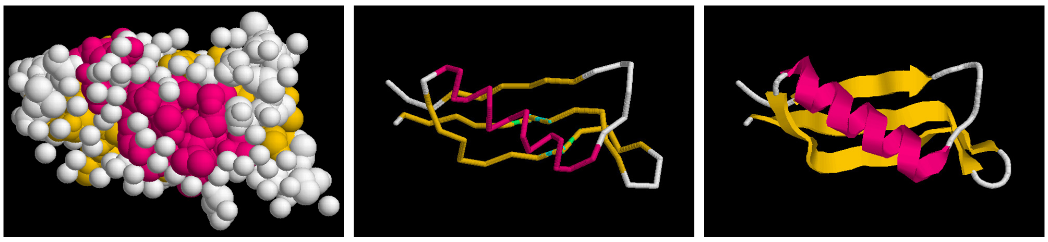


Figure 2.3: An example of a small protein with both β strands (orange) and an α-helix (pink). On the left the protein is shown with all non-hydrogen atoms shown, in the middle only the backbone is visualized, while on the right a cartoon representation of the structures is seen, and is used to better illustrate the position of the structures. The coloring is the same for the three illustrations and colors the α-helix with pink and the β-sheet with orange. The protein is the amyloid precursor protein of Alzheimer's, with PDB id 2FMA.

2.2.3 Tertiary structure

The final 3D form of the protein, the tertiary structure, is specifically defined by the primary sequence [10]. The creation of the final structure can come from disulfide bridges between side chains containing sulfur, it can be a result of hydrophobic effects, it can be van der Waals interactions or a number of other causes [1]. The folding of the protein reflects the summary of all the different forces coming from the different binding.

2.3 Structure determination

As proteins are such an important contributor to so many biological interactions, and the structure of the protein is what determines what the protein can contribute with, identifying structures for proteins is vital for many reasons. Two of the main reasons

[4]The definition is the same as for α-helices, but the difference is that a β-turn is only between two residues while an α-helix includes several residues. Slightly simplified, an α-helix could be described as several consecutive β-turns.

are, firstly, for understanding how existing proteins are formed and therefore understand how they function, and, secondly, for being able to design new proteins with a certain function, based on the structure.

When it comes to understanding the structure of existing proteins, the normal way of identifying the structure is to do it experimentally. This can be done in a few different ways, where the absolutely dominating method is through X-ray crystallography [13]. Although there are some different methods, they are all complex and time consuming [14]. X-ray crystallography does for instance require the protein to be grown into a crystalline form that is X-rayed to generate a diffraction pattern (because of the reflections of the crystals), and by applying Fourier analysis to the result, the structure can be calculated [2].

If one looks at the current development, one can see that currently the structure of some 100 000 proteins are known while the known sequences of proteins are more than 20 000 000. As if this was not enough, the number of known sequences is increasing at a much higher rate than known structures [3].

This means that just identifying existing proteins experimentally will be a major challenge and therefore algorithm based methods allowing to calculate the structure is important to enable future understand of how proteins are structured, to be able to evolve in the area of protein research [15].

Another reason for wanting to be able to computationally predict the structure of a protein, is for creation of completely new proteins. For instance, to be able to prevent diseases, constructing customized antibodies is a possibility. But to understand the behavior of the new protein, it is important to understand the structure of it. This can either be done by constructing the protein and experimentally analyze the result, or by having a good approach for how to predict the structure from the sequence.

2.4 Structure prediction

There are major challenges in the field of protein prediction. If one should try to calculate the number of possible ways a protein can fold, one can see that brute force calculation is simply not possible. Even if we restrict the possibilities of the backbone of a protein and assumes that only a few angles are possible for the rotational bonds, a protein of 100 residues would still have between 10^{30} and 10^{47} conformations to be considered [16]. Such calculations would be impossible to carry out for a normal computer. But likely a protein cannot fold into every shape imaginable, and there is most likely a very limited number of different folds to choose from, estimated to a range of some thousand [17].

There are a number of different method groups for predicting the structure of a protein, of which two are *template based modeling* and *de novo prediction*. None of them are currently at a level where they can be used to actually determine the structure of a protein, but the evolution is progressing and they are constantly improving [18].

Many of the methods uses, in one way or another, statistically generated data for a larger set of existing proteins, or uses parts of known proteins to match the unknown proteins with. In some of these, data is used for comparison, whiles in others interactions between structures, like α-helices, or distance between structures is used.

The fact that proteins with similar sequences have a tendency to adopt similar structures is the basis for *comparative modeling*, and this can then be used to identify template proteins as a basis for comparison [14]. *Threading* is an example of such a template based modeling method and it uses the assumption that there is a limited number of folds possible for a protein. Based on this, a database of experimentally determined folds is used, from where the sequence of a protein with unknown structure is matched with sequences from the database with known structures. In this way, the protein with unknown structure mimics the structure of the already known protein structures [16].

De novo modeling, on the other hand, tries to predict the structure for proteins by using the sequence information and without having similar proteins to compare with. Often *de novo* modeling still uses statistical information about known structures as for instance a basis for understanding interaction between different secondary structures. A number of different methods exists within this group, where some use fragments of folds. The method RosettaMembrane is an example of a method that uses distance constraints by taking helix-helix interactions into consideration [15].

3

Methods

To be able to detect $\alpha-$helices in proteins, there needs to be information about the 3 dimensional structure of the proteins that should be analyzed. Throughout the years, a standard for how to document the 3 dimensional structure of a protein has been developed, and such protein structures are stored in the Protein Data Bank as PDB files [19].

Such data files are what needs to be used as input for being able to detect α-helices and for analyzing if the helices are close to each other. When it comes to what defines close α-helices, in this project it is determined by measuring the distance between the closest points on the axes of the α-helices, for each pair of α-helices. This needs to be done in two steps:

1. Detecting α-helices, based on the data in the given PDB file
2. Identify if there are any α-helix pairs that are close to each other

In this thesis work the focus is on the methods chosen for detecting α-helices, and finding pairs of such α-helices that are close to each other. In the chapter 4, the program itself is described.

3.1 Finding helices

Finding secondary structures of a protein, based on the coordinate data provided in a PDB file, is an important step in understanding the structure and behavior of a protein. There is no objective truth as to when a secondary structure starts and ends, meaning that different approaches might have different advantages [20].

Also, different approaches might be better than others at certain types of secondary structures, or have other traits that makes them valuable from other aspects. In this work, however, only α-helices are of interest to be detected.

There are a number of different aspects that could be looked at for detecting secondary structure, but the majority of the methods either look at the coordinate data in the PDB file for calculating hydrogen bonds, or look at the coordinates of the α-carbon from the backbone of the proteins polypeptide chain [21].

The first method for calculating secondary structure was presented 1977 by Michael Levitt and Jonathan Greer; a method using the location of the C^α and angles between residues and looking at a window with four residues at a time [22]. This was followed by a number of methods for identifying secondary structure, where for instance one method looked at the coordinates of the C^α and based on those calculated the angles between residues in the backbone [23]. Another example of such a method is DEFINE, that used a matrix to identify distances between all C^α and if the distances matched to a mask, it would be detected as an α-helix [24]. As late as 2005 a method, called KAKSI, was developed trying to handle some of the drawbacks of the existing methods. It used the coordinates of the C^α in combination with the angles in the backbone, trying to improve the analysis and detection of secondary structures [20].

The most used methods [21] are based on calculations of the hydrogen bonds within the protein structure: DSSP [25] and STRIDE [26]. Of these, the majority of the secondary structures defined in PDB files are defined using DSSP [20]. The calculations are based on the coordinates of a number of atoms and varies between STRIDE and DSSP. STRIDE is developed to make automatic prediction fit better with the known assignments done manually by the crystallographers [26]. The main difference between the two methods lies in how the hydrogen bonds are defined, and that STRIDE also uses the angles between atoms on the backbone to improve calculations.

Several studies have been done, comparing different methods for identifying secondary structure. Cuff and Barton did one such study where they compared DSSP, STRIDE and DEFINE [27]. Their conclusion is that the result of STRIDE and DSSP are similar and have quite a high percentage of agreement, while DEFINE deviates from the other two methods [27]. In addition, DSSP does not find quite as long helices as the other methods, but finds more helices in total than the other two.

In another study, by Zhang and Sagui, compared STRIDE, DSSP and KAKSI to see what the advantages and disadvantages of the different methods are [21]. The conclusion is that STRIDE and DSSP defines more secondary structure than KAKSI, and DSSP is the method to achieve the highest results when distinguishing between different types of helices [21].

Looking at a number of different protein visualization tools, the majority of them uses DSSP or variants of DSSP. RasMol, Jmol and Chimera all uses DSSP, while PyMOL uses a variant similar to DSSP. Of the tools looked at, VDM uses STRIDE and is the only one using something else than DSSP.

This leads to the conclusion that the most suited method for usage in this work, to identify α-helixes, is DSSP. That is, since the limitations of DSSP are on other structures than α-helices and that is also the most used method for identifying secondary structure.

3.1.1 Finding $\alpha-$helices with DSSP

DSSP is a method using hydrogen bonds to determine secondary structures of the protein. It can determine several different structures, but here only the detection of α-helices will be of interest.

To calculate the hydrogen bonds, a formula is used for the electrostatic interaction energy between residue i and j:

$$E_{i,j} = q_1 q_2 \left(\frac{1}{r(O_i N_j)} + \frac{1}{r(C_i H_j)} - \frac{1}{r(O_i H_j)} - \frac{1}{r(C_i N_j)} \right) * f \tag{3.1}$$

In this equation $q_1 = 0.42e$ and $q_2 = 0.20e$, where e is the unit electron charge. Also, $f = 332$ is the dimensional factor and r is a function for calculating the distance in ångström between the given atoms. A good hydrogen bond should have a value of E about -3 kcal/ml, but for DSSP a generous cutoff is chosen to allow for divided H-bonds and for errors in coordinate data [25].

In PDB files there are usually no coordinates for the hydrogen atoms, and therefore they need to be calculated. The location of the hydrogen atom is easily calculated from the other atoms, since the backbone of the polypeptide chain is planar between two C$^{\alpha}$s. This means that the structure in figure 3.1 is all in one plane. In addition it is known that the distance between the nitrogen atom and the hydrogen atom is about 1 Å [10] and the angles between them are all approximately 120°[28].

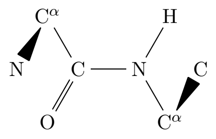


Figure 3.1: The six atoms in the middle of the pictured polypeptide chain, are all located in the same plane, while the bonds to the C$^{\alpha}$ can rotate and thus is what enables the structure to fold into different forms.

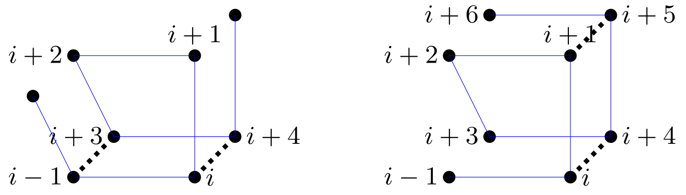
The location of these hydrogen atoms can be calculated under the assumptions that the direction of the vector $\vec{NH}$ is dividing the angle between the vectors $\vec{NC}$ and $\vec{NC^{\alpha}}$ in half [29]. Then the direction of vector $\vec{NH}$ can be calculated by adding the normalized vectors of $\vec{NC}$ and $\vec{NC^{\alpha}}$. To get the length, the vector can simply be normalized, since the distance between the N and H atom is 1 Å. Thus, the calculation is done as:

$$\vec{HN} = \frac{\hat{NC} + \hat{NC^{\alpha}}}{\|\hat{NC} + \hat{NC^{\alpha}}\|} \text{ where } \hat{V} = \frac{\vec{V}}{\|\vec{V}\|} \tag{3.2}$$

To get the actual coordinates of the hydrogen atom, the vector given by equation 3.2 needs to be added to the coordinates of the nitrogen atom: $H_{coord} = N_{coord} - \vec{HN}$.

By this the hydrogen bonds can be calculated using the DSSP algorithm, the coordinates provided in the PDB file and the equation for finding the location of the hydrogen atom.

The DSSP algorithm detects a number of elementary H-bond patterns [25]. Based on those, a number of larger structures are constructed. For detecting α-helices the basic structure needed to be detected is a β-turn, that is created by a hydrogen between residue i and residue $i+4$. These are then used to identify α-helices, by identifying two consecutive β-turns. Such a combination is named *minimal helix*, and is illustrated in figure 3.2.



(a) Turns at $i-1$ and i. **(b)** Turns at i and $i+1$.

