TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Effiziente Algorithmen

Fast Structure Searching for Computational Proteomics

Hanjo Täubig

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Arndt Bode
Prüfer der Dissertation:	
1.	Univ.-Prof. Dr. Ernst W. Mayr
2.	Univ.-Prof. Dr. Dr. h.c. mult. Wilfried Brauer, em.

Die Dissertation wurde am 27.02.2006 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 17.04.2007 angenommen.

Document Classification according to ACM CCS (1998)

Categories and subject descriptors:

F.2.2 [Analysis of Algorithms and Problem Complexity]:
Nonnumerical Algorithms and Problems—Pattern Matching

H.3.1 [Information Storage and Retrieval]:
Content Analysis and Indexing—Dictionaries, Indexing methods

H.3.3 [Information Storage and Retrieval]:
Information Search and Retrieval—Clustering, Information filtering, Search process

J.3 [Computer Applications]:
Life and Medical Sciences—Biology and Genetics

General Terms: Algorithms, Design, Measurement, Performance

Abstract

In 2003 the Human Genome Project and Celera Genomics celebrated the completion of sequencing the human genome. Although the project was a great success, it had become apparent that knowledge of the sheer sequence of amino acids would not allow to make significant progress in curing any illness. The usefulness of the results and methods was mostly restricted to detecting predisposition to a variety of diseases, but the responsible gene did not provide much information on the real reason of the disfunction. The key to a better understanding of the functional relations must therefore be the structure of the agents that are participating in the respective processes. In the majority of cases, these reactions involve biochemical macromolecules like proteins or nucleic acids. Their structural diversity, created by alternative splicing and post-translational modifications, is a prerequisite for performing the vast number of different functions, that depend on the chemical specificity of the reactants.

Starting in the seventies, molecular structures of polypeptides and nucleic acids (determined by x-ray crystallography and NMR spectroscopy) were deposited in the Protein Data Bank (PDB), which is the primary source for structure information used in today's molecular biology and structural genomics research. Since its first days, the PDB has witnessed a rapid growth at exponential rates. Today, it holds more than 30.000 structures, and the size of the database exceeds twenty gigabytes. These huge numbers emphasize the urgent need for fast methods allowing to search the database of existing structures.

This thesis aims at exploiting the advantages of text indexing methods commonly used in pattern matching for solving problems in structural genomics and computational proteomics. In particular, we apply suffix trees to problems related to structural databases of biopolymers like RNA and proteins.

The main contribution is a novel approach for fast searching in huge databases like the PDB. While existing methods provide only search times in the order of minutes, hours, or even days, our approach allows to perform standard queries within seconds. The data structure is based on a generalized suffix tree which is extended by a method for approximate matching of special adapted alphabets. These alphabets rely on the discretization of translation- and rotation-invariant measures that represent the structure of the protein backbone. The method was evaluated by applying structural queries to the PDB and comparing the results to established tools in this area. On the one hand, the experiments demonstrate a significant reduction of the query time while comparable results are produced. On the other hand, several structures have been found by our approach that are missing in the result list of other tools because they are filtered out by sequence-based heuristics.

Another contribution is a method for identifying frequent motifs and for partitioning a database of protein structures according to structural similarity which is, in contrast to currently used methods, fully automatic. The approach is based on the fast computation of the (sparse) similarity matrix which yields a partition using a spectral clustering algorithm.

Acknowledgments

To write this thesis would have been absolutely impossible without the permanent help of a couple of people.

First of all, I want to thank my advisor Ernst Mayr, for the invaluable support throughout the last years. I am thankful for a lot of insight into theoretical computer science and for a lot of freedom to realize my own ideas.

I am deeply indebted to the current and former members of the Efficient Algorithms Group, in particular Stefan Eckhardt, Jens Ernst, Alexander Hall, Volker Heun, Klaus Holzapfel, Sven Kosub, and Thomas Schickinger. My colleagues Arno Buchner and Jan Griebsch deserve special thanks, this work would not exist without their support. I am grateful to Moritz Maaß and Johannes Nowak for fruitful discussions and for being true friends. Moreover, I am thankful to Ernst Bayer for generous technical support.

For great support regarding the creation of graphics, I am thankful to my dear friend Alexander Mellich.

Furthermore, I want to thank my former teachers Michael Fothe, Udo Weitz, Hans-Joachim Brenner, Bernd Licht, Karl-Heinz Nießler, and Walter Schreiter for teaching me the fundamentals of computer science, mathematics, and chemistry.

Also, I am grateful to Uwe Römers and Anselm Kusser for excellent work on the protein structure project.

For the unbelievable and great support during the last hard days of writing, I am indebted to my family. The help of my brother Holger and my father Klaus Täubig pushed me forward and saved the day.

Last, but not least, I want to thank Anja Fischer for being patient and for believing in me all the time. You are my sunshine. Thank you so much for your love.

Contents

List of Tables

List of Figures

List of Algorithms

Preface

The subject of this thesis is inherently interdisciplinary. In the main, it can be classified as a contribution within computational biology (bioinformatics). In contrast to many other approaches, we focus on practical applicability rather than theoretical worst-case or average-case analysis (which often hides large constant factors in the complexity, that prevents it from being useful in practice). The thesis considers problems of structural genomics and computational proteomics that are solved by applying pattern matching techniques to data mining in biopolymer structure databases like the PDB.

This thesis is structured as follows:

The first chapter introduces the reader into the field of genomics and motivates the necessity to consider certain problems in the developing subject of proteomics.

The second chapter gives an overview of the biochemical properties of macromolecules like DNA, RNA, and proteins. It explains the basic principles of nucleic acid and protein structure.

In the third chapter, we explicate the principles of suffix trie and suffix tree construction and usage.

The next chapter gives an overview of existing work regarding structure searching. We discuss the drawbacks of the currently used methods. Afterwards, the construction of a new structure index is treated, together with a suitable structure representation for polypeptides and nucleic acids. We consider several measures of structural similarity and apply the new representation to the entries of the PDB. Methods for exact and tolerant searching are discussed. The chapter concludes with the description of various experiments that describe the application of the new method to structure searches in the PDB.

The fifth chapter applies the new method to the problems of finding frequent substructures and computing a classification for entries in structural databases.

The pictures illustrating the structure of proteins were created using a PDB file viewer that is an integral part of a protein structure mining tool written by the author of this thesis during the last four years. Most of the diagrams were created using the `gnuplot` program.

Chapter 1

Introduction

At the end of the day, proteins, not genes,
are the business end of biology.

Proteomics' new order,
Editorial of Nature 437

1.1 Motivation

1.1.1 The Human Genome Project

In 2003 the Human Genome Project (HGP) celebrated the completion of sequencing the human *genome*. The stem of this name refers to the Greek word ”genos” for origin, and the suffix ”ome” stands for all, every, or complete. It means the whole hereditary information that is encoded in the *DNA* (deoxyribonucleic acid) of an organism. The study devoted to the (global) properties of genomes is called *genomics*.

In the first stages of the HGP, researchers hoped for major advances in medicine that should result from the knowledge of the estimated 100.000 genes of the human genome. Although the project finished sequencing the roughly three billion base pairs two years ahead of plan, another goal of the project, the determination of all genes, is still in progress. A very surprising result was the updated estimate of the number of genes: between 30.000 and 40.000. Later on, even lower numbers of 20.000 to 25.000 were predicted, which is within a factor of two compared to seemingly simple model organisms like *Drosophila melanogaster* (fruit fly) and *Caenorhabditis elegans* (roundworm). It is actually less than the number of genes of *Arabidopsis thaliana*, a small plant(!). Consequently, the question arises, what makes the difference between highly developed and comparatively simple forms of life.

Although the Human Genome Project was a great success, it had become apparent that knowledge of the sheer sequence of amino acids would not allow to make significant progress in curing any illness. The usefulness of the results and methods was mostly restricted to detecting predisposition to a variety of diseases, but the responsible gene did not provide much information on the real reason of the disfunction. Hence, there was no starting point to interfere with the biochemical reaction system of the pathological cells. The key to a better understanding of the functional relations must therefore be the structure of the agents that are participating in the respective processes. In the majority of cases, these reactions involve biochemical macromolecules like nucleic acids and proteins.

1.1.2 The Importance of the Proteome

According to the standard pathway of information flow in molecular biology (nowadays spuriously cited as the *Central Dogma*), many parts of the DNA sequence are transcribed into *RNA* (ribonucleic acid) which is then, at least in large part, translated into *proteins*. Proteins are one of the most important classes of biochemical macromolecules. Together with nucleic acids, polysaccharides, and lipids they make up the primary constituents of living cells. Following the notation of *genome* and *genomics*, the entirety of all proteins is called the *proteome*. On the one hand this term may relate to either a particular cell type or a whole organism, on the other hand it can refer to a specified point in time or the whole life cycle. The respective subject of scientific study is called *proteomics*.

Proteins perform a rich variety of biological functions: *enzymes* catalyze biochemical reactions, *transport proteins* carry substances, *storage proteins* retain energy providers, *contraction proteins* facilitate motion by muscles, *structure proteins* provide a scaffold for things like hairs and feathers, *antibodies* enable the response reaction of the immune system, and, last but not least, *hormones* and *transcription factors* regulate biochemical processes of an organism. Altogether, proteins are involved in each and every process that is essential for life.

1.1.3 Sequence Determines Structure Determines Function

The reason for this supremacy in controlling the processes of life is the diversity of protein structures. Although all proteins are made up according to the same simple principle of construction, that is, they are chains of amino acid residues, they can adopt a rich variety of different shapes. This process of forming three-dimensional structures is called *protein folding*. As a matter of fact, each amino acid sequence folds (under natural conditions) into a unique structure. This so called *native state* determines the possible *functions* of the protein. In eukaryotes, the structural diversity of proteins is considerably increased by *alternative splicing* (see next chapter). This process is extensively found in human cells, a fact that seems to explain the small number of human genes. Further diversity is introduced into protein structure by *post-translational modifications*.

1.1.4 Determination of Structure

While determining a protein's sequence of amino acids is quite simple (at least from today's point of view), the experimental determination of its three-dimensional structure is a longsome, difficult, and expensive process. In spite of the great efforts for predicting protein structures from scratch with the help of computers (so called *ab initio* methods), there has been only moderate progress in solving this problem. Thus, the three-dimensional structures of proteins are determined by X-ray crystallography or NMR spectroscopy. The data is stored in 'databases' like the popular Protein Data Bank (PDB). Due to the explosive growth of this database (see Figure 2.19), there are urgent needs for fast methods that allow to find similar structures in the database. This can be used to answer the question whether a newly determined protein structure contains an already seen fold. On the other hand this would offer unprecedented possibilities to project knowledge to novel proteins – if structural similarities among proteins can be identified efficiently. Proven and efficient methods based on the amino acid sequence of proteins exist, yet an urgent interest in methods for structural comparison remains for several reasons. Since the function of a protein is largely determined by its three-dimensional shape, the structure is better preserved than the sheer sequence of amino acids, and hence may reveal evolutionary or functional relationships even if a similarity among the sequences is no longer detectable. Thus, structural similarities may provide clues about the function of a new protein. Furthermore, frequently occurring substructures may provide hints for functionally active sites of molecules. Finally, structural analysis may aid our understanding of the principles and rules of protein folding and architecture.