Figure 3.2: A simplified representation of a polypeptide chain, where the blue line represents the backbone and the black dots represents residues in the chain. The black, dotted lines represent hydrogen bonds as they would be detected by DSSP.
A minimal helix is identified when two consecutive β-turns have been identified. All identified minimal helices are identical, but they can be combined differently into longer helices, as seen in figure 3.3.

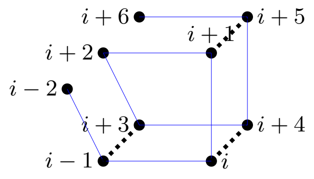
So in this work $E_{i,i+4}$ is calculated using the equation 3.1, to identify two consecutive β-turns, which results in a minimal α-helix. So when $E_{i,i+4}$ and $E_{i+1,i+5}$ fulfills the requirements for a valid energy bond, a minimal α-helix at position $i+1$ is found and the position for that is stored.

When all minimal α-helices are detected, those are iterated to find the full helices of the protein. In this iteration irregularities in the helices are also considered, and therefore a helix does not have to have formed all hydrogen bonds to be considered. In DSSP, this is catered for by allowing a helix to skip as many as two hydrogen bonds between two minimal helices, while still counting it as a complete helix [25]. This is illustrated in figure 3.3.

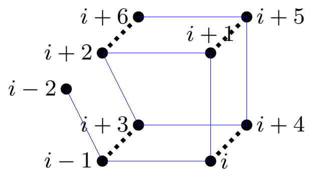
As a final step, the data about helices is stored for usage when calculating the axes of the helices found.

3.2 Finding closeness

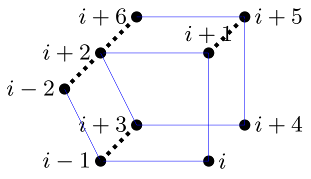
Detecting which helices are close can be done in different ways, depending on the data known as well as the approach taken. In this work, the desired goal is to test the approach of considering the full helix for closeness instead of only parts of it. This will be done by calculating the axes for all helices in a protein, and using those to detect which helices are close. This choice is based on the assumption that α-helices, despite their irregularities, are helix shaped and can be modeled, as such, fairly accurately. With such an assumption, the distance between the axis will closely map to the distance between



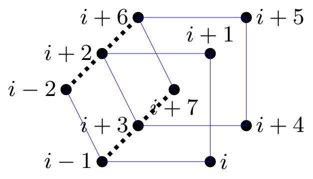
(a) Combination of minimal helices starting in $i-1$ and i ((a) and (b) in figure 3.2.)



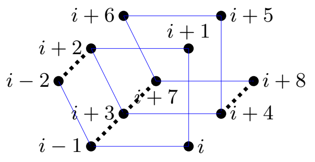
(b) Combination of minimal helices starting in $i-1$ and $i+1$.



(c) Combination of minimal helices starting in $i-2$ and $i+1$.



(d) Combination of minimal helices starting in $i-2$ and $i+2$.



(e) Combination of minimal helices starting in $i-2$ and $i+3$.

Figure 3.3: The figures illustrates how minimum helices are combined into larger helices. The minimal helices can be combined in different ways to create a longer helix. Of the examples above all but (e) will result in detection as a longer helix by DSSP.

the helices. To accomplish this, as a first step, the axes need to be calculated for the helices to be compared.

3.2.1 Calculating axes

When the helices of a protein are defined, the axis of each helix needs to be calculated to make it possible to identify helices that are close to each other. Axes can be defined in several different ways. For a perfect helix, formed as a cylinder, it is fairly simple to identify the axis, but an α-helix in a protein is seldom perfectly formed [30], in that sense. Thus, a method for identifying axes for helices needs to be robust.

There exist many methods for identifying different geometrical properties of protein helices, where usually axis is included as a property. Singh and Brutlag presents a method for identifying the axis of a helix by a method when they are looking at the ends of the helices [31], while Richards and Kundrot presents a method using a method considering using vectors and planes [24].

The method by Singh and Brutlag uses the first (and last) four C^αs and approximates them as a circle. By calculating the centers of mass for the two circles, a starting and end point of the axis of the helix is obtained.

Richards and Kundrot presents a method where they calculate a vector pointing in to the middle of the helix, by calculating axis between three consecutive C^αs and subtracting one from the other. By defining two such vectors with the C^αs next to each other, three planes can be defined (from two perpendicular axes and the cross product of those and the initial vectors). The intersection of those three planes will be an axis point.

The calculation is then done for all C$^{\alpha}$s and the resulting axis is created by calculating the mean of the resulting segments.

In a study by Christopher et al. five different methods for identifying an axis of a helix was compared [32]. The methods compared were (with the names taken from the study, as well as the basis for the description):

parlsq A parametric least squares method, where three dimensional linear least squares fitting is used for the C$^{\alpha}$ to find the axis.

eigenfit A moment matrix method, where the least squares axis is found by creating a least squares matrix from the coordinates of the C$^{\alpha}$ on the helix, and identifying the eigenvectors of that matrix, where the one with the highest eigenvalue is an approximation of the direction of the helix and thus represent the axis.

Åqvist A cylindrical fit method, where the coordinates of the C$^{\alpha}$ are fitted to a cylinder, by minimizing the root mean square difference from each atom to the axis. This does require an initial estimate of the axis to be available.

Kahn A method using cross-product of triad bisectors, where vectors are calculated from the known coordinates of the C$^{\alpha}$ that are perpendicular to the axis. By calculating the cross product of two such vectors, one get a directional vector of the axis. By combining multiple such directional vectors one can use a least squares method to identify the directional vector for the full helix axis.

rotfit A rotational least squares method, where the helix is superimposed to a copy of itself with a displacement of one residue. By using a quaternion-based method, the axis was found.

Of the tested methods, rotfit is the one recommended in the study. It is considered being the quickest to evaluate, while still being accurate and fairly insensitive to noise. Actually, it is described as given ”an excellent combination of speed and accuracy” [32].

Comparing rotfit with the other methods, the method by Singh and Brutlag gives an axis that fits well for a straight helix, but for a kinked or bent helix the variations are not considered at all since the method only considers the first and last residues of the total helix.

The method by Richards and Kundrot does consider all atoms in the helix, but the approach of calculating the mean to achieve the final axis will not give any better result than the least squares method used by rotfit.

rotfit - how it works

The principle of the rotfit method is to let a copy of the helix rotate around itself by moving the start point of the copied helix one residue ahead. Then it is possible to map the ${C^{\alpha}}_i$ of the original helix to ${C^{\alpha}}_i - 1$ of itself, using the copy. By identifying the axis of rotation the axis of the helix itself is also determined. Since an α-helix is not perfectly symmetric, the result is calculated by using least squares fitting to get the best value possible.

The rotation is done by using a fast quaternion transformation method described by Alan Mackay [33], and using this to superimpose the C^α on top of itself [32]. The transformation method results in a linear least squares fitting problem, that is solved by using singular value decomposition. To accomplish this, the origins of the two helices need to be moved, making the centers of mass for the helices the origin of the coordinate system.

Quaternions are an extension of the complex numbers, and are in particular used for rotations in three dimensions where usage of quaternions has a number of advantages over other methods [34]. With a unit quaternion Q with a scalar p_s and a vector $\vec{p}$, the C^α of the copied helix, r' as unit vectors, and of the original helix, r as unit vectors, are related by a rotation θ around an axis with direction cosines l, m and n [33].

Q is not uniquely defined by only one pair of C^α but rather of multiple pairs, since the axis of rotation could lie anywhere in the plane of symmetry. This means that several pairs of C^αs are needed for identifying Q.

With the correct usage, a set of equations is formed that are solved using least squares fitting to identify how much rotation is needed and thus around what axis the rotation needed to be done. Based on the rotational axis, the real axis for the helix can be calculated by moving the rotation axis to the center of mass for the original helix.

As a final step, the first and last residues in the helix are projected onto the axis to get the start and end point of the axis for the specific helix. This is needed when detecting if the points of distance between the helices are actually between the real helices and not only between the calculated axes of the helices.

3.2.2 Finding close pairs

When the axes of two helices are calculated, the distance between those helices can be analyzed. A number of tests need to be performed to determine if the helices are to be defined as close, or not.

The tests carried out to analyze the distance between the helices are divided into a number of steps, to ensure all cases are covered. For all cases, the found distance between helices needs to be below a specified threshold for the helices to be considered as close to each other.

Depending on the test of distance, the helix pairs are grouped into one of four categories. These categories are dependent on what type of tests are carried out to detect the helices, and are illustrated in the coming sections. The illustrations shown are projections of the 3 dimensional environment of the helices onto a 2 dimensional space. By rotating the 3 dimensional structure they will fit into one of the cases described.

The first category is based on calculating the closest distance, while the three other tests are all based on projections of the end points of the helices onto each others axes. This means that if no projection points fall within the helices, and the minimum distance between the helix axes is not within not helices, and the helices will not be detected as close. But if any of the projection points fall within a helix, the remaining tests will be performed.

First category

As a start, the minimal distance between the two axes is calculated using an algorithm for finding minimal distance between two lines [35]. To ensure that the helices are close, and not only the axes, the points of closest distance is checked to see if both points are inside the actual helices. An illustration of the difference can be seen in figure 3.4.

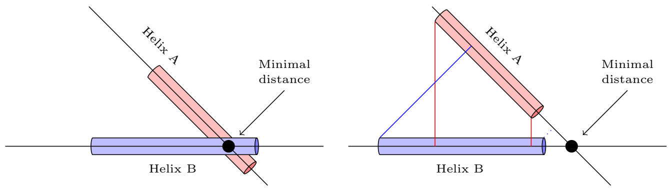


Figure 3.4: Helix closeness is determined by looking at the minimal distance between the actual helices. If that is not found, the projection of the end points of one helix onto the other helix is checked to see if it fulfills the criteria for being close.
The helix pair to the left is a pair where the minimal distance would be used, while the pair to the right would instead use the projection points. In the helix pair to the right, the projections of the ends of the helices are illustrated with colored lines. The filled lines are projection points that fall within the other helix, while the dotted lines are when the projection points are outside of the helix.

The tests of minimal distance does require that the helices are ”crossing” for a point to be detected as close. The possible cases for this is illustrated in figure 3.5.

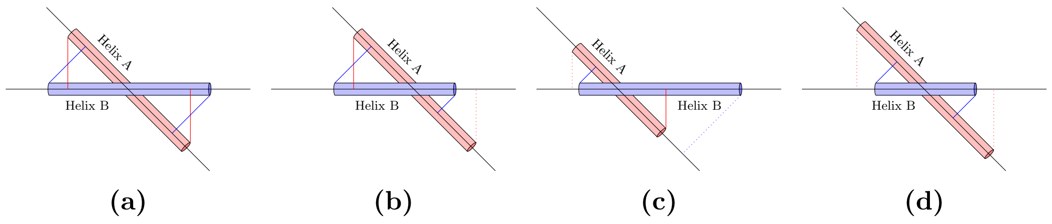


Figure 3.5: Helices where true minimal distance can be used, for determining the distance between the helices, require that the helices intersect with each other. These are the cases when the actual helices intersect and make up the Category 1 in the closeness categorization.

Second category

As the second test for closeness, the projection points from one helix end points onto the other is evaluated. If both helix end points are projected onto the other helix, as illustrated in figure 3.6, the distance between the end point of each helix and the points where they are projected is evaluated. If any of these distances are below a requested threshold, the helices are detected as close.

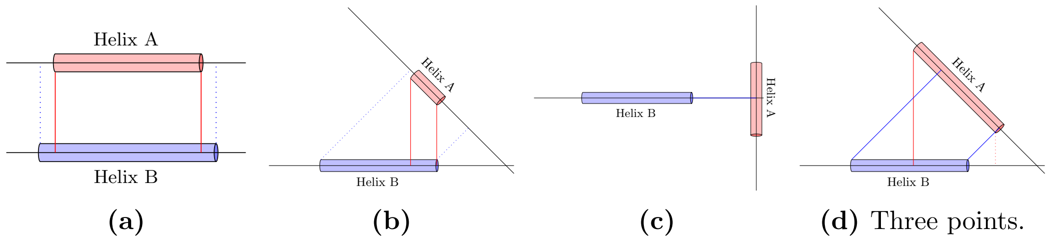


Figure 3.6: The second test for distance evaluates the distance between helices where one of the helices has projections of the end points where both fall on the other helix. These are the cases evaluated and make up the Category 2 in the closeness categorization.

Third category

The third test for closeness evaluates helix pairs where two end points are projected onto the other helices, but not on the same helix, as illustrated in figure 3.7. The distance between the end point and the projection of that end point is evaluated for both cases, and if any of them are below the threshold, the helices are detected as close.

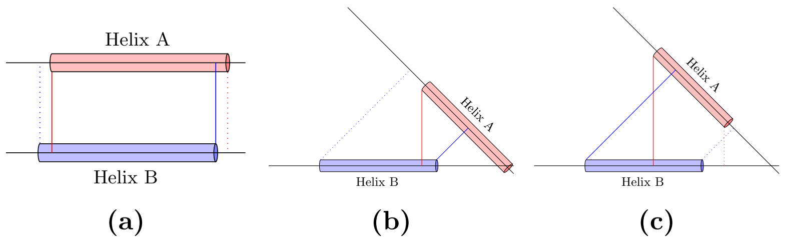


Figure 3.7: The third test for distance evaluates the distance between helices where each helix has a projection point falling within the other helix. These are the cases evaluated and make up the Category 3 in the closeness categorization.

Fourth category

Finally, the fourth test looks at helix pairs where only one projection point is projected inside the other helix, as illustrated in figure 3.8. Once again the distance between the end point and the projection of that end point is evaluated, and if it is below the threshold, the helices are detected as close.

Not detected cases

There are also a number of cases when detection of closeness will never be done. Those cases are illustrated in figure 3.9.

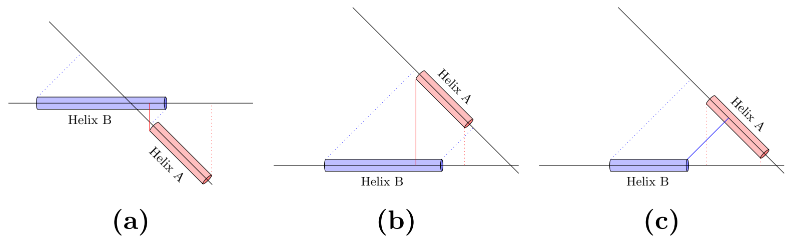


Figure 3.8: The fourth test for distance evaluates the distance between helices where one of the helices has a projection of the end points that falls on the other helix. These are the cases evaluated and make up the Category 4 in the closeness categorization.

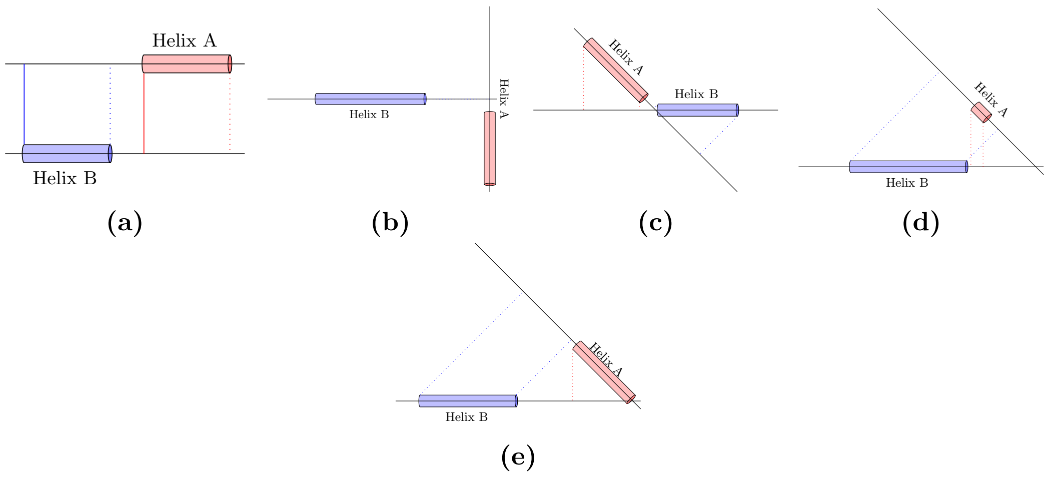


Figure 3.9: A number of cases for how helices are situated will result in that no test for closeness is performed, and no closeness is detected. These are those cases.

Projection length test

For all distance tests where projection points are used, an additional verification test can be done to ensure the helices are somewhat aligned. This is done by looking at the projection of one helix onto another helix. If the length of the projection is at least a set percentage of the original length of the helix, the helices are considered for being close. This eliminates helices based on the angle between them, but also where only a part of the helix is overlapping with the other helix. If they completely overlap, the angle between them will be the fully determining factor, while if they do not overlap fully, the requirement on the angle becomes tighter.

In figure 3.10 three examples are seen, where in example (a) the helices will be tested for closeness if the percentage of the red helix overlapping the blue helix is more than a defined percentage of the red helix. In example (b) the helices are completely overlapping, and therefore the angle between them decides if they will be tested for closeness. In example (c) the combination of the angle and the length of the overlap will determine if the helices will be tested for closeness.

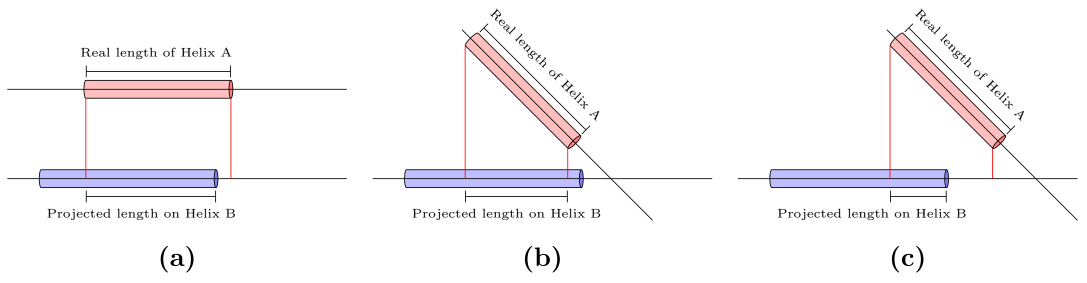

Real length of Helix A
Projected length on Helix B
(a)
Real length of Helix A
Projected length on Helix B
(b)
Real length of Helix A
Projected length on Helix B
(c)

Figure 3.10: An option is possible to limit the number of detected close helices. If used, the projection of the helix onto the other helix must be meeting a threshold set for the length of the original helix. Here three examples of different cases are seen, where in (a) the angle does not impact at all and only the overlap will impact while in (b) the angle is important. For (c) both the overlap and the angle will impact.

3.3 Combining the methods

To be able to identify α-helices, two things were identified as needed: identify helices and identify which helices are close. As has been shown, this is done in three steps, where identifying if helices are close is done by first calculating the actual helix location followed by calculating the axis of every helix. When that is known, the final step is to measure the distance between the found helices.

The identified methods for solving the different areas is the base for the program constructed to identify helices that are close. In chapter 4 the program will be described in more detail.

4

The program - HelixFinder

The methods and algorithms described are implemented in a C program called HelixFinder, where of course also a number of other operations are carried out. The program is structured such that each area is contained within a separate file, and the shared data structures are defined separately, as visualized in figure 4.1.

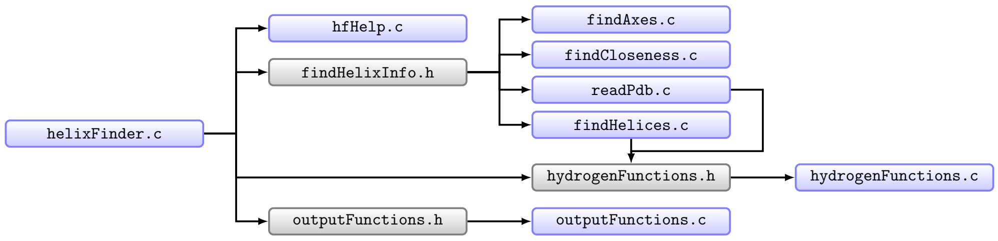


Figure 4.1: The program consists of a number of files, containing the code. The different files contains the functionality of HelixFinder and the functionality is distributed over the different files based on the process of the work of the program.
In addition to the files in the figure, the program has two modules with data structures (*xyzVectors.h* and *datastructures.h*) that most modules use.

The major parts of the program are divided into the different files, where each file contains the functions needed for accomplishing a major step in the program. Those major steps are:

reading data Reading and parsing protein data; done in the file readPdb.c

findHelices Detecting α-helices; done in the file findHelices.c

findAxes Calculating the axes of the α-helices; done in the file findAxes.c

findCloseness Identify α-helix pair closeness; done in the file findCloseness.c

outputFunctions Providing output data; done in the file outputFunctions.c

More information about the files and the content is provided in Appendix C.

4.1 Reading data

The Protein Data Bank files contain data that describes various characteristics of a protein and the structure of it, such as the residue sequence, the coordinates of the atoms in the residues as well as other atoms interacting with the protein. It also contains, among other things, a lot of information about for instance how the structure was determined, what limitations there are when it comes to resolution of the analysis and also possibly some information about the secondary structures in the protein.

PDB files are regular text files with a fixed row width of 80 characters and where each row is tagged with a label described in the specification of the file format [19]. To be able to do the analysis of the α-helices, the data used in the program is the atom data labelled as ATOM in the PDB files. Those lines describe the relevant atoms the protein consists of, with the coordinates for each atom (or combined for a group of atoms) within the polypeptide chain.

Each atom line is parsed and combined at a residue level, where the data is stored in a C struct defined to represent the important data for a residue, as can be seen in listing 4.1. Each such residue struct is, during the PDB parsing, populated with information about the residue type, the residue sequence number as well as the coordinates of the important atoms (C^{α}, C, O and N). In addition, it is populated with information on some data needed for recreating snippets of the PDB file (for reasons described in section 4.6.2). After that, the residues are combined into a list of residues, representing a chain of residues. Those chains are then stored in a chains struct for handling proteins with multiple chains.

Residue struct

```
typedef struct resStruct{
        char nr[5], type[4], chainID;
        int start, end, intNr;
        int isHcalculated;
        xyzVector CA, C, O, N, H;
        struct resStruct *next;
} *residues;
```

Chain struct

```
typedef struct chainsStruct{
        residues chain;
        residues lastRes;
        int count;
        struct chainsStruct *next;
} *chains;
```

Listing 4.1: In the program, data for residues is stored in a residue C struct. The residues contains information about relevant atoms, as well as the type of residue, among other things. The residues are linked in a list, representing residue chains of a protein. Those chains are then contained in a chain struct, where multiple chains are contained.

In addition to the atom information, the program considers information about models in the files, that distinguish between different models of the same protein stored in the same PDB file (labelled with MODEL and ENDMODEL). The consideration of models is limited to selecting the first model in the file. Although only one model is considered, if the protein consists of several chains, all chains are taken into consideration when evaluating the protein data. The separation of chains is identified by the TER label in the PDB file, and the chain identifier is actually not used.

Although it is not part of parsing the data, during the data collection the position of the hydrogen atom is also calculated. This is done since the formula used in DSSP for calculating the hydrogen bonds require information about the hydrogen bonds, and since the data is already iterated the extra calculation is done directly for each residue, using the function seen in listing 4.2 (located in the hydrogenFunctions.c file). The function directly sets the coordinates on the residue provided by the call to the function setCoordinate, that also sets the flag isHcalcualted in the residue struct. The function for calculating the hydrogen atom position does so for all amino acids except for Proline, since that amino acid has no hydrogen atom in the relevant position.

```
void calcH(residues rp) {
        if (savedAtomInfo && (strncmp(getResidueType(rp), "PRO", 3) != 0)) {
                xyzVector atom_H;
                xyzVector atom_N = getCoord(N, rp);
                xyzVector vect_NC = vectorSub(atom_N, atom_save);
                xyzVector vect_NCA = vectorSub(atom_N, getCoord(CA, rp));
                atom_H = vectorAdd(vectorDiv(vect_NC, vectorNorm(vect_NC)),
                        vectorDiv(vect_NCA, vectorNorm(vect_NCA)));
                atom_H = vectorAdd(atom_N, vectorDiv(atom_H, vectorNorm(atom_H)));
                setCoord(H, rp, atom_H);
                savedAtomInfo = 0;
        }
}
```

Listing 4.2: The function for calculating the position of the hydrogen atoms, based on the method described in section 3.1.1. The function also sets the value of the coordinates in the residue. As can be noted, no hydrogen is calculated for the amino acid Proline, because of the special form of that amino acid.

4.2 Finding helices

The entire algorithm for DSSP is not implemented, since only α-helices are of interest for the program. The general algorithm used is described in algorithm 4.1, where the most important parts of the calculations are captured. The status of the calculations is actually not stored as part of the residues, but rather in a string with the length defined by the number of residues in the chain.