1.2 Structural Bioinformatics and Computational Proteomics

While sequencing the human genome was a huge project and a matter of 13 years, a much more complicated task is to be accomplished next. It has become apparent that there are many problems in medicine and computational biology that cannot be solved based on the DNA sequence alone. Often it is essential to look at the particular biochemical processes which are necessarily dominated by the structure of the reaction participants. The main reasons, why proteomics is an even more challenging field of research than genomics are the following. On the one hand, the genome is static and the same in all cells of an organism, whereas the proteome heavily depends on the genes that were expressed in the cell under consideration. These can be very different for cell types of different tissues or even in the same cell at different points in time. On the other hand, there are much more proteins found in the proteome of eukaryotic cells than protein-coding genes in the genome. This seems to be somewhat surprising at first sight, but can be explained by a process called *alternative splicing*. Moreover, most of the translation products are subject to *post-translational modifications* (see next chapter) to finally yield the functional protein.

Besides the problems in determining structures from crystallography or spectroscopy experiments, there are several subjects considered important in the field of *structural bioinformatics*, in particular within the emerging field of *computational proteomics*. These are (among others) the representation and parameterization of structures, the storage of structures in databases and methods for efficient access, methods for extracting new knowledge from the databases (e.g., identification of frequently occurring substructures, detection of active sites), optimal pairwise superposition or multiple alignment of structures, structure comparison, structure-based clustering and classification, as well as prediction of biopolymer structures from the sequence of monomers. For a more detailed description of the challenges within structural bioinformatics see the introductory chapter by Altman and Dugan [AD03]. A description of the closely related field of *structural genomics*, together with further references, can be found in the chapter by Burley and Bonanno [BB03].

Chapter 2

Biochemical Foundation

Perhaps the most remarkable features of the molecule are its complexity and lack of symmetry.

John Kendrew about the first protein structure that he determined in 1958

2.1 Biopolymers

Most of the biochemical macromolecules are *polymers*, that is, they consist of repeating units that are the residues of a linking reaction of *monomers*. Well-known examples for biopolymers are the *nucleic acids*, in particular *deoxyribonucleic acid* (*DNA*) and *ribonucleic acid* (*RNA*). Other important examples of biopolymers are *polypeptides* (or *proteins*, see below), and *polysaccharides* (made of monosaccharides, simple sugars like glucose).

While the building blocks of polymers in general can be arranged in a complex structure (e.g., glucose units of polysaccharides like starch or glycogen are arranged in a branched structure), the composition of many polymers exhibits a chain-like assembly, at least from the viewpoint of covalent bonds. This is particularly true for the cases of DNA, RNA, and polypeptides. We will discuss the advantages later that may arise from this linear setup.

2.2 Proteins

2.2.1 Amino Acids as Basic Modules of Proteins

The basic building blocks (monomers) of proteins are α-amino acids. These are comparatively small molecules that possess an amino group (NH_2) and a carboxyl group ($COOH$). The α indicates that both functional groups are attached to the same carbon atom (C_α). The other two valences of this carbon atom establish bonds to a hydrogen atom (H) as well as a variable side chain (R). For a schematic view of an amino acid see Figure 2.1.

In the case where the side chain is more than just a single hydrogen atom, the central carbon atom C_α is called *chiral*, since there are four different substituents and thus two

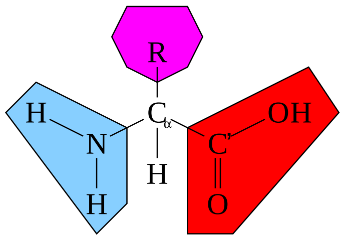


Figure 2.1: Schematic structure of amino acids.

possibilities of attaching the groups that cannot be superimposed. In fact, they are mirror images of each other (enantiomers, see Figure 2.3). In chemistry these two forms are traditionally denoted by D and L, or, according to legal naming conventions more correctly, R and S. As a matter of fact, proteins found in nature are almost entirely made of L-amino acids. The ultimate reason is unknown so far, but it is assumed that there is an evolutionary advantage if all amino acids share the same configuration, and it happened just by chance to be the L-form. In rare cases, D-amino acids were found in proteins, but these were due to direct enzymatic synthesis.

The set of amino acids that are regularly built into proteins comprises 20 different variations of the side chain. These so called *proteinogenic* amino acids are encoded for by DNA/RNA triplets of the standard genetic code. It is of prime importance to note, that the final structure of a protein is largely determined by the combination of the different amino acids and the characteristic properties of their side chains. These can be classified according to hydrophobicity, charge, aromaticity, and size, see Figure 2.2.

2.2.2 Historic Background

In the thirties of the past century, W. T. Astbury and coworkers showed in a series of experiments that the X-ray diffraction pattern of keratin contained in wool, feathers, and human hair changed under different conditions. They found mainly two different diffraction patterns which were referred to as α and β. These two states could be transformed into each other by suitable treatment, leading to the assumption that they are different conformers of the same molecule. Since the β-state was supposed to be more extended than the α-form, this provided an reasonable explanation of the elasticity of wool and human hair.

Although all the necessary information to propose a model for the structure of keratin was available with the data of these experiments, it was not until the year 1950, when Linus Pauling and Robert Corey [PC50, PCB51] proposed two variants of helical structures which were believed to be very stable. They had been derived from

1. an essential assumption of the general geometry of polypeptides that followed from Pauling's resonance theory of chemical bonds: planarity of the peptide group,

2. bond lengths and bond angles of related small molecules which had been measured with sufficient accuracy by X-ray diffraction.

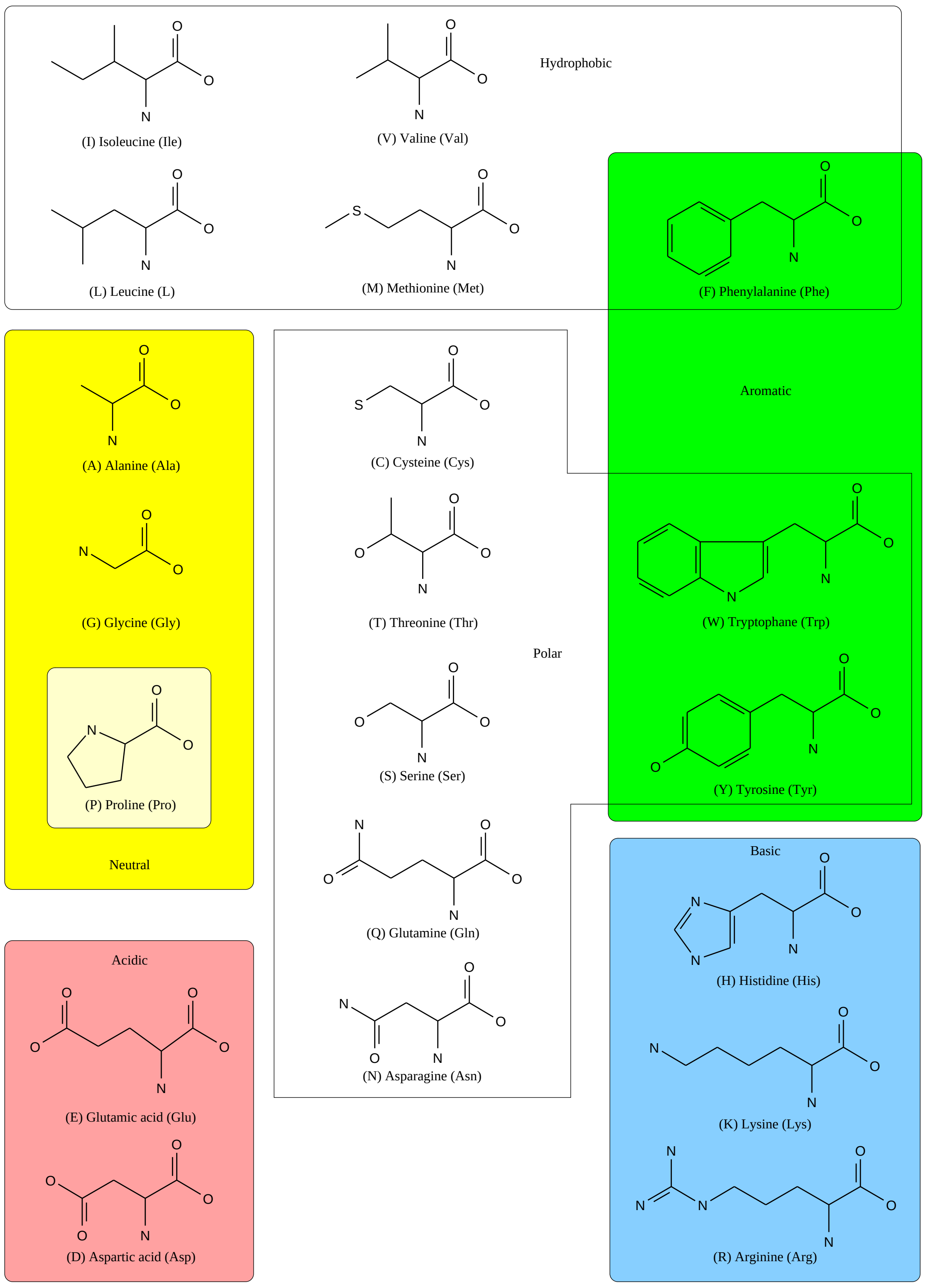
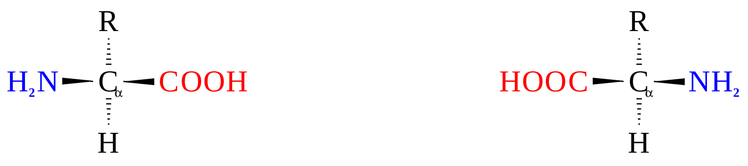


Figure 2.2: Classification of the 20 proteinogenic amino acids, roughly ordered top down by decreasing hydrophobicity.



(a) L-amino acid (b) D-amino acid

Figure 2.3: Enantiomers of an amino acid.

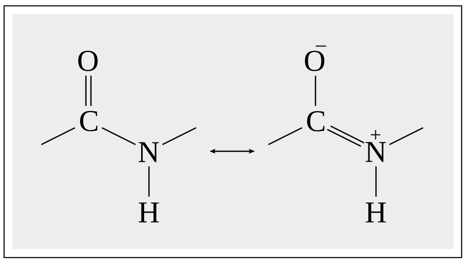
In particular, the prediction of one helix having 3.7 residues per turn and a unit translation per residue of 1.47Å was a major breakthrough. It was supposed to be contained in α-keratin, contracted myosin, hemoglobin, and other structures showing the α-diffraction pattern. Hence, it was subsequently named the *α-helix*. (On the other hand, the second predicted helix, the so called γ-helix, has virtually never been observed as of today. . .) The predicted structure was supported in 1951 by X-ray experiments of Max Perutz [Per51] and finally confirmed in 1957 by the first determination of a (low-resolution) three-dimensional protein structure (myoglobin) by John C. Kendrew et al. [KBD$^+$58]. Kendrew made a remarkable statement, probably disappointed of the not so regular structure, which was in contrast to the regular structure of the DNA double helix that had been determined some years before:

> "Perhaps the most remarkable features of the molecule are its complexity and lack of symmetry. The arrangement seems to be almost totally lacking in the kind of regularities which one instinctively anticipates, and it is more complicated than has been predicted by any theory of protein structure."

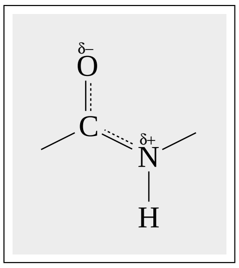
It should be noted that the figure in the paper of Pauling and Corey shows a left-handed α-helix made of D-amino acids, which is in contrast to the naturally occurring right-handed α-helices made of L-amino acids, that is, the figure shows the mirror image of the true structure. Since the γ-helix is depicted as a right-handed spiral, this indicates that the authors did not pay much attention to this seemingly negligible fact (see the review by Eisenberg [Eis03] for further information). In the following years, α-helices and β-sheets [PC51] turned out to be the most frequent patterns (by far) of all known protein structures.

2.2.3 Formation of the Backbone

Amino acids form linear polymers through a reaction of their respective amino and carboxylic functional groups. The resulting molecule is called a *peptide* (from Greek for 'digestible'), or, in the case of many amino acids, a *polypeptide*. The monomer parts are afterwards called *(amino acid) residues*. The new kind of bond between the C' and N atoms that is formed by releasing a water molecule is called the *peptide bond*. Due to the mesomeric effect, the π-electron pair of the $C = O$ double bond is delocalized over the

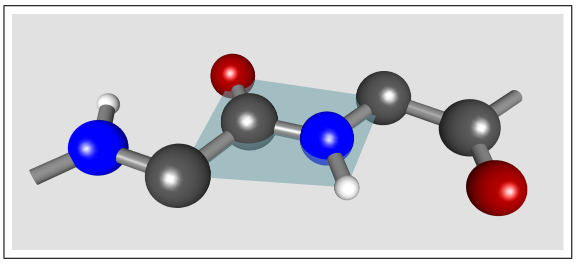


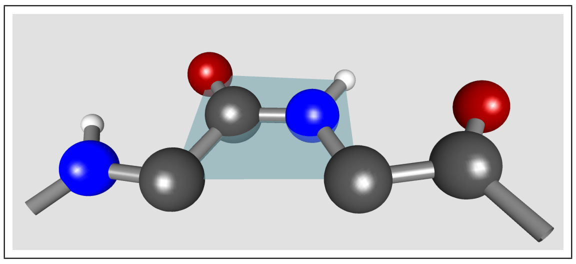
(a) Mesomeric structures.



(b) Delocalized π-bond.

Figure 2.4: The peptide bond.


(a) The *trans* conformation.


(b) The *cis* conformation.

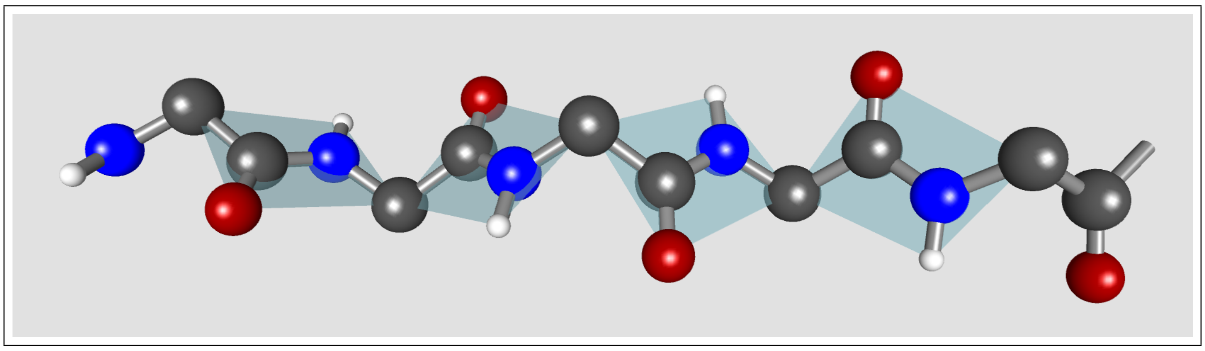

(c) The peptide planes intersect each other at the C_α atoms.

Figure 2.5: The plane of the peptide group.

Bond	Length
single $C - N$ bond	1.47Å
peptide $C = N$ bond	1.33Å
double $C = N$ bond	1.27Å

Table 2.1: Typical bond lengths of C and N. The length of the peptide (amide) bond is between the lengths of the single and the double bond.

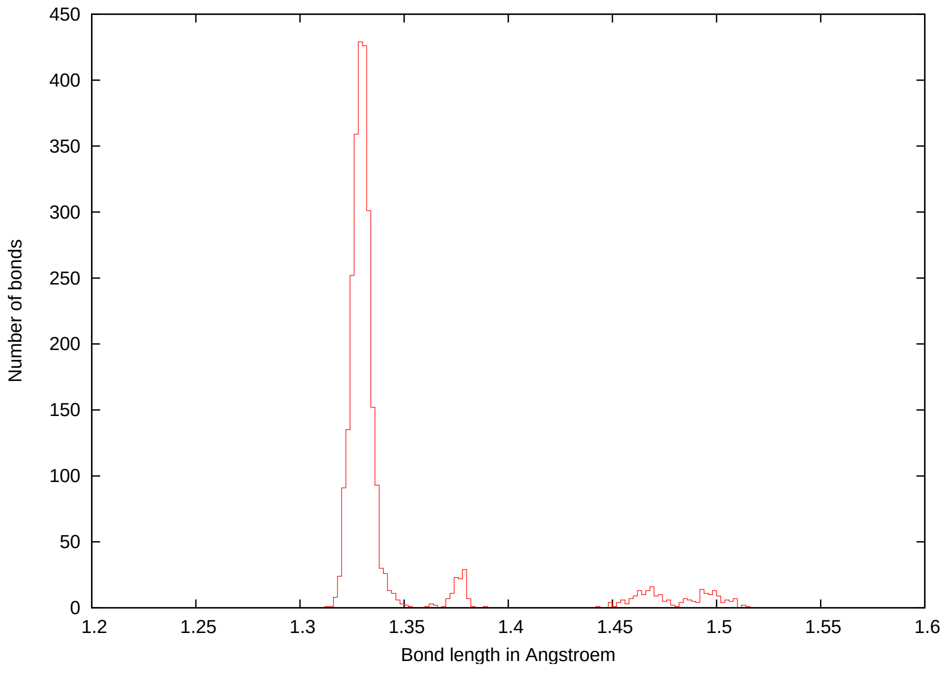


Figure 2.6: Sample bond length histogram between C and N atoms of the PDB file 1d2r.

$C' - N$ bond, which causes the peptide bond to have a partial double bond character (see Figure 2.4). Consequently, the peptide bond has a reduced bond length and, in contrast to a normal covalent $C - N$ bond, rotation around the bond is prevented by a strong energy barrier. This forces the participating atoms of each peptide unit into a coplanar position (which is mostly the *trans*-conformation where the neighboring C_α atoms reside on different sides, and rarely the *cis*-conformation where the C_α atoms are on the same side of the peptide bond). Thus, the variability of the *backbone* of the polypeptide, that is, the chain of the atoms $\cdots N - C_\alpha - C' \cdots$, is restricted to rotations around the $C_\alpha - C'$ and $N - C_\alpha$ bonds. These rotations can be measured formally by the torsion that is defined by the bonds of the two neighboring backbone atoms.

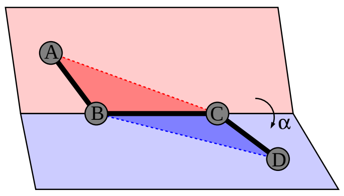
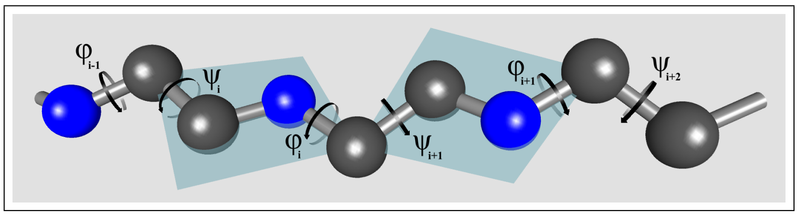
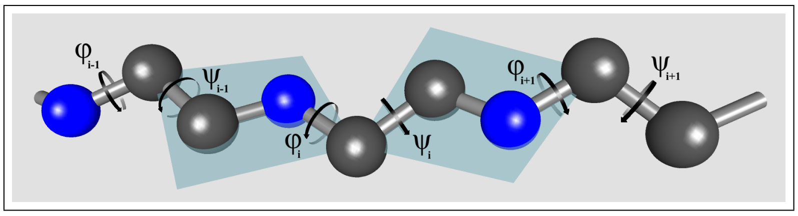


Figure 2.7: The general definition of torsion angles.



(a) Counting-based numbering scheme



(b) C_α-oriented numbering scheme

Figure 2.8: The backbone torsion angles ψ and φ.

2.2.4 Protein Structure

The overall structure of proteins is usually described in a 4-level hierarchy. Starting at the lowest level, the particular sequence of the different amino acids constitutes a proteins *primary structure*. Except for the concrete side chains (and their respective properties like size or charge), this sequence does not tell much about the actual three-dimensional structure and shape of the protein. Nevertheless, the particular side chains constrain the possible conformations of the backbone to a large degree. At the end, it is the amino acid sequence that predefines the final structure of the folded protein.

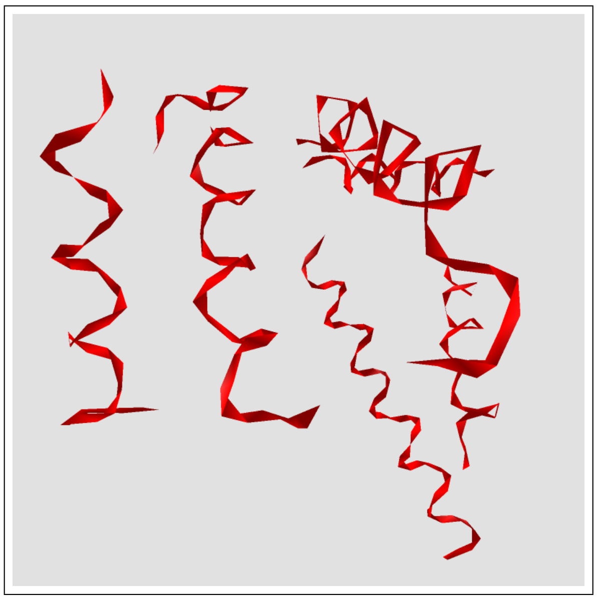
The *secondary structure* of a polypeptide describes the local conformation of parts that fold independently from the rest of the protein. This structure is mainly effectuated by hydrogen-bonding interactions between the carbonyl oxygen and the amide proton of the peptide bond atoms. Exceptionally stable (and thus frequently occurring) sub-structures are formed by regularities of the backbone conformations: the so-called *secondary structure elements*. The most prominent ones are the helices (most notably the α-helices) as well as the β-strands.

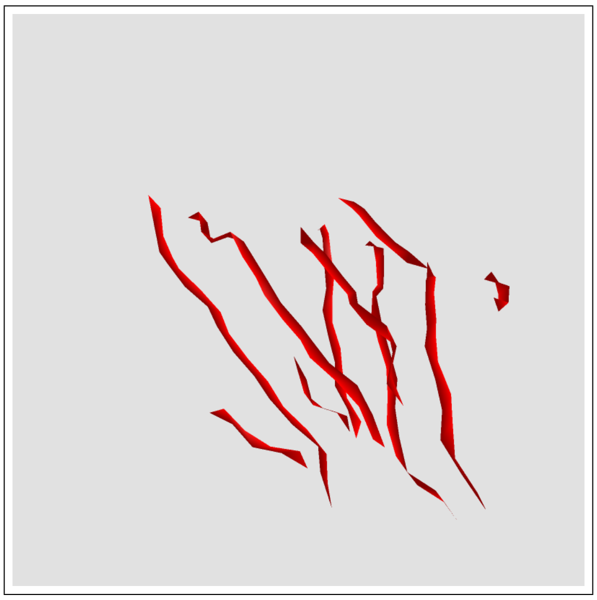
The term *tertiary structure* denotes the global three-dimensional conformation of one polypeptide chain. It describes how compact substructures (domains) are formed from several motifs, and how these domains are arranged within the molecule.