Algorithm 4.1 Calculating α-helices in all chains of a protein

```
for all $chain$ in $allChains$ do
  for all $residue_i$ in $chain$ do
    if H-bond between $residue_i$ and $residue_{i+4}$ then
      mark $residue_i$ and $residue_{i+4}$ as minimal helix
    end if
    if $residue_i$ already is end of other helix then
      merge minimal helices to one
    end if
  end for
  for all $residue_i$ in $chain$ do
    if $residue_i$ is start of helix then
      $helixStart \leftarrow residue_i$
    else if $helixStart$ has value and not $helixEnd$ then
      if $residue_{i+1}$ is not in helix then
        $helixEnd \leftarrow residue_i$
        $helixLength \leftarrow helixEnd(\mathrm{id}) - HelixStart(\mathrm{id})$
      end if
    end if
  end for
end for
```

In the algorithm for calculating helices, a function is needed for calculating the H-bond. This function is located in the hydrogenFunctions.c file as shown in listing 4.3. As can be seen in the code listing, the energy level threshold is not a constant. It is actually a variable possible to alter as an input to the program.

```c
int isHbond(residues r1, residues r2) {
        float calcEnergyL, ON, CH, OH, CN;
        if (isH(r2)) {
                ON = 1/(vectorDistance(getCoord(O, r1), getCoord(N, r2)));
                CH = 1/(vectorDistance(getCoord(C, r1), getCoord(H, r2)));
                OH = 1/(vectorDistance(getCoord(O, r1), getCoord(H, r2)));
                CN = 1/(vectorDistance(getCoord(C, r1), getCoord(N, r2)));
                calcEnergyL = Q * (ON + CH - OH - CN) * F;
                return (calcEnergyL < (-energyL));
        }
        return 0;
}
```

Listing 4.3: The function for calculating the hydrogen bonds, as described in DSSP in 3.1.1. The variable energyL has a default value of 15, but can be changed as an option to the program.

There is also an option in the program to disregard a helix if it is shorter than a specified number of residues. This control is done when the length is calculated, and if the length is below the threshold the helix is never saved but simply disregarded.

The data collected is stored in a C struct representing the helices. In the struct data of at what residues the helix start and ends is stored (as links to the residue chains) as well as the length of the helix. In addition, a unique number is set for each helix created, to make it possible to refer to it in for instance output data. Additional data in the helix struct is populated by the functions for calculating axes and closeness of helices.

```
typedef struct helixStruct{
        xyzVector axStart, axEnd;
        residues start, end;
        closeHelix closest;
        struct helixStruct *next;
        int length, nr, isAxisCalc;
} *helices;
```

Listing 4.4: In the program, information about helices is stored in a special C data struct, where information is stored about at what residues the helix starts and stops, the length in residues of the helix, in addition to data calculated in the steps for calculating distance between helices.

When helices are calculated the number of found helices is also calculated, to provide it as a feedback to the user in the result.

4.3 Calculating Axes

Calculating the axis of the helix is done using the rotfit algorithm, as described in section 3.2.1. It does a rotation of the helix around itself to identify the rotational axis. In the algorithm description 4.2 the most important parts are included.

Algorithm 4.2 Calculating axes of all helices in a protein

```
for all $helix$ in $allHelices$ do
    $centerOfMass_A \leftarrow$ center of mass for all $\mathrm{C^\alpha}_i$ in helix
    $centerOfMass_B \leftarrow$ center of mass for all $\mathrm{C^\alpha}_{i+1}$ in helix
    for $residue_i$ in $helix$ do
        $\mathrm{C}^\alpha Cross \leftarrow \mathrm{C^\alpha}_A i \times \mathrm{C^\alpha}_B i$
        for $j = 1$ to 3 do
            for $k = 1$ to 3 do
                $\mathbf{v} = \mathrm{C^\alpha}_A i \times \mathrm{C^\alpha}_B i$
                $\mathbf{a}(i)_{j,k} \leftarrow \mathbf{v}_{j,k}$
            end for
        end for
    end for
    $\mathbf{A^T}b \leftarrow \sum \mathrm{C}^\alpha Cross$
    $\mathbf{A^T A}x \leftarrow \sum \mathbf{a}(i)$
    $axisVector \leftarrow$ solution to $\mathbf{A^T A}x = \mathbf{A^T}b$ with SVD
    $axisStart \leftarrow$ projection of first $\mathrm{C}^\alpha$ on $axisVector$
    $axisEnd \leftarrow$ projection of last $\mathrm{C}^\alpha$ on $axisVector$
end for
```

To simplify the calculations and the code of the program, solving the equation system by singular value decomposition is done using the GNU Scientific Library (GSL) [36], rather than implementing the procedure as part of the program itself. This means that when compiling the program GSL needs to be available.

The data of the axes calculations is stored in the helix struct as two vectors representing the axis start and end points. In addition, the variable isAxisCalculated is set to make identifying if the axis data exists simpler.

4.4 Finding closeness

Closeness is identified by calculating the distance between the axes of two pair of helices. The distance is, as described in subsection 3.2.2, calculated using a number of different tests to cover the different cases that can occur.

Beside the possibility in the DSSP implementation, as a default, there are no limitations as to what helices to consider; if they are found by the DSSP implementation, they will be considered. But to allow for some flexibility, there are two options implemented for disregarding helices if they are too close to each other in the chain, or if they are too far apart in the chain. What defines too close or too far apart is given as input to the program.

The tests and calculations can be divided into two groups:

- Shortest distance between the two axes is used
- Distance between end point projections is used

All in all there are four tests done for distance, where each test represents a category in the four types of distances categories provided. The program also allows for selecting how many of the distance tests to be executed. This is done by selecting a value between 1 and 4, where the number indicates how far in the list of distance tests the program will go. 1 means that only the first test will be executed, 2 means that the first two will be executed, 3 means that the first three will be executed (and thus skipping the last one) and 4 means running all tests (and is thus equivalent to not setting the option at all).

To identify the shortest distance between two helix axes, an algorithm for finding closest path between two lines is used, using geometrical approach [35]. The understanding that two lines, that are not parallel or crossing, have a minimum distance that is ”uniquely simultaneously perpendicular” to both lines. By using that knowledge, an equation for calculating the distance is derived.

When the shortest distance is identified, as described in subsection 3.2.2, the position of the axis is calculated to ensure the position is between the start and end position of the helix. If both points of the shortest distance (one on each helix axis) are within the actual axes, the shortest distance is used as the distance between the helices. If the distance is lower than the specified threshold value, the two helices compared are seen as close.

To allow for flexibility, the threshold defined for the maximum allowed distance between two helices to be considered close, is a variable. That variable can be altered as an option to the program, allowing for varying its value and thus what helices are determined as close.

If the shortest distance calculated between the axes is not located within the helices, the projection distance is instead evaluated. This is, as described in subsection 3.2.2, done in a number of steps, where the test for closeness is stepwise relaxed.

The projection tests are described in algorithm 4.3 and 4.4. A number of values are pre-calculated before the tests are run, to simplify the code. Of those, the most important are the projection points where it is also calculated beforehand if the projection points are within the helix of the axis they are projected on.

Algorithm 4.3 Projection distance test number 1

```
if $p(i)_1$ and $p(i)_2$ are hits then
   if distance between $p(i)_1$ and $p(i)_2$ $> 50\%*$ axis $i$ length then
      closeness is detected
   end if
end if
```

Algorithm 4.4 Projection distance test number 2

```
for each combination of $p(i)_1$ and $p(i+1)_2$ and $a(i+1)_2$ do
   if $p(i)_1$ and $p(i+1)_2$ are hits then
      if distance between $p(i)_1$ and $a(i+1)_2$ $> 50\%*$ axis $i$ length then
         closeness is detected
      end if
   end if
end for
```

In all of the projection distance tests, there is a possibility to enable a test for if the helices have very small amount of overlap or if they have an angle that is deemed to high, as described in subsection 3.2.2. This test is turned off by default but can be turned on by an option to the program. In addition, it is possible to set the percentage of projection overlap used by the program. If no value is set, 50 % is used as default.

When the distance is calculated, and if the helices are considered close, the data related to the closeness is stored in a struct, as described in listing 4.5. This information is then linked to from the main helix evaluated. Thus, stored in each helix struct, there is a link to a list of closeHelix structs describing the relation of one helix to all other helices identified as close.

When a closeness is detected data about the distance, the points of where the minimum distance was detected and the category for the distance test is stored in the closeHelix struct.

```
typedef struct closeHStruct{
        float angle, mindist;
        int resDist, isResDist;
        xyzVector hitC[2];
        struct helixStruct *H;
        struct closeHStruct *next;
        int distTest;
} *closeHelix;
```

Listing 4.5: The information about helices close to each other is stored in a closeHelix struct, where data about the interaction is stored, as well as direct links to the helices in question.

4.5 Other calculations

When helices that are close are identified, two more values are also calculated to allow for different approaches to doing analysis of the data created. The number of residues separating the helices, and the angle between the axes of the helices.

The number of residues between helices is simply calculated by usage of the internal sequence number of the residues, and taking the distance from the end residue of the helix first on the chain to the start residue of the other helix. To handle the case of when the helices are on different chains, a variable is used to keep track of whether the residue distance is set or not.

The angle between the helices is calculated by using the definition of the dot product (where θ is the angle between the vectors):

$$\vec{A} \cdot \vec{B} = \|\vec{A}\|\|\vec{B}\| \cos\theta \iff \theta = \arccos \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\|\|\vec{B}\|} \tag{4.1}$$

Since two points of each axis are saved (the start point and the end point) after the axis calculation, subtracting the start point from the end point will give a standard vector that can be used in the equation.

4.6 Writing results

When all calculations are done, the result is presented to the user. This can be done either as output written to standard output, or a file. In addition, there are two options that impact the output of the program: the program can provide more feedback of the run during execution, and it can generate analysis data for visualization of what has been done.

4.6.1 Execution feedback

When the program is running it provides no information about the status, unless something goes wrong or until the result of the analysis is shown, unless one chooses the verbose mode. Turning on the verbose mode will provide information about what steps the program is taking as well as some information about the findings.

The information given is:

1. what file it is working with and how many chains were found in the file when it is processed
2. that searching of helices has started and that it ended, and the number of helices discovered
3. that axes are being calculated
4. that closeness calculations have started and how many close helices were found, and at what distance
5. that output data is written

All of these feedback messages are handled as part of the main helixFinder program and are produced either when entering a function or at the return of a function.

4.6.2 Analysis data

If the user chooses to, the program can provide files to enable analysis of the run. The data provided is two files containing 1) a snippet of a PDB file, and 2) a script possible to run in the protein visualization tool RasMol [37].

The PDB file is generated by a combination of data generated, and data saved at the initial parsing of the original PDB file. The format does not fulfill the specification of a real PDB file and only contains atom data for the atoms relevant for displaying the protein with some additional data to allow for visualizing axes and distance points. The additional data is really just atoms not used within proteins (iron for the axis points and xenon for the distance point), added under the label HETATM.

The script file is done to enable stepping through the different helices identified as close, one helix at a time. In addition, the script colors the helices, the axes, and the distance between the helices to allow for simple understanding of what the program has actually done.

4.6.3 Result data

As the result of the program, the discovered data is listed. This is done by simply iterating through the lists as described above and displaying the data in a structured format. The format of the output is described in the User Guide included in Appendix A.

5

Results and discussion

Finding α-helix pairs in protein files is one of many important steps in helping to gain a better understanding of how proteins are structured. The program HelixFinder detects such pairs on proteins with good results. It has been tested on some 50 protein files with a varying number of α-helices and the result has been visually analyzed. Both helix detection and closeness evaluation matches the criteria as described.

The algorithms used fit the need of the intended goal of identifying valid α-helix pairs, and in figure 5.1 there is an example of how the close helices are detected around the active helix (where the active helix is colored read and the helices detected as close to the red helix are colored green).

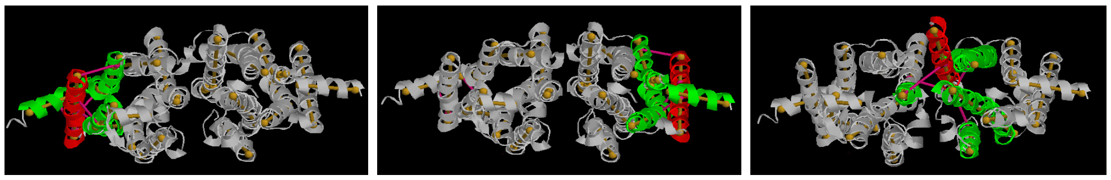


Figure 5.1: Closeness is calculated for every helix detected in the protein. Here some examples are shown how close helices are found when iterating through the helices of the protein. The main helix is colored red, and the helices detected as close to the main helix are colored green.
The protein used is the thermostabilised human a2a receptor, with PDB id 4UG2.

The default minimum distance between helices is set to 15 Å in the program. Although an α-helix has a radius of 2.5 Å there will be additional side chains between the expanding lengthening the actual helix size. Having said that, 15 Å is still a generous maximum value for the distance between two helices. But the argument behind the choice is that allows for reasoning about cut-off points or areas where patterns can be seen to change. The distance is also provided as output data for allowing selection of only a subset of the pairs detected, and it is also possible to set as an option to the program defining the minimum distance to be used if one does not want them in the output at all.

5.1 Flexibility of evaluation

To allow for flexibility and variations of usage of the program, it is possible to alter the evaluation of the program in a several different ways. The options available are fully described in the Reference Manual (included in Appendix A), and gives the user the possibility to change how evaluations are done as well as what is to be considered, in a number of different ways.

Throughout chapter 4 the options are mentioned in relation to the sections of implementation. Here is a summary of them:

energy level Editing the energy level used in DSSP, as described in section 4.2

helix size Disregarding helices shorter than a certain length, as described in section 4.2

chain distance Disregarding helices that are too close and/or too far apart, as described in section 4.4

minimum distance Editing longest allowed distance value, as described in section 4.4

projection angle test Turning on the test of limiting overlap or angle in connection, as described in section 4.4, as well as the possibility to set the percentage value to be used

distance test selection Select how many of the distance tests to use, as described in section 4.4

The options allow for doing different analysis of helix interaction on the same dataset and using the same program. Also, they are independent of each other, and are therefore possible to combine in any way one would like to.

5.1.1 Program options examples

The options for chain distance and helix size, as well as the altering of maximum allowed distance between helices, limits the search space of helices. In figure 5.2 it is shown how some of the options reduced the number of helices detected as close.

This allows the program to be very flexible in the detection, and allowing for a broader usage than only looking at comparison of any type of helix. For instance, it could be relevant to disregard helices that are so tight that the connection between the helices itself have too great impact on the properties of the relation between the two. On the other hand, those type of helices could be just what might be desirable to evaluate. Both possibilities exists by varying the settings of the program.

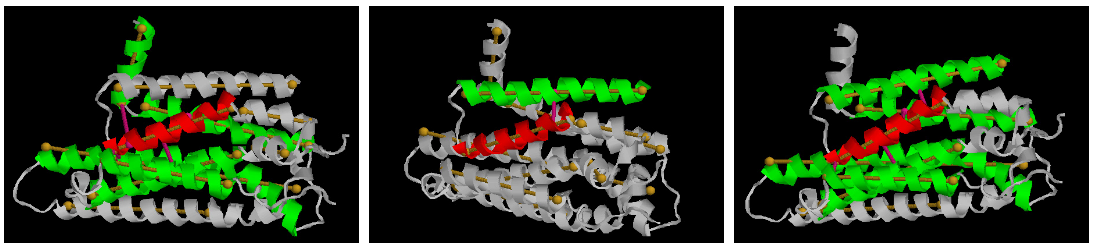


Figure 5.2: When doing a search for close helices, the program allows for a number of settings to reduce the search space. In the first picture, a requirement for a min number of residues on the chain between the helices is set. In the second picture, a requirement for a max number of residues on the chain between the helices is set. In the third picture, only helices that are longer than a max value are considered. The value is set to 8 in all cases. The protein used is the adenosine receptor A2a, with PDB id 3REY.

5.2 Visual feedback

As mentioned previously, in subsection 4.6.2, there is a feature for allowing to create some scripting files for a protein visualization tool called RasMol [37]. The reason for enabling the possibility to create these files is to allow for visual analysis of the result of a HelixFinder run. This feature can easily be used for any user and how to use it is described in Appendix A. In figure 5.3 an example is given of what the visualization can provide.

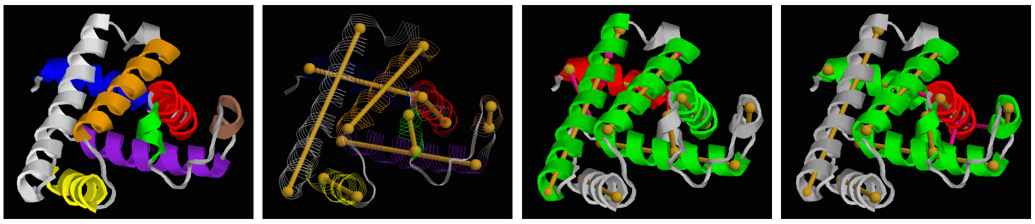


Figure 5.3: An example of the output from the different steps one can see when running the script. The steps in the script file follows the process of the program, and here the steps are shown. The first step shows the helices, the second step highlights the axes, and the finishing steps shown helices that are close to each other (with one step per helix with close helices detected). This is just a sample of all the detected close helices for this protein. The protein used is myoglobin, with PDB id 1A6M.

All protein images included in this report are actually generated using RasMol with the script files as input (both the generated PDB file and the actual RasMol script). The scripts have then been used to get to the desired state of the detection process.

5.3 Usage of HelixFinder

When analyzing proteins, searching for patterns and similarities is one way of gaining a better knowledge of groups of proteins. The HelixFinder can be used for such analysis, when it comes to looking at the relation between, for example, length of helix and distance or angle between helices. Here are some examples of what can be done with some simple scripting.

Besides analyzing single protein files, and visualizing the results, the program can be used for analysis of different interactions between helices in a larger set of protein files. The program only handles one PDB file at a time, but with some simple scripting analysis of larger data sets can easily be done, as described in Appendix B.

5.3.1 Inter-helix angles vs. distance

By evaluating multiple protein files with different properties, and comparing the angle between the helices with the distance between them, it might be possible that there are some patterns to be found.

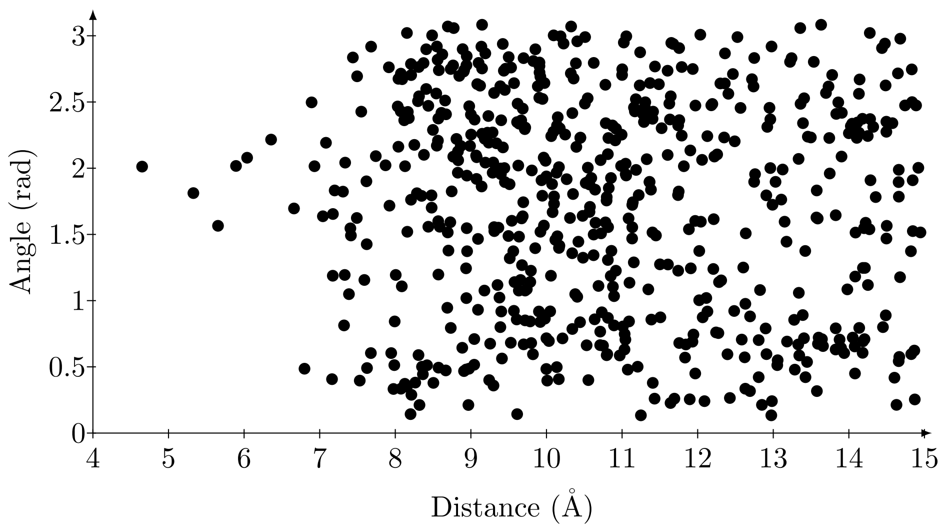


Figure 5.4: A scatter plot over relation between the helix angle and helix distance, as an example of data than can be retrieved from the HelixFinder program.

An example of this is seen in figure 5.4 and is based on a dataset of 47 proteins[1], selected to ensure there is a variation in the selection. They where selected by using the filtering mechanism at RCSB PDB (rcsb.org) and selecting all alpha proteins with a high resolution where the limitation on the sequence identity of maximum 50 % (see Appendix D for more details). Based on this data, the program detected 607 unique helix pairs for evaluation with the maximum distance of 15 Å between them.

One can in the scatter plot see that short distances seems to be occurring at a short range of angles, somewhere in the area of 1.5 to 2 radians. This could be because of a possibility to interlace when not in a more parallel state, and that in that range of angles

[1]Actually only 46 proteins are used since one of the selected proteins (1jni) had too short α-helices to be able to calculate axes.

the helices are actually able to come closer. To verify this though, one would have to look at a much larger set of proteins.

5.3.2 Inter-helix angle distribution

Looking at the scatter plot in figure 5.4 one can see indication of groupings of angles that does not match what would be expected if the angles where randomly distributed. To be able to see if that is the case, random pairs of points on a sphere are generated to ensure a random distribution of angles [38]. Two such numbers are generated at a time, and are treated as unit vectors between which the angle is calculated. 10 000 such pairs are generated to ensure a randomly distributed variation of angles. In the histogram in figure 5.5 the distribution is visualized (the dark bars in the background) and one can see that it is a sinusoidal distribution [39]. This means that the distribution is what is expected if the protein helix pairs has a randomly distributed angular relation.

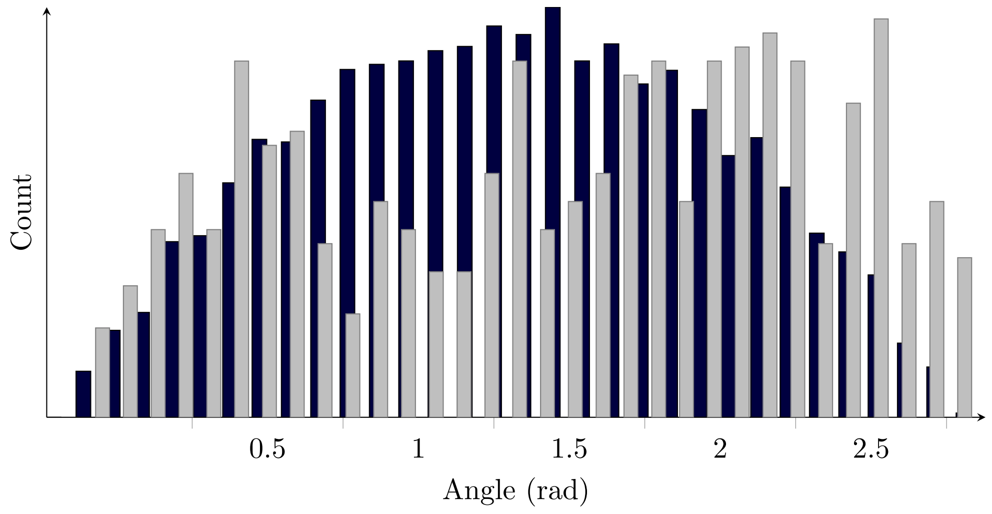


Figure 5.5: An indexed histogram comparing the distribution of angles between the 607 helix pairs of the 47 proteins (light bars) with a randomized population of 10 000 angles (dark bars).

When adding the 607 helix pairs to the histogram in figure 5.5 (the gray bars) it is easily seen that the distribution of the angles is not sinusoidal and does not follow the expected random distribution. In fact, it can be seen that the analyzed helix pairs have a very much higher number of large angles, and also does not have as many angles as expected in the area around 1 - 1.5 radius.

In many proteins with a lot of α-helices it is very common that the helices tend to be roughly parallel or anti-parallel. The higher value of larger angles would indicate that helices are in fact anti-parallel in a greater extent than what would be expected if they where randomly distributed. In addition, the test data shows that angles of higher or lower value is more common than values in the middle of the histogram. This indicates that helices are more often lined up in proteins than crossing in a perpendicular manner.

Further analysis could be done with a larger set of proteins and by reducing the allowed closeness, to see if there are clearer such patterns when helices are closer together.

5.3.3 Sequence separation

If one instead looks at the distance in residues on the chain between helices, HelixFinder provides that data as well in the output. Using a similar approach as described in Appendix B, information about the angle and the number of residues separating the helices can be extracted.

Also here the 46 proteins described earlier are used, but of the 607 helix pairs, only the helices with fewer than 50 residues between them are used. This results in 418 helix pairs where the relationship is visualized in the graph.

Plotting these as a scatter plot (figure 5.6) one can quite easily see that for helices close to each other on the chain, the angles are quite high, while for the other helix pairs it is hard to make any conclusions although there might be a slight predominance of greater angles when the distance is between 15 - 30 residues.

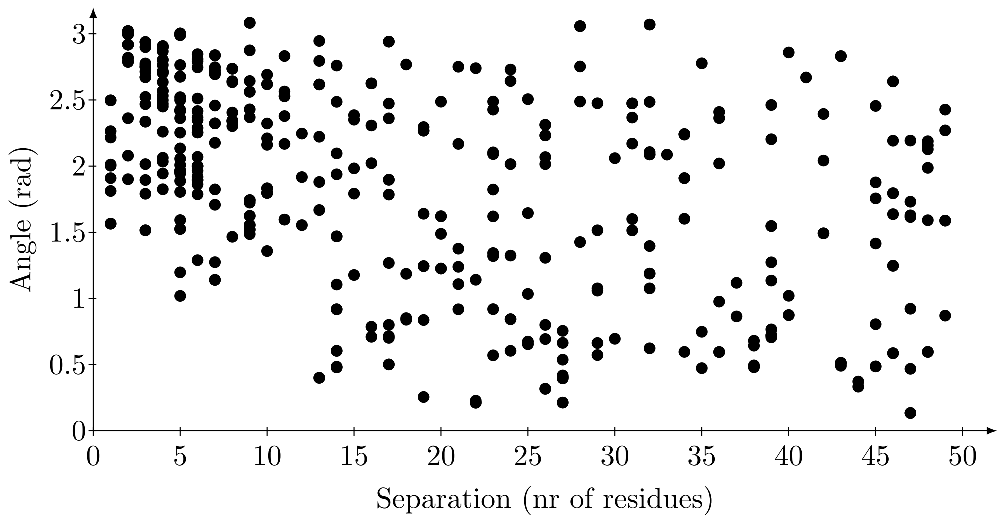


Figure 5.6: A scatter plot over relation between the helix angle and the number of residues between the helices, as an example of data than can be retrieved from the HelixFinder program.