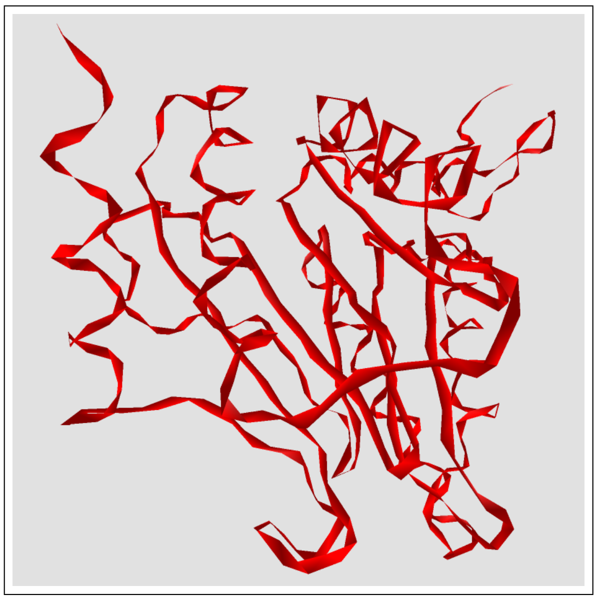
The spatial arrangement in which two or more polypeptides assemble a multimeric protein is referred to as the *quaternary structure*.

```
Gly Thr Gly Tyr Asp Leu Ser Asn Ser Val Phe Ser Pro Asp Gly Arg Asn Phe Gln Val Glu Tyr Ala Val Lys Ala Val Glu Asn Gly Thr Thr Ser Ile
Gly Ile Lys Cys Asn Asp Gly Val Val Phe Ala Val Glu Lys Leu Ile Thr Ser Lys Leu Leu Val Pro Gln Lys Asn Val Lys Ile Gln Val Val Asp Arg
His Ile Gly Cys Val Tyr Ser Gly Leu Ile Pro Asp Gly Arg His Leu Val Asn Arg Gly Arg Glu Glu Ala Ala Ser Phe Lys Lys Leu Tyr Lys Thr Pro
Ile Pro Ile Pro Ala Phe Ala Asp Arg Leu Gly Gln Tyr Val Gln Ala His Thr Leu Tyr Asn Ser Val Arg Pro Phe Gly Val Ser Thr Ile Phe Gly Gly
Val Asp Lys Asn Gly Ala His Leu Tyr Met Leu Glu Pro Ser Gly Ser Tyr Trp Gly Tyr Lys Gly Ala Ala Thr Gly Lys Gly Arg Gln Ser Ala Lys Ala
Glu Leu Glu Lys Leu Val Asp His His Pro Glu Gly Leu Ser Ala Arg Glu Ala Val Lys Gln Ala Ala Lys Ile Ile Tyr Leu Ala His Glu Asp Asn Lys
Glu Lys Asp Phe Glu Leu Glu Ile Ser Trp Cys Ser Leu Ser Glu Thr Asn Gly Leu His Lys Phe Val Lys Gly Asp Leu Leu Gln Glu Ala Ile Asp Phe
Ala Gln Lys Glu Ile Asn
```

(a) Primary structure.


(b) Secondary structure (helices).


(c) Secondary structure (sheets).


(d) Tertiary structure.

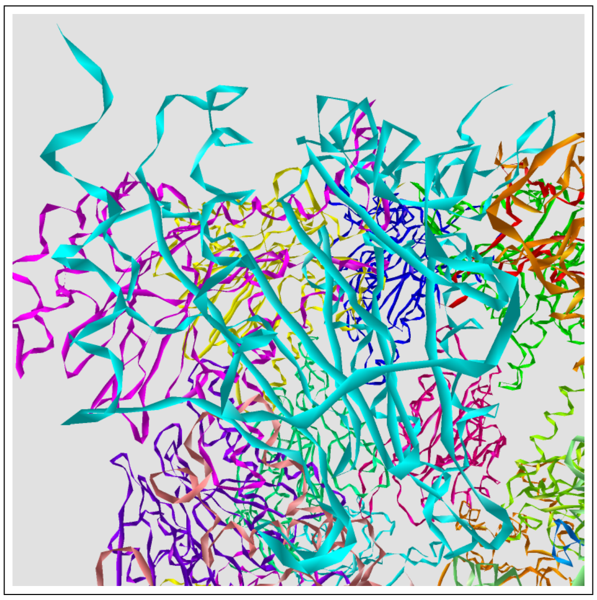
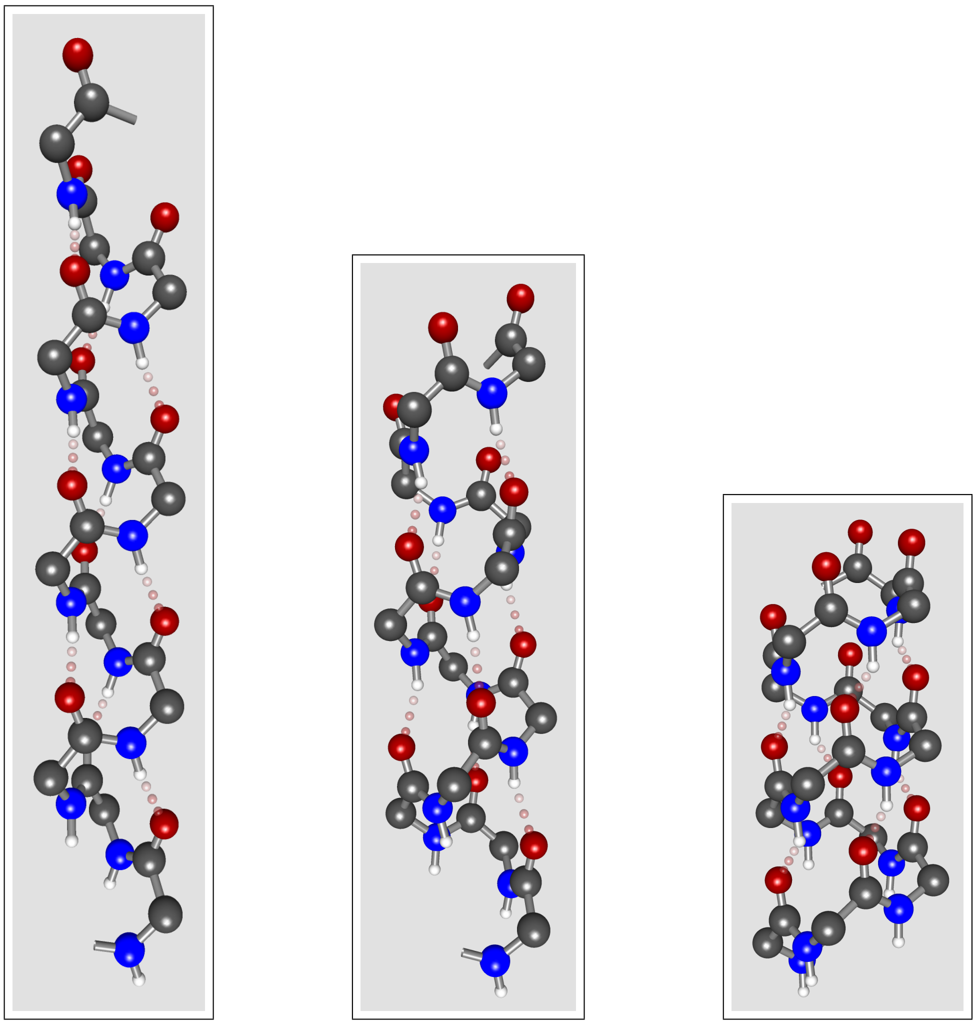
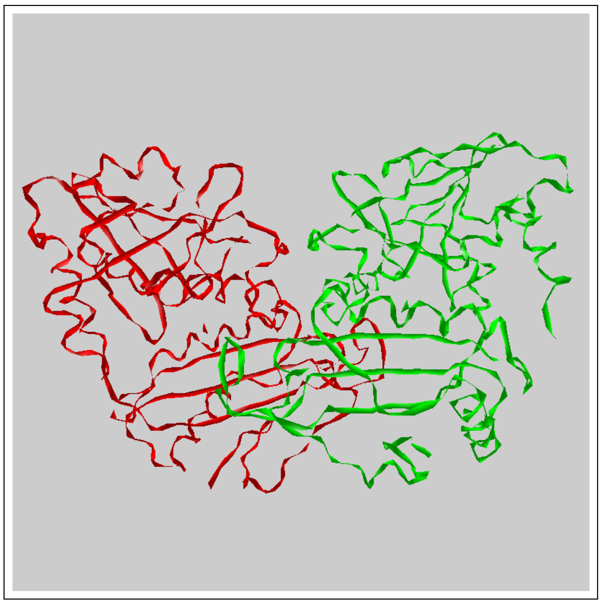

(e) Quaternary structure.

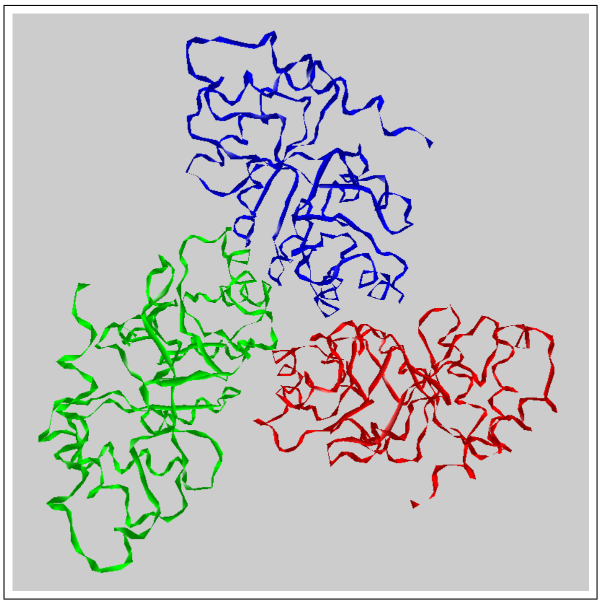
Figure 2.9: The protein structure hierarchy.

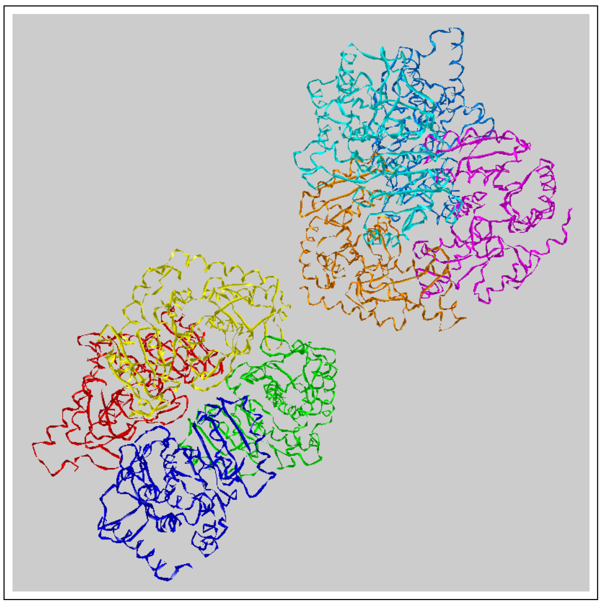


(a) 3_{10}-helix (b) α-helix (c) π-helix

Figure 2.10: Visual comparison of the different helix types. Note the different H-bonding schemes: $CO_{(i)} \cdots HN_{(i+3)}$ for the 3_{10}-helix, $CO_{(i)} \cdots HN_{(i+4)}$ for the α-helix, $CO_{(i)} \cdots HN_{(i+5)}$ for the π-helix.


(a) D-amino acid aminotransferase from thermophilic *Bacillus* sp. (PDB code `3daa`)


(b) KDPG aldolase from *Escherichia coli* (PDB code `1fq0`)


(c) L-lactate dehydrogenase from *Bacillus stearothermophilus* (PDB code `1ldn`)

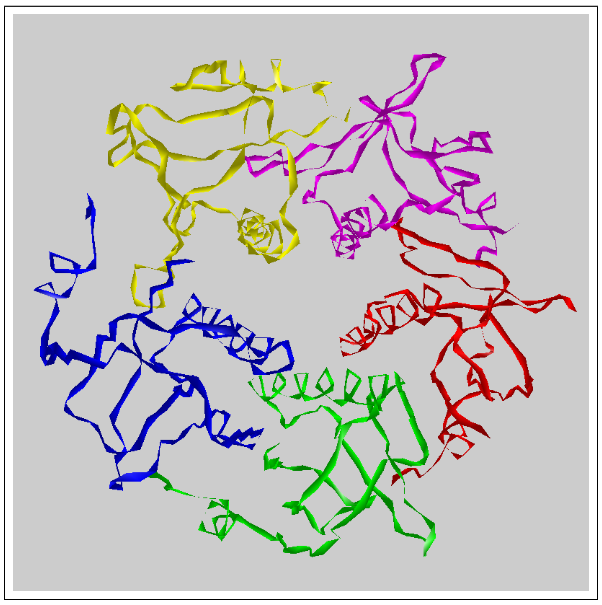
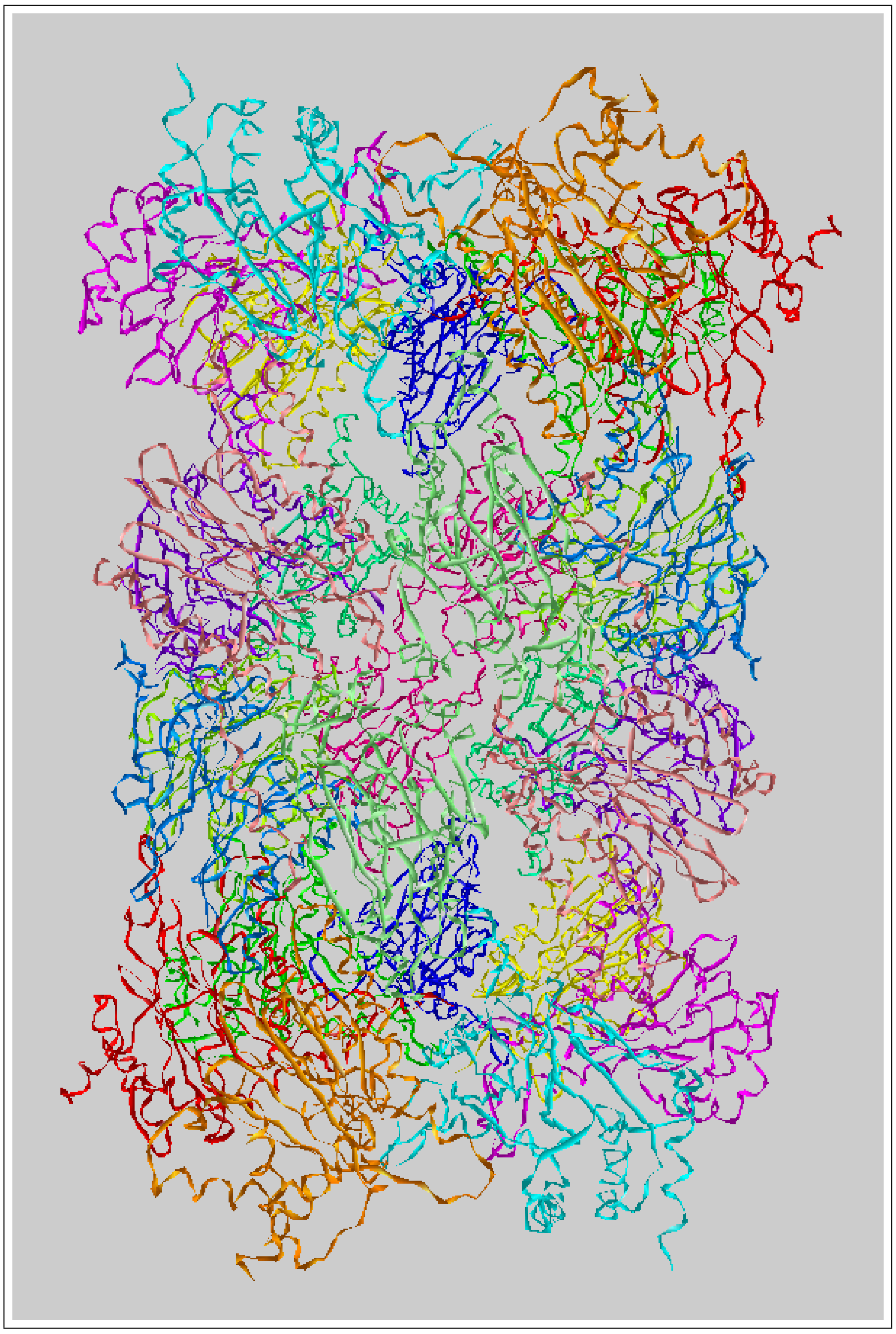

(d) Cholera toxin B from *Vibrio cholerae* (PDB code `1chp`)

Figure 2.11: Examples of quaternary structure (for further instances see [PR04d] p.45).



(a) Lateral view of the 20S core protease particle (PDB code 1g65) of the proteasome of *Saccharomyces cerevisiae*. Together with two 19S regulatory particles it is used for the digestion of ubiquitin-marked proteins into small peptides and amino acids. The hollow barrel consists of four heptameric rings of two different types. The outer rings (here at the top and bottom) consist of α- and the inner ones of β-subunits.

Figure 2.12: Examples of quaternary structure (continued).

Name	H-Bond $CO \cdots HN$	φ	ψ	α	Residues per turn	Rise [Å/res.]	Pitch [Å/turn]	Ref.
3_{10}	$i, i+3$	-49	-26	81	3.0	2.00	6.0	[RS68, Cre93]
α	$i, i+4$			50				[OH94]
		-57	-47	45	3.6	1.50	5.4	[RS68, Cre93]
		-65	-41	43				[Cho84]
π	$i, i+5$	-76	-41	30	4.4	1.2	5.3	[FAK02]
		-57	-70	17	4.4	1.15	5.0	[RS68, Cre93]

Table 2.2: Characteristics of different helices. We calculated approximate values (gray) for missing α-angles (for a formula, see [Lev76]), and for the pitch.

2.2.5 Secondary Structure Elements

Helices

Helices are spiral structures that are formed by regular repetitions of hydrogen bonds between the carboxyl and amide functional groups of the backbone, see Figure 2.10. In α-helices, the hydrogen bond is formed between the carboxyl oxygen of residue i and the amide hydrogen of residue $i+4$. Please note, that it makes a difference whether the bond is in the forward or backward direction along the polypeptide chain. Due to steric hindrance, most of the helices are right-handed. There are also helices, where the hydrogen bond is tied to the residue $i+3$ or $i+5$, which causes the coil to be more twisted (3_{10}-helix), or less (π-helix). For further information concerning other helix types, see[FAK02, NK05].

The geometric properties of helices are usually described by the average or most frequent φ and ψ angles that are usually observed in these structures. While it seems to be easy to give such a pair of angles for the abundant α-helices, it could be a problem for the rare cases of π, 3_{10}, as well as the left-handed helices. Consequently, there are different values given in the literature, but –most surprisingly– this also holds true for the α-helices. So, what might be the reason for these extremely differing values? For this question to be answered we have to consider

1. the admissibility of φ-ψ pairs and
2. the preference of certain conformations.

The first point can be thought of as trying all possible (φ, ψ) combinations for all different sequences consisting of a few amino acids. The regions where steric hindrance occurs between any atoms of either backbone or side chains (assuming standard geometry of the bond lengths and angles) are cut out from the φ-ψ (Ramachandran) plot. Allowed regions are marked. Additionally, in the transient parts between allowed and forbidden regions, there are zones that could occur by some relaxation of steric hindrance (small torsion or strain compared to standard bond geometry).

We believe, that the different measures of the typical helix parameters are in fact due to their variability. There are sufficiently large admissible regions in the Ramachandran plots that enable some flexibility of the backbone. The plots of the hypothetic length of H-bonds of the different types are shown in Figures 2.13 and 2.14. We can see that the maximum of the torsion angle deviations is located between the ideal values for α- and π-helices. Thus we assume the frequent existence of bifurcated three-center hydrogen bonds, which was also suggested by PREISSNER et al. [PES91].

Strands and Sheets

Other important regular secondary structures are the β-strands. Several of these almost fully extended chains are usually aligned near each other to form a β-sheet (also called *pleated sheet*) which is stabilized by hydrogen bonds between adjacent strands. The two types (parallel or antiparallel) differ in the distance of the neighboring chain segments. Sheets with mixed parallel and antiparallel interactions also occur frequently.

Turns and Loops

Turns and loops are irregular secondary structures. They occur at the surface of proteins and contain mostly polar residues. Short loops can be classified according to their structure, whereas longer loops exhibit a very variable shape. The most abundant loop type connects the ends of the strands of antiparallel β-sheets. It is referred to as β-turn or β-hairpin.

2.2.6 Supersecondary Structure, Motifs, and Domains

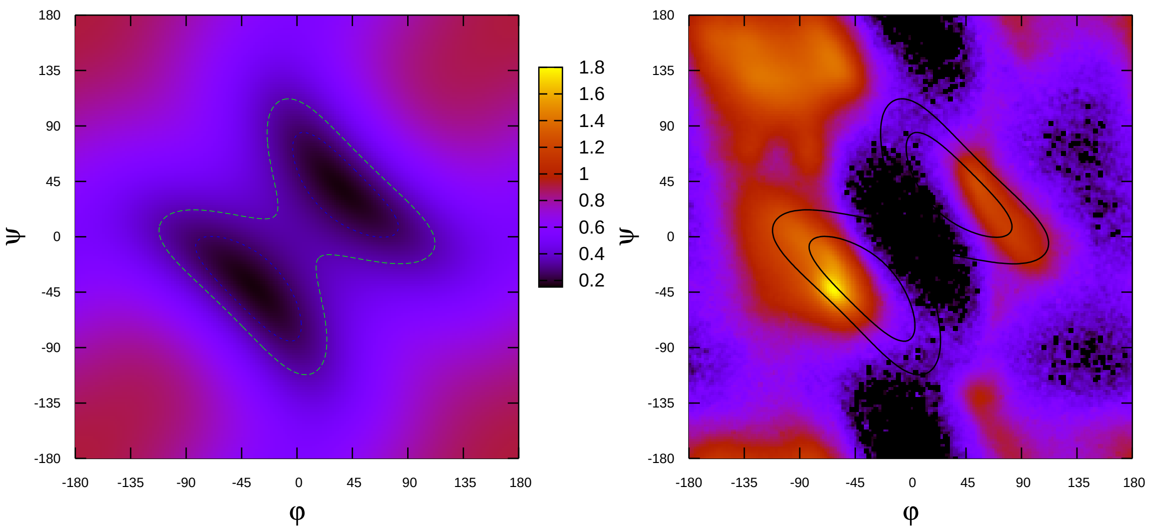
Motifs are Patterns of Supersecondary Structure

Several secondary structure elements that are contiguous within a chain build a more or less complex *supersecondary structure*. If this consecutive pattern occurs frequently in the database it is called a *(structural) motif*, in contrast to *sequence motifs* which are conserved patterns of the amino acid sequence. While sequence motifs are usually strong indicators of a particular function (therefore they are also called *functional motifs*), the presence of a structural motif may or may not be an indicator for similar function performed by the respective carriers.