The result of having greater angles for helices being close to each other is not surprising. Although not all proteins are packed in the same way, the structures seen in figures 5.2 and 5.1 is a very common one, with helices laying in a somewhat parallel form. This means that for helices close to each other on the chain they will have a quite large angle.

5.3.4 Helix pair classification

In the result of an evaluation, data for what category of group of closeness is also provided. That data can be used to visualize if there are any patterns between angles and the given test. In figure 5.7 the angles related to the tests are provided. Based on that graph one can see some indication on relationships between category and angles, but more proteins would have to be tested to verify any true patterns between the angles and the tests.

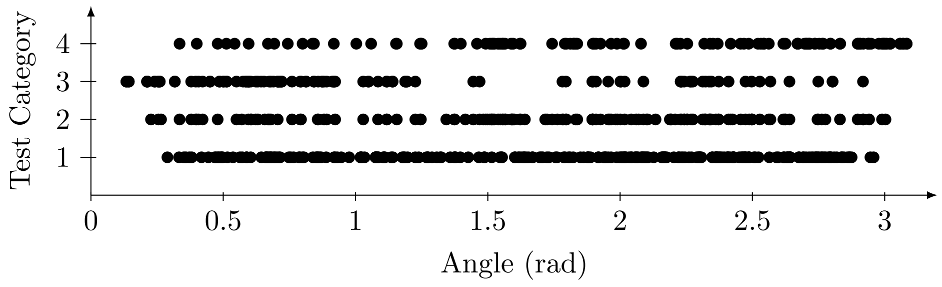


Figure 5.7: A scatter plot of relation between angles between helix pairs and the closeness category it was detected as.

With the data provided in the output it is also possible to combine the data to allow for a visualization of the graph in figure 5.4 with indications of the tests passed, as seen in figure 5.8. In the graph one can also see that the Category 1 closeness pairs seems to be dominating. In fact, the Category 1 pairs are 249, the Category 2 pairs are 146, the Category 3 pairs are 114 and the Category 4 are 115.

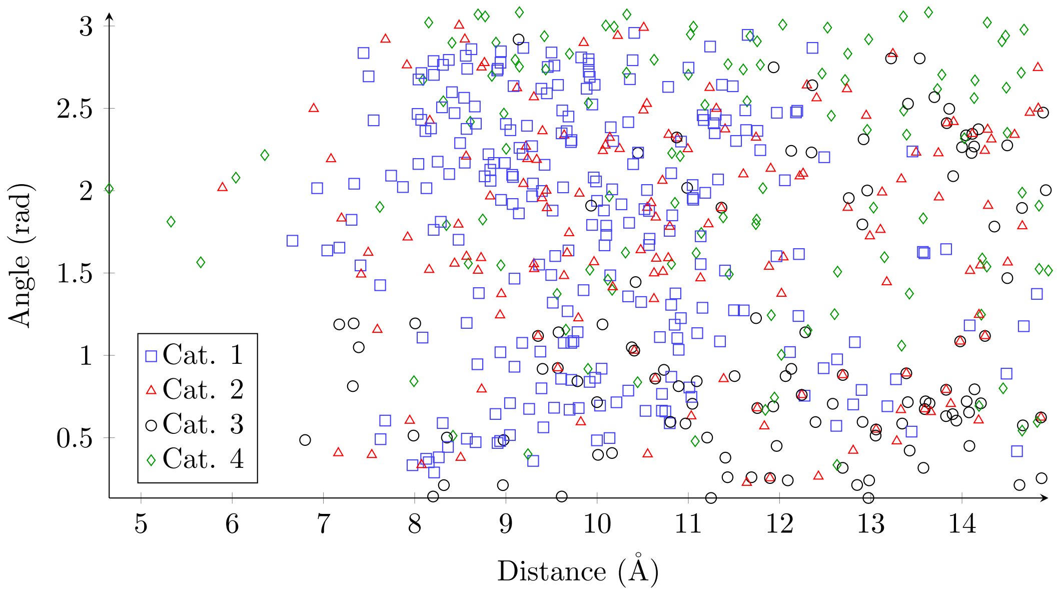


Figure 5.8: A scatterplot visualizing the relationship between the angle and the distance, with added information of what category the pair belongs to.

Looking at the relation between angles and residue distance in the chain, as shown in 5.6, one can also add the categories to the plot for a visualization of any the relation between the angle, the residues between two helices and the category of the closeness, as seen in figure 5.9. Once again one can see that the Category 1 closeness tests is the dominating test. The Category 1 tests are actually 180 of the total 424 helix pairs compared.

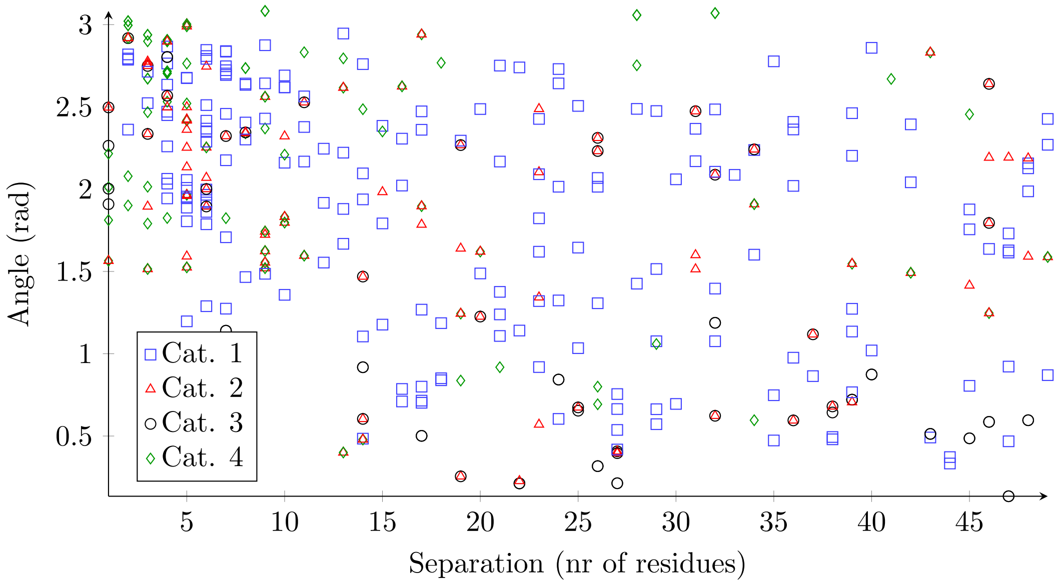


Figure 5.9: A scatterplot visualizing the relationship between the angle and the distance between the residues on the chain, with added information of what category the pair belongs to.

5.3.5 Other comparisons

Other comparisons can of course be done by using the different options to the program, in combination with some parsing of the result files. One could for instance look for frequency of residues on helices that interact with each other (where interaction would be defined as being close enough). Or one could see if there is any correlation between the lengths of helices and distance.

5.4 Performance

In the test runs with the 47 proteins also performance of the application was evaluated. The time it took to evaluate and produce results for those proteins was less than 0.4 seconds[2]. With a larger set of randomly selected proteins, with a large proportion of α-helices, the evaluation time for 1392 protein files was about 15 seconds.

On average each protein takes about 0.01 seconds to execute. Evaluating 100 000 proteins would then be done in under 20 minutes.

[2]The tests where done on a dual core 1.7 GHz Intel Core i5 equipped computer.

5.5 Future work

Although the program works and delivers results in accordance with what was set up as a goal for this work, there are areas where it is possible to evolve the program. Looking at the major parts of the program, there can be seen some possible areas where more work could be done to increase flexibility or even improve the result.

5.5.1 Detection of α-helices

When it comes to detection of helices, it has previously been described, in section 3.1, that there is no objective truth as to where an α-helix starts and where it stops on a polypeptide chain. But there is no other method currently as accepted as DSSP for identifying such structures. As an improvement, an option for using the structures provided in the PDB file could allow for a possibility to change what defines helices for the program.

Another option for improving the detection of helices could be to allow for a selection of algorithms for detecting helices, or at least to allow for also using STRIDE [26] as an algorithm. Potentially an even more advanced approach would be to use both methods, and find a way to combine the results to get an evaluation where the advantages of the different methods could be taken into account.

5.5.2 Detection of axes

For the detection of the axis of a helix, a linear approach is used. This simplifies the detection of closeness, but does not fully fit within the true structure of α-helices, as can be seen in figure 5.10.

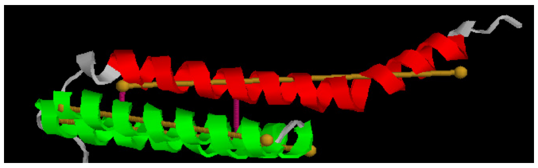


Figure 5.10: A helix that is bent can result in an axis not falling completely within the helix. Here is an example of a helix where the axis is partly outside of the helix itself. The protein used is a section of the leukotriene C4 synthase, with PDB id 4J7T.

Since there is no detection of bent helices, there will be errors in the detection of truly bent helices. Currently there is no way to detect this in the program, but one could look at the shortest distance between each residue and the axis to detect if any of the residues are further away than a set threshold based on the normal radius of an α-helix. Having this would allow for detection and to drop such helices, or parts of them, from the analysis, although it would add a substantial amount of extra calculations.

To enable thorough analysis of bent helices an axis detection method would need to be used that provides bent axes, when needed. Such algorithms exists (see for instance the work of Smith et al. [30]) but this would, however, also require a changed definition of what closeness is for helices and would add complexity to the solution. As a continuation of this work, it could be worth examining the approach of bent helices and providing indications of the deviation, but for the aim of this thesis the approximation of the axis as straight is deemed sufficient to identify helices close to each other.

5.5.3 Detection of closeness

If, when calculating closeness of helix pairs, the projection length test (the requirement that the length of the projection of a helix onto another helix needs to be at least a specified percentage of the helix being projected) is used, it is only done for the distance tests based on projection points (category 2 - 4) and not for the first test of closeness. This means that the first test for closeness can detect helices that would not be detected by the other tests, since they are more limiting in that perspective. To make it transparent, the projection length test should be added to the first closeness test as well. This will reduce the number of detected pairs, but on the other hand it would result in more consistent detections if the option is used.

On the other hand, the projection length test might result in that a large helix is not detected as being close to a small helix, while the small helix could be detected as being close to the large helix, as illustrated in figure 5.11. This will result in symmetric results in some cases, while in other cases the result will be asymmetric, where one helix will be detected as close to another helix, but not the other way around.

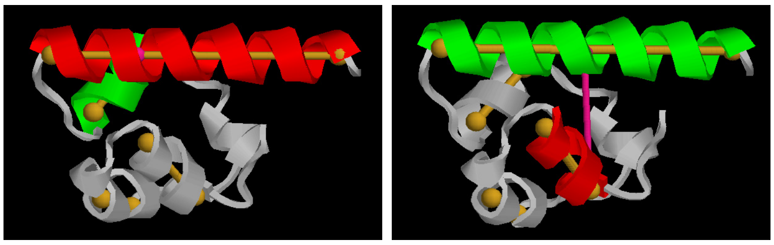


Figure 5.11: The projection length test of the helix closeness detection will result in that the same pair of helices will be evaluated differently depending on which helix is the active helix the other helices are compared with, as can be seen in the two examples.
The protein used is the ephrin type-b receptor 2, with PDB id 1F0M.

The behavior of sometimes reporting duplicate pairs, and sometimes not, could be somewhat confusing. Cleaning out duplicates from the result list would eliminate that potential confusion. The behavior of asymmetry does have a point in that a large helix could be assumed to not be impacted by a small helix in a way that the a larger helix has on the small helix. But especially, the projection length test verifies that the near perpendicular helices are not detected as close. An alternative to the projection length test would be to simply verify that the angle between the helices are below a specified threshold.

5.5.4 Data handling

Data in the program is handled both when reading data to work with as well as when producing the result of the runs.

Reading data

When it comes to reading data of proteins, the program is limited to reading PDB files, and cannot handle the newer XML-based PDBML files. The program also assumes that the PDB files are correctly formatted, and will not detect errors in the format or in the structure of the protein (like number of side chain atoms, distance between the C^{α}s, but also pure formatting of the file). Here improvements could be made to do some analysis of the read data, but for formatting errors to be handled, the best approach is probably to support the PDBML format instead, since format verification is easier with an XML structure.

When reading the files, there are also some limitations as to what is handled in the PDB files. As stated previously, any information about secondary structures contained in the PDB file is ignored. Also, if there is more than one model in the PDB file, the program will use the first one. Data about scale and other coordinate systems used, as described in the PDB file [19], is also not taken into consideration.

In addition, the program only handles single files and cannot process multiple PDB files at once. This is a limitation when it comes to creating data for many PDB files, but on the other hand, as has been shown in producing the data in 5.3 it can easily be scripted.

Writing data

In the result data, the same helix pair can be included twice, since all helices are matched to all helices. This means that when using the result for further analysis, it is important to take into consideration that there can be duplicates and handle those, if they would be a problem[3]. An improvement here would be to make sure the program does not return any such duplicates.

The program could also add even more data describing the relations of the helices. For instance could one add data on the coordinates of the axis points of the helices for analyzing how the helices actually overlap.

Another improvement of the result data would be to change the format to either a comma separated file, or to an XML format. This would simplify the parsing of the data and allow for easier processing of the results of the program.

The generation of the visualization script file could also be improved by supporting also other viewers than RasMol, or entirely replace the RasMol support with another tool. The visualization programs PyMol, Jmol and VMD all support scripting, although none of them share scripting language. The fact that the different programs have their

[3]In the visualization of the result done in section 5.3 the used python script actually removed all duplicates.

own script language would mean that one implementation would have to be done per program, and therefore if any other program should be added, the user base of the different programs should be used to identify what program(s) to support.

6

Conclusions

In the aim of gaining a better understanding of how proteins work and how they interact, analysis of proteins is needed. Searching for patterns in groups of proteins can gain a greater understanding of similarities between proteins. In addition, gathering information about similarities can be used as part in prediction of unknown protein structures.

In this work, a tool for analysis of α-helices and how they interact within a protein is developed. Such analysis is important when trying to understand how proteins function as well as how similar proteins are structured. The intention has been to provide a flexible tool for analysis of α-helix interaction as well as trying to develop a different approach for defining interaction between helices than what has previously been seen.

As has been shown, finding helix pairs can be done in three steps: a) identifying helices, b) identifying axes of the helices and c) calculating the distance between the helices. Identifying helices can be done in different ways, but DSSP as the chosen method is the most commonly used and provides good results for the purpose needed. Detecting if helices are close is, in this work, done by calculating the axis of each helix and measuring the distance between those. Although there is no objective truth about what "close" means or a standard method for detecting helices that are close, the chosen approach provides good results on the tested data and seems to be providing a result that is satisfactory for the majority of the helices.

By detecting helices that are close, one cannot say that those helices are necessarily interacting. The advantage of this design of the program is that it lets the user decide what close means, in the sense of distance between the helices. This flexibility also allows for varying the usage of the program and the data gathered will therefore be context dependent.

In addition, the program categorizes the detected helix pairs into four groups depending on the type of distance measurement done between the helices. The categories are based on how the distance is measured in the program, where the presence of a closest distance value between the axes within both the helices is one category, the other categories depends on the number of overlapping projected end points.

Compared to other studies, like the one developed by Lo et al. [6], the calculation of what is determined as close is done differently, but handles detection of close helices well. In addition, the possibility to vary the distance between helices allows for analyzing break points of when patterns of behavior changes depending on the distance between the helices. Allowing for this type of analysis can give more information of how for instance α-helices in groups of proteins behave or how such groups behave in comparison to other groups of proteins.

The result of the program can be used to statistically investigate patterns of groups of proteins when it comes to relations between the data generated as a result of the runs. But the program could also be used for generating statistical material for scoring functions in protein prediction tools, when it comes to interactions between helices.

Moving forward with this work as a standpoint, there are possibilities to add more analysis data as a result of a program run, like coordinates of the helix axes, allowing for more extended analysis of the result. In addition, creating XML based output data would simplify the parsing of the result.

Improvements could also be made in the area of how the axes are calculated and especially when it comes to considering α-helices that are bent. Of interest could be to add an indicator of how the axis deviates from the helix, and especially if the calculated axis is partly outside of the helix itself.

Also, an extension that could be interesting to explore further, is the possibility to process multiple files at a time and provide a combined result. This would most likely require some more configuration possibilities of the output data, though, since the current result could result in a very long listing of protein data if many files are provided. It is also, as has been shown in this report, fairly easy to create small scripts for creating the same effect.

Though much work remains in the field of protein structural prediction, this work contributes to raising the knowledge about the possibilities around collecting data on structures in proteins. In the work it is shown that it is possible to analyze and gather such data fairly quickly from currently evaluated protein structures, as they are stored in the Protein Data Bank.

Although there are more steps possible to take to advance the program, this does however go beyond the present work. In this thesis a set of methods for calculating α-helix pairs that are within a specified distance is identified, and from that a program that detect helices close to each other is created. A classification scheme for helix pairs has been proposed and implemented. In addition, the program provides the possibility to visually view the result using the RasMol protein viewing program and by the usage of the RasMol scripting functionality it even allows the user to walk through the steps taken by the program.

Bibliography

[1] D. Whitford, Proteins: Structure and Function, J. Wiley & Sons, 2005.

[2] M. Campbell, S. Farrell, Biochemistry, Cengage Learning, 2014.

[3] B. Alberts, A. Johnson, J. Lewis, D. Morgan, M. Raff, K. Roberts, P. Walter, Molecular Biology of the Cell, Taylor & Francis, 2014.

[4] D. Thévenin, T. Lazarova, Identifying and measuring transmembrane helix–helix interactions by FRET, in: Membrane Protein Structure and Dynamics, Springer, 2012, pp. 87–106.

[5] S.-Q. Zhang, D. W. Kulp, C. A. Schramm, M. Mravic, I. Samish, W. F. DeGrado, The membrane- and soluble-protein helix-actome: similar geometry via different interactions, Structure 23 (3) (2015) 527–541.

[6] A. Lo, C.-W. Cheng, Y.-Y. Chiu, T.-Y. Sung, W.-L. Hsu, TMPad: an integrated structural database for helix-packing folds in transmembrane proteins, Nucleic Acids Research 39 (suppl 1) (2011) D347–D355.

[7] A. E. C. Burba, U. Lehnert, Z. Y. Eric, M. Gerstein, Helix Interaction Tool (HIT): a web-based tool for analysis of helix-helix interactions in proteins, Bioinformatics 22 (2006) 2735–2738.

[8] S. J. Fleishman, N. Ben-Tal, A novel scoring function for predicting the conformations of tightly packed pairs of transmembrane α-helices, Journal of Molecular Biology 321 (2) (2002) 363–378.

[9] P. Barth, B. Wallner, D. Baker, Prediction of membrane protein structures with complex topologies using limited constraints, Proceedings of the National Academy of Sciences 106 (5) (2009) 1409–1414.

[10] J. Tymoczko, J. Berg, L. Stryer, Biochemistry: A Short Course, W. H. Freeman, 2012.

[11] C. A. Opitz, M. Kulke, M. C. Leake, C. Neagoe, H. Hinssen, R. J. Hajjar, W. A. Linke, Damped elastic recoil of the titin spring in myofibrils of human myocardium, Proceedings of the National Academy of Sciences of the United States of America 100 (22) (2003) 12688–12693.

[12] M. A. Persinger, A simple estimate for the mass of the universe: dimensionless parameter a and the construct of "pressure", Journal of Physics, Astrophysics and Physical Cosmology 3 (1) (2009) 1–3.

[13] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, P. Bourne, The Protein Data Bank, Nucleic Acids Research 28 (2000) 235–242. URL `www.rcsb.org`

[14] G. H. Lushington, Comparative modeling of proteins, in: Molecular Modeling of Proteins, Springer, 2015, pp. 309–330.

[15] T. Nugent, De novo membrane protein structure prediction, in: Molecular Modeling of Proteins, Springer, 2015, pp. 331–350.

[16] A. E. Torda, Protein threading, in: The Proteomics Protocols Handbook, Springer, 2005, pp. 921–938.

[17] R. D. Schaeffer, V. Daggett, Protein folds and protein folding, Protein Engineering Design and Selection 24 (1-2) (2011) 11–19.

[18] J. Moult, K. Fidelis, A. Kryshtafovych, T. Schwede, A. Tramontano, Critical assessment of methods of protein structure prediction (CASP) — round x., Proteins 82 (2014) 1–6.

[19] wwPDB, Protein Data Bank Contents Guide: Atomic Coordinate Entry Format Description, 3rd Edition (July 2007).

[20] J. Martin, G. Letellier, A. Marin, J.-F. Taly, A. G. De Brevern, J.-F. Gibrat, Protein secondary structure assignment revisited: a detailed analysis of different assignment methods, BMC Structural Biology 5 (1) (2005) 17.

[21] Y. Zhang, C. Sagui, Secondary structure assignment for conformationally irregular peptides: Comparison between DSSP, STRIDE and KAKSI, Journal of Molecular Graphics and Modelling 55 (2015) 72–84.

[22] M. Levitt, J. Greer, Automatic identification of secondary structure in globular proteins, Journal of Molecular Biology 114 (2) (1977) 181–239.

[23] C. Ramakrishnan, K. Soman, Identification of secondary structures in globular proteins - a new algorithm, International Journal of Peptide and Protein Research 20 (3) (1982) 218–237.

[24] D. F. M. Richards, C. E. Kundrot, Identification of structural motifs from protein coordinate data: Secondary structure and first-level supersecondary structure, Proteins: Structure, Function, and Bioinformatics 3 (2) (1988) 71–84.

[25] W. Kabsch, C. Sander, Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features, Biopolymers 22 (12) (1983) 2577–2637.

[26] D. Frishman, P. Argos, Knowledge-based protein secondary structure assignment, Proteins: Structure, Function, and Bioinformatics 23 (4) (1995) 566–579.

[27] J. A. Cuff, G. J. Barton, Evaluation and improvement of multiple sequence methods for protein secondary structure prediction, Proteins: Structure, Function, and Genetics 34 (1999) 508–519.

[28] G. Petsko, D. Ringe, Protein Structure and Function, Primers in biology, New Science Press, 2004.

[29] E. Baker, R. Hubbard, Hydrogen bonding in globular proteins, Progress in Biophysics and Molecular Biology 44 (2) (1984) 97–179.

[30] B. J. Smith, PS – a program for the analysis of helix geometry, Journal of Molecular Graphics and Modelling 33 (2012) 52–60.

[31] A. P. Singh, D. L. Brutlag, Hierarchical protein structure superposition using both secondary structure and atomic representations, in: ISMB, Vol. 5, 1997, pp. 284–293.

[32] J. A. Christopher, R. Swanson, T. O. Baldwin, Algorithms for finding the axis of a helix: fast rotational and parametric least-square methods, Computers and Chemistry 20 (3) (1996) 339–345.

[33] A. L. Mackay, Quaternion transformation of molecular orientation, Acta Crystallographica Section A: Foundations of Crystallography 40 (2) (1984) 165–166.

[34] R. Mukundan, Quaternions: from classical mechanics to computer graphics, and beyond, in: Proceedings of the 7th Asian Technology conference in Mathematics, 2002, pp. 97–105.

[35] D. Sunday, Distance between lines, segments and their CPA, [Online; accessed 18-April-2015].
URL `geomalgorithms.com/a07-_distance.html`

[36] B. Gough, GNU Scientific Library Reference Manual, Network Theory Ltd., 2009.

[37] R. Sayle, E. J. Milner-White, RasMol: Biomolecular graphics for all, Trends in Biochemical Sciences (TIBS) 20 (9) (1995) 374.

[38] E. W. Weisstein, ”sphere point picking” from MathWorld - a Wolfram web resource, [Online; accessed 3-June-2015] (2015).
URL `mathworld.wolfram.com/SpherePointPicking.html`

[39] J. B. Mitchell, R. A. Laskowski, J. M. Thornton, Non-randomness in side-chain packing: the distribution of interplanar angles, Proteins: Structure, Function, and Genetics 29 (1997) 370–380.

A

HelixFinder user guide

This is a user guide, describing how to work with the application HelixFinder, a program for detecting α-helices that are close to each other in protein structure data.

A.1 Introduction

The HelixFinder program is used for detecting helices that are located close to each other in the protein structure. The program is a command line program operating on one Protein Data Bank (PDB) file at a time. It takes a PDB file as input and provides the result as either a text file or as output directly on the screen. For more information about the options possible, see section A.2.