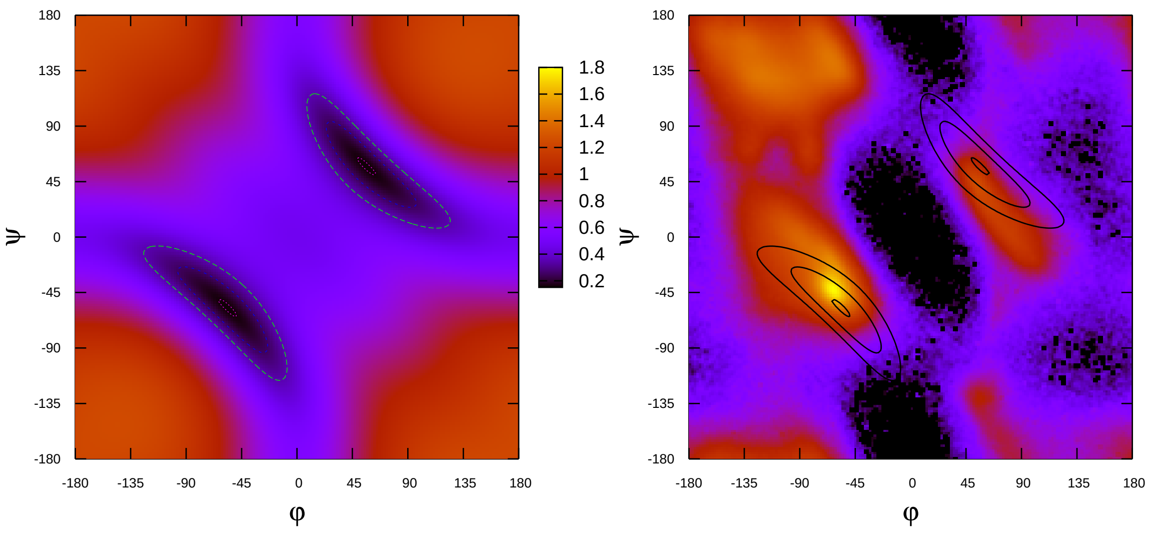
Examples of structural motifs are the beta hairpin, the Greek key, the helix-turn-helix / helix-loop-helix motif (in particular the DNA- and the calcium-binding motif), the zinc finger, the β-α-β motif as well as the jelly roll (that is made of several Greek key motifs).

Domains are Units of Function

While motifs are required to be arranged successively in the chain, there are also non-consecutive parts that assemble a particular structure: a *domain*. These parts of a polypeptide chain are often built from structural motifs and secondary structure elements. They usually fold independently from the rest of the protein into a stable tertiary structure that performs a highly specific function (q.v. [JC85, Ros85, WW03]).

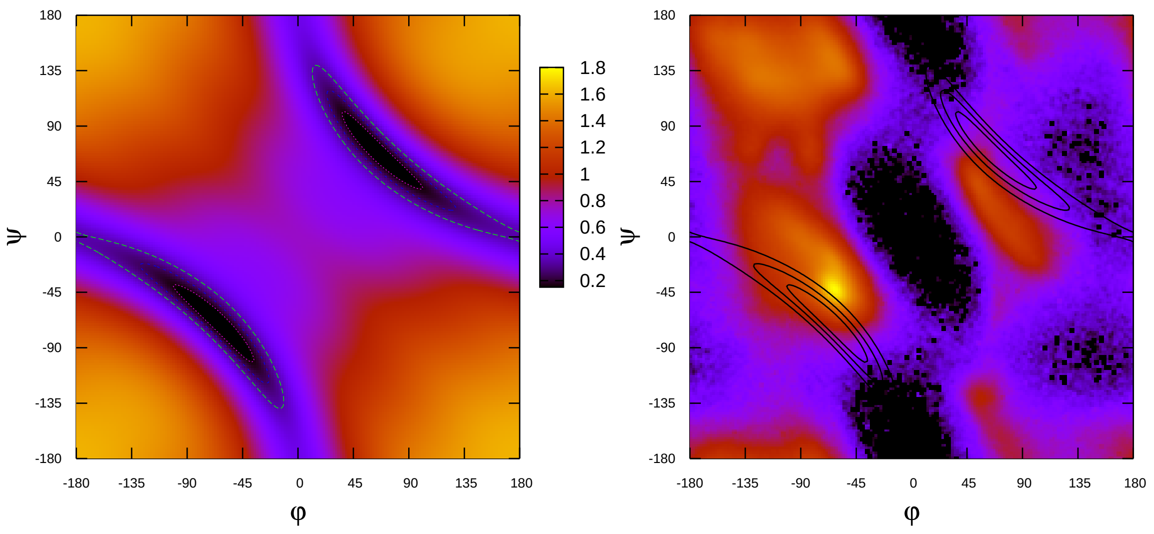


(a) Helix of H-bond type $(i, i+3)$, or 3_{10}-helix resp.

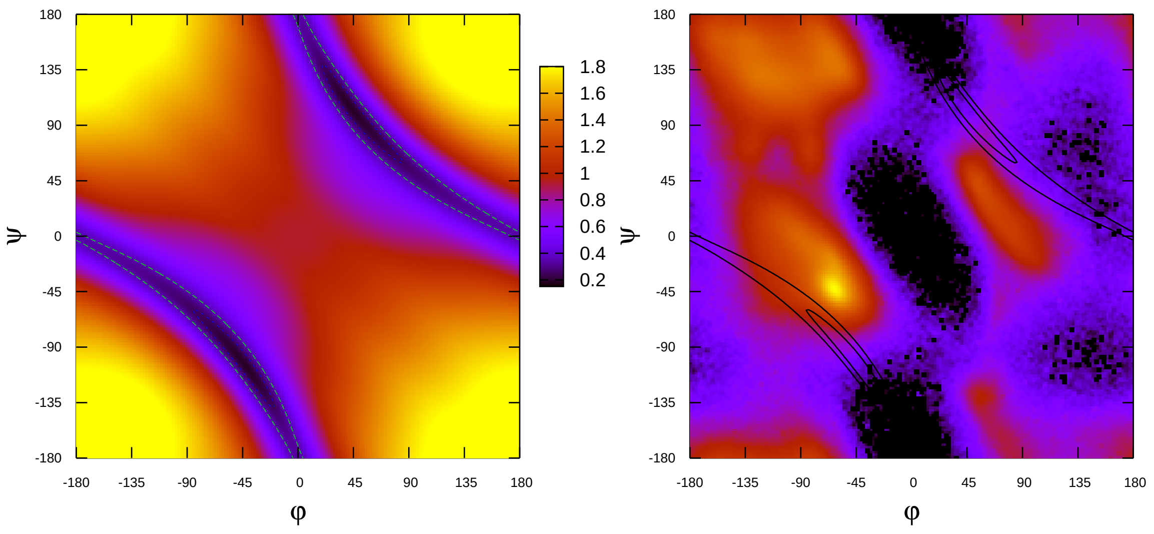


(b) Helix of H-bond type $(i, i+4)$, or α-helix resp.

Figure 2.13: The H-bond length of the 3_{10}- and the α-helix.



(a) Helix of H-bond type $(i, i+5)$, or π-helix resp.



(b) Helix of H-bond type $(i, i+6)$

Figure 2.14: The H-bond length of the π- and the $(i, i+6)$-helix.

2.2.7 Comment

The definitions of the described hierarchy together with the terms motif and domain are in fact a simplified view of the real nature. Of course, there cannot be such a strict separation of all these levels and terms. For instance, it is not exactly clear at which point to draw a distinction between secondary and supersecondary structure. How often must a structure occur to be called a motif. Why are β-sheets usually classified as secondary structure, although the strands may be interrupted, and thus may be non-consecutive in the chain. Why should β-strands be secondary structure elements while they do not fold independently, they need H-bonds to parallel or anti-parallel strands to be stable.

2.2.8 Further Reading

Further information on protein structure can be found (amongst others) in the richly illustrated primer of Petsko and Ringe [PR04d], the detailed textbook by Branden and Tooze [BT99], the standard work of Creighton [Cre93], the compact work of Darby and Creighton [DC93], as well as the chapter by Scheeff and Fink [SF03]. Topics regarding the physical properties of biochemical macromolecules are treated in the book of van Holde, Johnson, and Ho [vHJH98].

2.2.9 Protein Folding

The folding of proteins occurs according to the minimization of free energy. For globular proteins, this means to bring all polar residues to the surface, all hydrophobic residues into the interior of the molecule. This process is referred to as the *hydrophobic collapse*. Membrane proteins follow other rules of folding due to their special environment. The comparatively fast folding of proteins seems to contradict the hard computational problem of predicting the structure for a protein of known sequence, which is known as Levinthal*'s Paradox*.

2.2.10 Protein Functions

Proteins adopt a rich variety of different shapes, and it is exactly this diversity in three-dimensional structure that makes them suitable to fulfill specific functions. In the context of evolution, this leads to a conservation not only of the sequence of amino acids, but particularly of the structure within functionally active parts of proteins.

2.3 Nucleic Acids and the Central Dogma

2.3.1 DNA and RNA

At the end of the 19th century, a substance had been isolated from the nuclei of cells, which was subsequently called 'nuclein' (by F. Miescher) and later *'nucleic acid'* (by

Function	Description	Example(s)
Enzymes	catalyze biochemical reactions	lactase (cleavage of lactose), amylase (digestion of polysaccharides)
Transport proteins	carry substances	hemoglobin carries oxygen
Storage proteins	retain energy providers	ovalbumin (white of egg), ferretin, casein
Contraction proteins	facilitate motion by muscles	actin, myosin (muscles)
Structure proteins	provide a scaffold for stability	keratin (hair, feathers), collagen (scar tissue, tendons, bones)
Antibodies	enable the response reaction of the immune system	IgG (binding of pathogens like viruses or bacteria)
Hormones	regulate biochemical processes of an organism	insulin (regulation of carbohydrate metabolism)
Transcription factors	regulate the transcription of specific genes	STAT (Signal Transducers and Activator of Transcription, regulate cell growth)
Receptor proteins	specific binding to neurotransmitters or hormones	5-HT receptors (for serotonin binding)

Table 2.3: The main functions of proteins.

R. Altmann). It became evident, that there were two different forms, one containing ribose, the other one containing deoxyribose. This led to the well known names *ribonucleic acid* and *deoxyribonucleic acid*, or for short *RNA* and *DNA*.

It was widely believed that proteins carry the hereditary information (genes) from one population to the next. Today, it is a matter of common knowledge, that DNA carries the genetic code of all organisms and many viruses. But first evident experimental indication to this essential fact was given not until the forties of the 20th century by Avery et al. [AMM44]. This perception was finally proved through a series of experiments conducted by A. Hershey and M. Chase in 1952 [HC52].

In 1953, the helix structure of DNA was proposed in the celebrated publication of James Watson and Francis Crick [WC53]. However, it should be noted that there work was heavily based on discoveries that were made by Maurice Wilkins and Rosalind Franklin.

In September of 1957, Crick presented a summary of his ideas regarding the genetic code at the Symposium of the Society for Experimental Biology and postulated the *Sequence Hypothesis* and the *Central Dogma* [Cri58, Cri70], which is considered a seminal contribution to the development of Molecular Biology.

The Sequence Hypothesis "... assumes that the specificity of a piece of nucleic acid is expressed solely by the sequence of its bases, and that this sequence is a (simple) code for the amino acid sequence of a particular protein." [Cri58]

The Central Dogma

> "... states that once 'information' has passed into protein *it cannot get out again.* In more detail, the transfer of information from nucleic acid to nucleic acid, or from nucleic acid to protein may be possible, but transfer from protein to protein, or from protein to nucleic acid is impossible. Information means here the *precise* determination of sequence, either of bases in the nucleic acid or of amino acid residues in the protein." [Cri58]

Unfortunately, it is today's perception of the Central Dogma simply stating that DNA is commonly processed into RNA which is then used to make proteins. As has been pointed out by Crick [Cri70] this (positive) statement is a rather specified version of the Sequence Hypothesis and not the meaning of the Central Dogma which is an essentially negative proposition. It says that information flow does *not* occur in the direction from protein to DNA, RNA, or protein itself.