The output of the program can be written to a file or written directly to standard output.

The program generates a simple output file which contains information about:

1. the number of helices found
2. the length of every helix and between which residues the helix lies
3. the coordinates of the C^{α}s included in a found helix and information about what type of residue the carbon belongs to
4. a list of which helices are detected as close to which other helices and also the shortest distance and the angle between them

For more information about the format, see section A.2.1.

A.2 Reference manual

The program is a command line program and is used as follows:

```
HelixFinder [-e n] [-d n] [-r n] [-m n] [-x n] [-sotpavh] filename
```

Where the options available are the following:

-e n: n is the energy level used when the DSSP algorithm tries to detect H-bonds between C^{α} and O (the default value is 0.51). A higher energy level gives a more strict detection of helices.

-d n: n is the largest allowed distance the two closest points on two helices are allowed to be apart (the default value is 15 ångströms).

-r n: n is the minimum number of residues needed in a helix for it to be considered when detecting closeness. No default value is set.

-m n: n is the minimum number of residues allowed between two helices that might be detected as close. That means that if two helices are separated by fewer than z residues on a chain, they will not be detected as close (the default value is 0).

-x n: n is the maximum number of residues allowed between two helices that might be detected as close. That means that if two helices are separated by more than n residues on a chain, they will not be detected as close. No default is set.

-s: makes the program generate a RasMol Script file and a new PDB-file. These files shows in which areas helices where detected and which helices where detected as close. The PDB file is named the same as the original PDB file but with a suffix of "_fake.pdb" and the script file uses the PDB file name but with a suffix of "_script.spt".

-o: returns result of function to file instead of to standard output. File name is same as input file but with '_result.txt' appended at end.

-t: selects how many of the closeness tests that should be used in a run. The values is an integer between 1 and 4 and indicates how many of the relaxations to use. 1 means that only the closes distance tests is used.

-p: sets the program to use the projection limitation tests, to reduce possibilities for when helices are detected close, by comparing the length of the helix with its projection on the other helix. If the projection is below 50 % of the helix length, it will not be detected. The used percentage value can be changed with the option -a.

-a: sets the percentage of overlap for the projection of a helix onto another helix that should be used if the test in -p is used. If -p is not set, the value will be ignored. The default value is 50.

-v: verbose mode, gives feedback while the program is running.

-h: displays a help message with information about the options possible.

A.2.1 Output

The program provides the result of the run either as a printout at the end of the program, or in a file named after the PDB file, but with a suffix of "_result.txt". In addition one can get a number of script files and a generated PDB file, to allow for visualizing the result in the protein visualizing program RasMol.

Result format

The result of the analysis can either be written to a file or displayed directly to standard output. The result contains a section about the helices detected and a section describing the closeness of the protein. The file contains comments where each line with a comment starts with a hash sign (#), to make it easy to parse.

The file starts with the number of helices, followed by a section for each helix, with a number (which is used later to reference for the close helices), the length of the helix, the residue number from the PDB file and a listing of all residues with the coordinates of the C^{α}s.

Following the listing of the helices, the section of what helices are detected as close is listed. That list consist of a pair of references to the number for each helix, the calculated minimum distance between the helices, the angle between them and the number of residues between the helices. In this list, a number of duplicates will be present, with only the order of the helix numbers swapped.

If the helices are in different chains, the value for residues between helices will be set to -1.

Each section is labelled in upper case. The labels are HELIXCOUNT, for the number of helices, HELICES, for the listing of helices and PAIRS for the helix pair information.

A shortened example is shown in section A.3.

Scripts

If the option of getting script files is chosen, two files are generated:

PDB file A faked PDB file, with only the atom coordinate data remaining and with some extra data added to enable showing axes and closeness data visually in RasMol, using the script file generated.

script file A RasMol script for displaying what the program has calculated, showing the detected helices in different colors, and then gives the possibility to step through all helices that are detected as close. It is possible to step through the helices one by one, showing all helices that are close as well as the axis of each helix and the points of where the closest distance was calculated.

The files can be used in RasMol to visualize the result of a HelixFinder run.

A.3 Example of result

Here an example of the output format is provided, with the majority of the residues listed in the helix section removed.

```
# Number of helices:
HELIXCOUNT
16

# Helix number, number of residues, residue interval
# followed by the residue type and its alpha-C coordinates.
HELICES
1  14  : 17 - 31
TYR  17.072 12.523 19.569
[...]
ASP  16.793 28.542 7.672

2  9  : 43 - 52
LEU  9.678 12.420 30.889
[...]
ALA  3.182 21.761 32.436

# Helix pairs (references to above), closest distance,
# angle between helix axis, residues between helices
# and test detecting the pair.
PAIRS
1   2  11.590  0.168  11   1
2   1  11.590  0.168  11   1
```

A.4 Compiling program

HelixFinder is a program written in C consisting of 15 files, as listed below in table A.1. In addition to the files listed, the GNU Scientific Library [36] is needed to be able to compile the program.

Table A.1: The files the HelixFinder program consists of

helixFinder.c	findHelixInfo.h	outputFunctions.h
hydrogenFunctions.h	xyzVector.h	datastructures.h
readPdb.c	findHelices.c	findAxes.c
findCloseness.c	outputFunctions.c	hydrogenFunctions.c
datastructures.c	xyzVector.c	hfHelp.c

The source code will be available for download from GitHub as a git repository, at the link `https://github.com/helixfinder/HelixFinder.git`.

No make file or configuration files are included in the download.

B

Result parse example

This is an example of looking at a large set of PDB files and running and doing some analysis of all files at once. It is some bash commands in combination with a simple python file allowing for running the program on multiple files at the same time and creating a simple output. The format of the output is created to fit the needs ot the LaTeXpackage Ti*kZ*, so that plots can be generated directly into the report.

With the assumption that all files are downloaded into one single folder, it is possible to get the result for all files with one command. On a machine with the shell program *bash* the following command does this:

```
$ find . -name "*.pdb" -depth 1 -exec helixfinder -o '{}' \;
```

The result of the operation will generate result files for all PDB files in the folder the program is run from. By applying a small script to the result files of the HelixFinder run, the interesting data from the result files can then be parsed by using the following bash command (assuming the only .txt files in that folder are the result files):

```
$ find . -name "*.txt" -exec ./get_result.py '{}' \; > summary.txt
```

On the next page is the program listing of the rudimentary python script get_result.py used to parse multiple result files. The program reads one result file and prints the angle and distance for each unique helix pair to standard output.

```
#!/usr/bin/env python
import sys

if len(sys.argv) > 1:
        filename = sys.argv[1]
        file = open(filename)
        text = file.read()
        file.close()
        lines = text.split("\n")
        protein = filename.split("_",1)[0].split("/")[-1]
else:
        sys.exit(0)

l = lines.pop(0)
while l != "PAIRS":
        l = lines.pop(0)

pairs = set()

for l in lines:
        e = l.split()
        if len(e) == 4:
                if e[2] == "nan" or e[3] == "nan":
                        continue
                elif e[0] > e[1]:
                        pairs.add((e[1], e[0], e[2], e[3]))
                else:
                        pairs.add((e[0], e[1], e[2], e[3]))

while len(pairs) > 0:
        print "%s %s   %s" % (pairs.pop()[2:] + (protein,))
```

C

Program information

This appendix contains a listing of all files in the program and some more information about the content of the files, as an addition to the information provided in chapter 4. Mainly it lists the functions shared as well as file size of each file.

helixFinder

The file helixFinder.c is the main file for the program and contains the main function for the program. Here the parsing of the commands is done and all other major functions are called from the main function.

hfHelp

The file hfHelp.c contains help functions for displaying short help when faulty input is given, and a longer help for when help is requested.

datastructures

The files datastructures.h and datastructures.c contain the data structures used in the program, where the main structures are described in chapter 4. In addition, functions for manipulating the structures are provided as well as for reading all data.

xyzVector

The files xyzVector.h and xyzVector.c contain data structures and functions for doing vector operations as well as a vector data structure for 3 dimensional vectors. Besides a function for setting the values of a vector, the following functions are provided in xyzVector.h:

vectorAdd Function for calculating the sum of two vectors.

vectorSub Function for subtracting a vector from another vector.

vectorDot Function for calculating the scalar product for two vectors.

vectorNorm Function for calculates the norm of a vector.

vectorDistance Functions for calculates the distance between two vectors.

vectorMult Function for scaling a vector, with a given float value.

vectorDiv Function for dividing a vector with a given float value.

hydrogenFunctions

The files hydrogenFunctions.h and hydrogenFunctions.c contains functions for calculating the location of the hydrogen atom as well as the hydrogen bond as described in the DSSP algorithm.

outputFunctions

The files outputFunctions.h and outputFunctions.c contains the functions for writing the output data of the program. The following functions are provided in outputFunctions.h:

printPDBlines Function for printing file in PDB format, with only the atom coordinate information from the original PDB file as well as new atom coordinates for representing the helix axes and the closeness points.

printHelixScript Function for printing the part of the RasMol script that visualizes the helices.

printClosenessScript Function for printing the part of the RasMol script that visualizes the axes and the closeness distance points.

printOutput Function for printing the result file of the program.

findHelixInfo

The file findHelixInfo.h is the header file for the files readPdb.c, findAxes.c, findCloseness.c and findHelices.c. The file describes the following functions:

read_pdb Function for reading and parsing the PDB file. This function is implemented in the file readPdb.c.

findHelices Function for finding the helices of the read PDB file according to the implementation of the DSSP algorithm. This function is implemented in the file findHelices.c.

setMinHelixSize Function for setting the minimum length of a helix for it to be considered. The function overrides a global variable in the file findHelices.c.

calcAxes Function for calculating the axis of each helix as described by the algorithm rotfit. The function is implemented in the file findAxes.c.

calcHelixCloseness Function for calculating the distance between helices and detect if they are close, based on the different tests described in the report. This function is implemented in the file findCloseness.c.

setMinChainDist Function for setting a value for minimum number of residues between helices on the chain for them to be considered for closeness. The function overrides a global variable in the file findCloseness.c.

setMaxChainDist Function for setting a value for maximum number of residues between helices on the chain for them to be considered for closeness. The function overrides a global variable in the file findCloseness.c.

setMaxProc Function for setting a value for the percentage value used for the projection length tests, if it is activated. The function overrides a global variable in the file findCloseness.c.

D

Proteins used for scatter plots

THIS APPENDIX contains a listing of all proteins used in the usage examples of the program HelixFinder, in section 5.3. In total it is 47 proteins used and they are identified using the advanced filtering option at http://www.rcsb.org with the following combined search criteria:

1. X-ray resolution at a maximum of 1.25 Å
2. SCOP classification of all alpha proteins
3. Only one chain per protein
4. A sequence identity of 50 %

Proteins

The PDB ID and the molecule name for all 47 proteins. Of these, 1JNI did not generate any result because of too short helices.

3BVU	Alpha-mannosidase 2
2P5K	Arginine repressor
2V7F	RPS19E SSU ribosomal S19E
2V9V	Selenocysteine-specific elongation factor
2NSZ	C-terminal MA3 domain
2CIW	Chloroperoxidase
2FBA	Glucoamylase GLU1
2EUT	Cytochrome c peroxidase
1XMK	adenosine deaminase
1TUK	Nonspecific lipid-transfer 2G
1Y6X	Phosphoribosyl-ATP pyrophosphatase
1XQO	8-oxoguanine DNA glycosylase

1T8K	Acyl carrier
1TQG	Chemotaxis cheA
1TU9	Hypothetical PA3967
1RWH	Chondroitin AC lyase
1J0P	Cytochrome c3
1QV1	Obelin
1LWB	Putative secreted
1H12	Endo-1,4-beta-xylanase
1N40	Cytochrome P450 121
1LB3	Ferritin light chain 1
1M15	Arginine kinase
1LS1	Signal recognition particle
1L9L	Granulysin
1MC2	Acutohaemonlysin
1M1Q	Small tetraheme cytochrome c
1JNI	Diheme cytochrome C NAPB
1K8U	S100A6
1GKM	Histidine ammonia-lyase
1KWF	Endoglucanase A
1JFB	Nitric-oxide reductase cytochrome P450 55A1
1E29	Cytochrome C549
1I2T	Hyd
1G4I	Phospholipase A2
1I8O	Cytochrome C2
1I27	Transcription factor IIF
1EXR	Calmodulin
1C75	Cytochrome C-553
1MUN	Adenine glycosylase
1A6M	Myoglobin
1NKD	Rop
2PVB	Parvalbumin
1BKR	Spectrin beta chain
2ERL	Mating pheromone ER-1
1CTJ	Cytochrome C6
1YCC	Cytochrome C