2.3.2 Protein Biosynthesis

The process of protein (bio)synthesis starts by binding an enzyme (RNA polymerase) to a specific marker position (the *promoter*) on the DNA which initiates the transcription phase by unwinding the two complementary strands. RNA polymerase slides along the template strand and creates a complementary RNA chain by successive elongation. In eukaryotes, the transcribed mRNA has to be processed by cutting out the introns and by splicing the exons; then the processed mRNA has to pass the membrane of the cell nucleus. The mature mRNA is protected by a 5' cap against degradation; in the case of eukaryotes also a poly-A tail is added at the 3' end. In the cytoplasm, the base triplets of the mRNA are translated into a sequence of amino acids, mediated by ribosomes and tRNA molecules.

		2nd base					
		U	C	A	G		
	U	Phe	Ser	Tyr	Cys	U	
		Phe	Ser	Tyr	Cys	C	
		Leu	Ser	Stop	Stop	A	
		Leu	Ser	Stop	Trp	G	
	C	Leu	Pro	His	Arg	U	
		Leu	Pro	His	Arg	C	
		Leu	Pro	Gln	Arg	A	
1st		Leu	Pro	Gln	Arg	G	3rd
base	A	Ile	Thr	Asn	Ser	U	base
		Ile	Thr	Asn	Ser	C	
		Ile	Thr	Lys	Arg	A	
		Met	Thr	Lys	Arg	G	
	G	Val	Ala	Asp	Gly	U	
		Val	Ala	Asp	Gly	C	
		Val	Ala	Glu	Gly	A	
		Val	Ala	Glu	Gly	G	

Table 2.4: The genetic code that translates a codon into an amino acid.

2.3.3 DNA and RNA Structure

DNA encodes the genetic information by a sequence of *nucleotides*, each one consisting of the sugar 2-deoxyribose, a phosphate, and one of the four *bases* adenine, thymine, guanine, and cytosine, abbreviated by the letters A, T, C, and G (see Figures 2.15, 2.16 and 2.17). The nucleotides are arranged in two complementary anti-parallel strands forming a double-helix, such that each adenine of one strand is paired with a thymine of the other strand, just as each cytosine is paired with a guanine. In ribonucleic acid (RNA), the nucleotide thymine is replaced by uracil.

A comprehensive introduction into DNA and RNA structure can be found in the chapter by Neidle, Schneider, and Berman [NSB03]. Geometric parameters in nucleic acids were studied in [GSC$^+$96].

2.4 The Protein Data Bank

The Protein Data Bank (PDB) was established in 1971 to hold the set of available structures of biological macromolecules. It was held by the Brookhaven National Laboratory

Type	Full Name	Description
mRNA	messenger RNA	After transcription from a DNA template part, mRNA carries the genetic information to the ribosome for protein synthesis.
ncRNA / sRNA / (s)nmRNA / fRNA	non-coding RNA / small RNA / (small) non-messenger RNA / functional RNA	A type of RNA that is not translated into a protein. Most prominent examples are rRNA and tRNA.
rRNA	ribosomal RNA	Central component of the ribosome.
tRNA	transfer RNA	Carries a specific amino acid to a growing peptide chain.

Table 2.5: The different types of RNA.

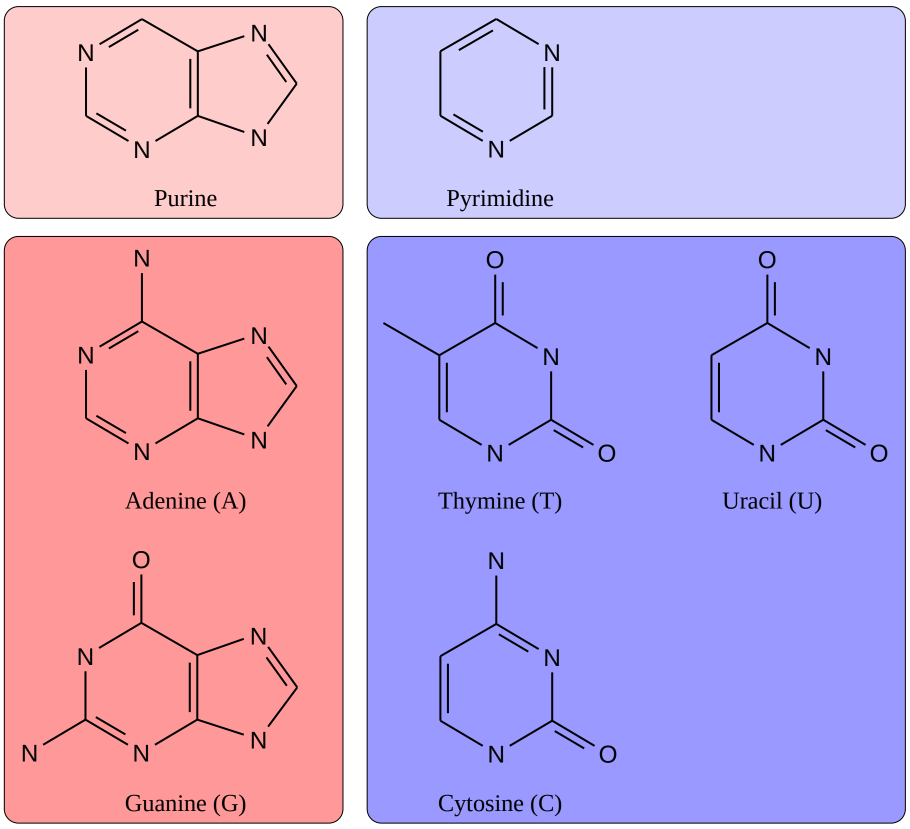


Figure 2.15: The DNA- and RNA-bases. Adenine and guanine are derived from purine; cytosine, thymine, and uracil are derived from pyrimidine.

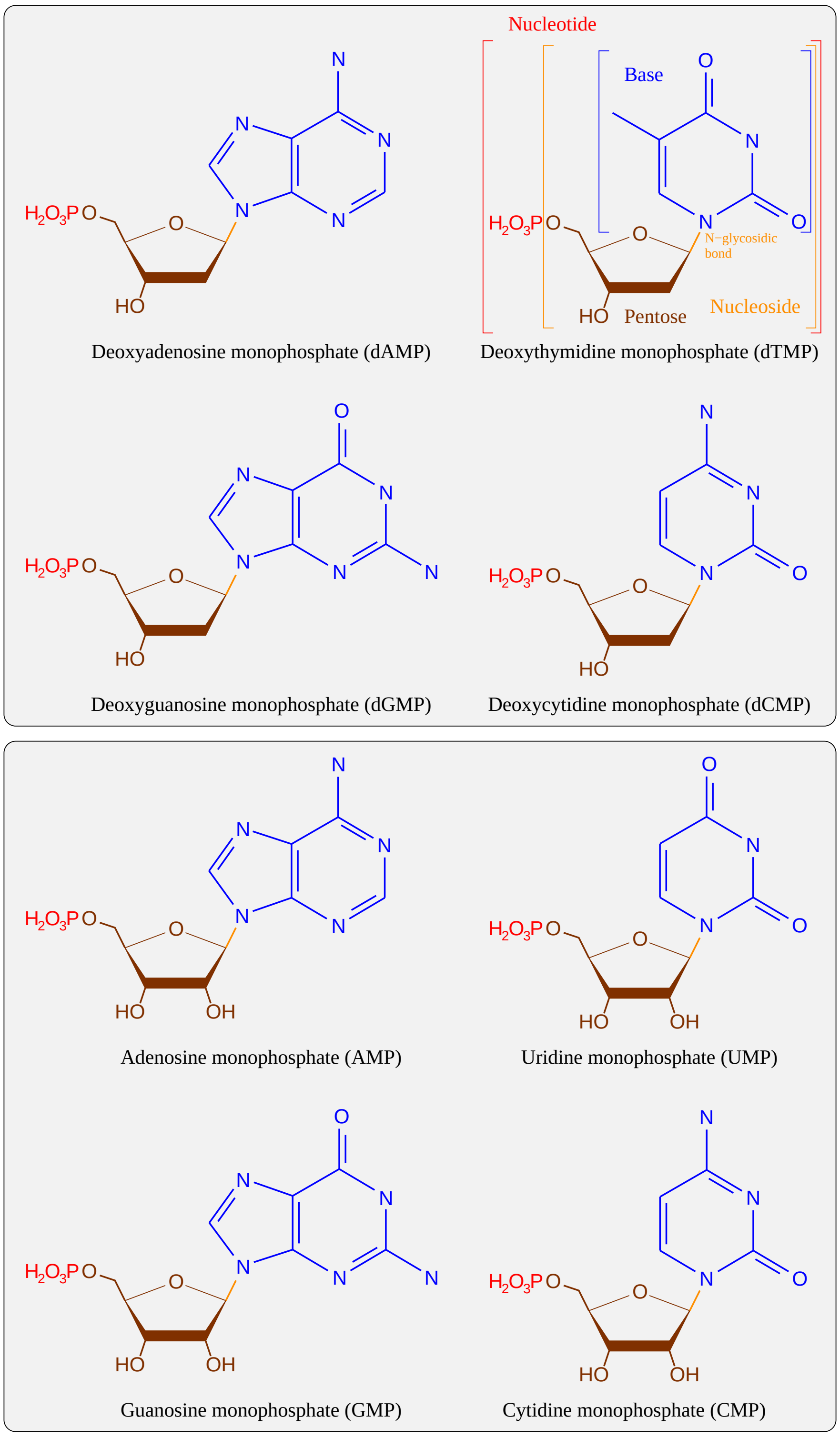

Nucleotide
Base
N-glycosidic bond
Pentose
Nucleoside
Deoxyadenosine monophosphate (dAMP)
Deoxythymidine monophosphate (dTMP)
Deoxyguanosine monophosphate (dGMP)
Deoxycytidine monophosphate (dCMP)
Adenosine monophosphate (AMP)
Uridine monophosphate (UMP)
Guanosine monophosphate (GMP)
Cytidine monophosphate (CMP)

Figure 2.16: The nucleotides of DNA and RNA are made of one of the heterocyclic bases, a pentose sugar (ribose for RNA, deoxyribose for DNA), and a phosphate group.

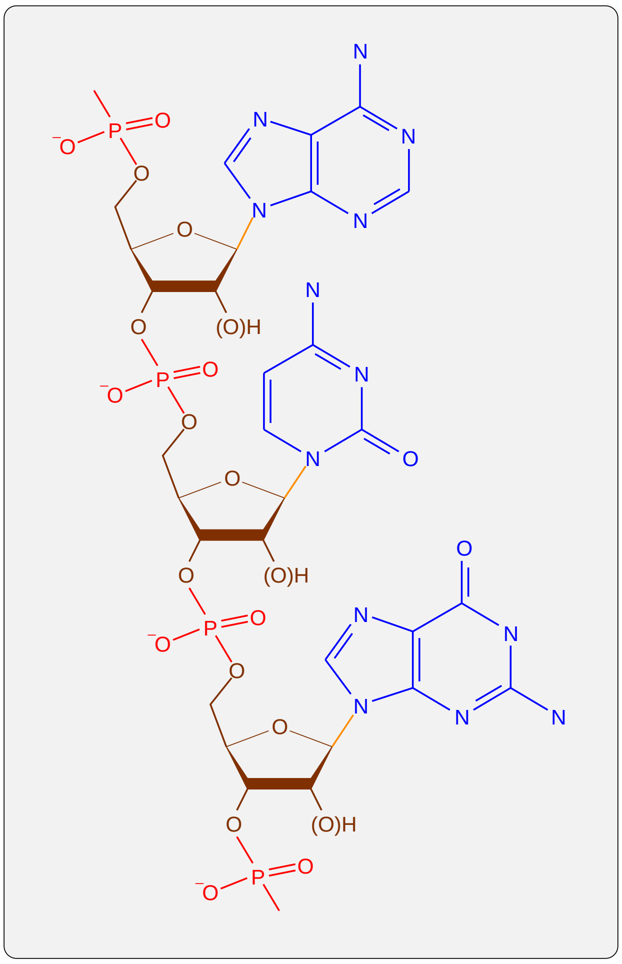


Figure 2.17: The principal structure of DNA and RNA.

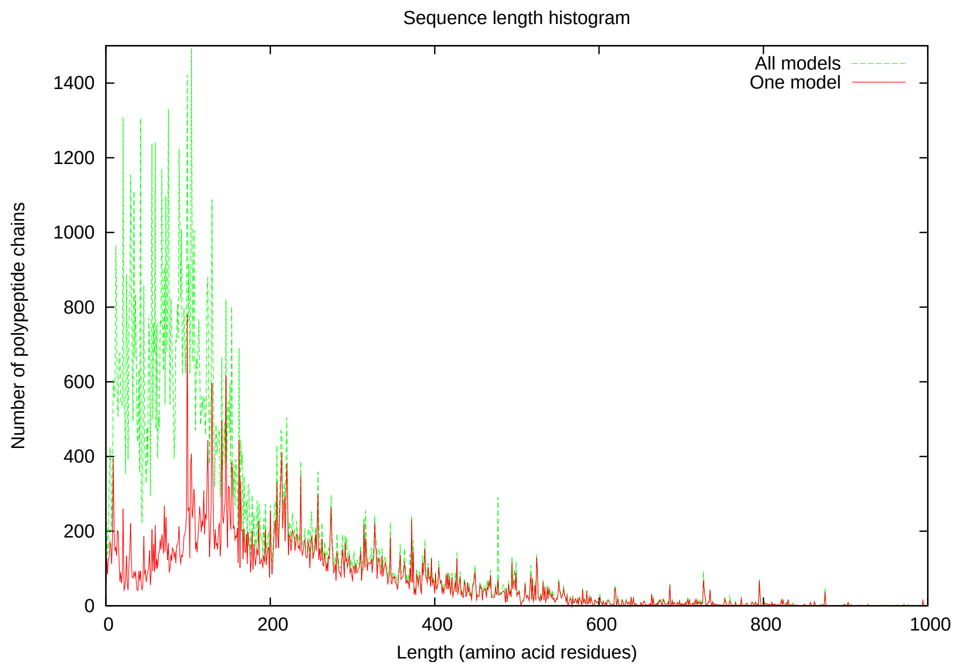
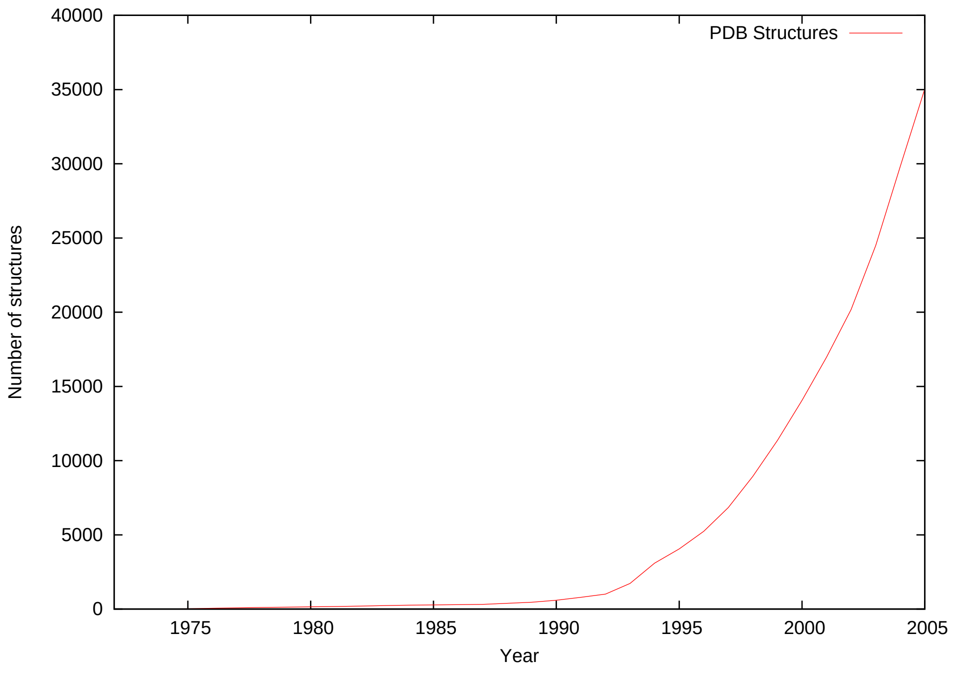


Figure 2.18: Length histogram of the Protein Data Bank.

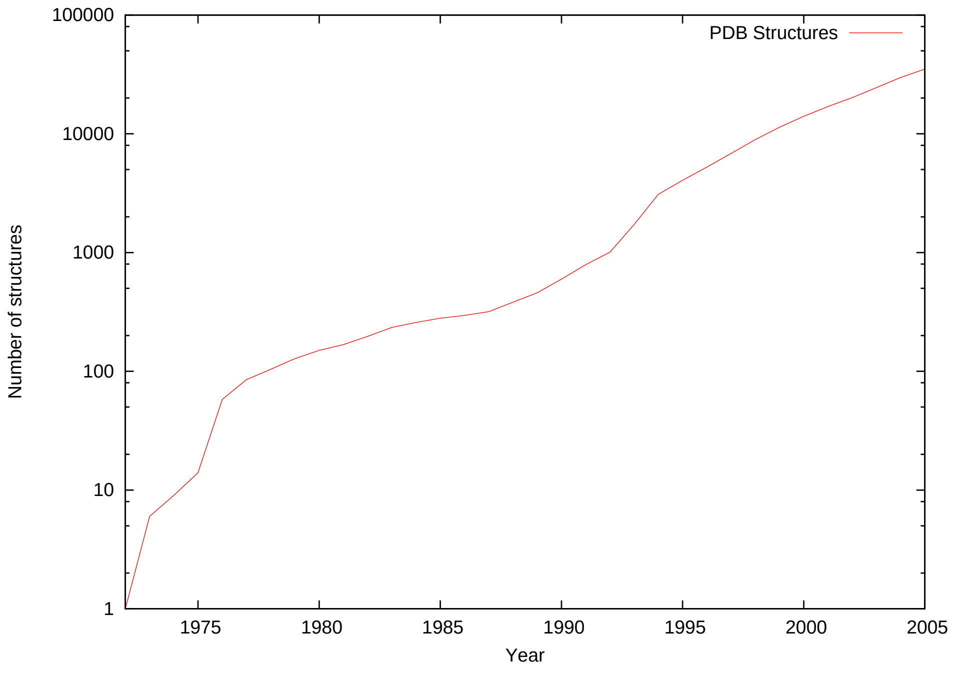
(BNL) until 1999 when responsibility moved to the Research Collaboratory for Structural Bioinformatics (RCSB), a consortium composed of Rutgers, The State University of New Jersey, the San Diego Supercomputer Center at the University of California, San Diego, and the National Institute of Standards and Technology (NIST). Today, it is the largest repository for storing data related to molecular biology. In particular, the PDB holds the three-dimensional structures of proteins and nucleic acids, that were determined by X-ray crystallography or NMR spectroscopy. A few hundred theoretical models were deposited too. Further information can be found in [ASPM97, BBM$^+$97, BWF$^+$00, PDB03].

Since the PDB was established by molecular biologists and biochemists, its initial state was more that of a *collection of text files* containing all the coordinates and structural information. It did not follow a computer scientist's guidelines for using modern *database systems*, such as bringing the tables into certain normal forms for saving space, minimizing redundancy, and assuring data integrity, as well as providing methods for fast access, e.g. via indexing structures. In the first years, these considerations were not so important since the PDB contained only a few structures and people working with it were aware of nearly all aspects of its concise contents. But a serious problem appeared with the explosive growth of this database (which seems to be exponential, see Figure 2.19).

For information regarding other structural databases, such as the Nucleic Acid Database (NDB), see [TG89, BOB$^+$92, GPK00, BWF$^+$03, WB03].



(a) Linear scale



(b) Logarithmic scale

Figure 2.19: Growth of the Protein Data Bank.

Chapter 3

Algorithmic Foundation

If a tree has leaves, what does a trie have?

Exercise 6.3.1 in Donald E. Knuth's "The Art of Computer Programming" (Vol.3: Sorting and Searching)

3.1 Pattern Matching

The primary task in the area of pattern matching is to perform a certain kind of *search* within a given *arrangement of symbols*. More formally, the various problems are stated as follows.

Throughout this thesis, we will assume a linear composition, that is a *sequence*, of the symbols. (Two-dimensional, three-dimensional, and d-dimensional arrangements have been studied in the literature too, but seem to be much more difficult to handle than the first case.) The linear sequence is commonly called a *string*, whereas the underlying symbols are called the *characters* of the string.

Let Σ be a set of characters which is called the *alphabet*. Usually, it is assumed that Σ is a *finite* set and the only operations defined for its entities are the *comparison operators* which state whether two character variables x and y hold the same value or not (e.g., for $\Sigma = \{\text{'a'},\ \text{'b'}\}$, $x = y$ holds for the assignments $\{x = \text{'a'},\ y = \text{'a'}\}$ and $\{x = \text{'b'},\ y = \text{'b'}\}$, whereas $x \neq y$ holds for the assignments $\{x = \text{'a'},\ y = \text{'b'}\}$ and $\{x = \text{'b'},\ y = \text{'a'}\}$). Sometimes, the alphabet is allowed to be infinite, and, as we will see, it makes sense to consider also approximate comparison operators based on similarity or distance functions (rather than absolute identity) which rate pairs of characters to be more equal than others.

We denote by Σ^ℓ the set of all strings of exactly ℓ characters from the alphabet Σ:

$$\Sigma^\ell = \{x_1 \cdots x_\ell :\ \forall i \in [1, \ell]\ x_i \in \Sigma\}$$

Please note that, within program code or pseudocode, we usually use the bracket notation and assume the index to start with zero:

$$\Sigma^\ell = \{x[0] \cdots x[\ell - 1]\ :\ \forall i \in [0, \ell - 1] \quad x[i] \in \Sigma\}$$

The length of a (finite) string s, that is the cardinality of its constituting multi-set of characters, is denoted by $|s|$. The string having length zero is represented by ε. By Σ^+ we denote the union of all strings over alphabet Σ having an arbitrary (finite) *strictly positive* length. Similarly, Σ^* is defined as the set of all strings of finite length:

$$\begin{aligned} \Sigma^+ &= \bigcup_{\ell=1}^{\infty} \Sigma^\ell \\ \Sigma^* &= \bigcup_{\ell=0}^{\infty} \Sigma^\ell = \Sigma^+ \cup \{\varepsilon\} \end{aligned}$$

We shall also consider sets of strings, for instance $D = \{s_1, \ldots, s_k\}$. We denote the number of strings by $|D| = k$, while the overall size of the dictionary or the database content, that is the sum of all string lengths, is denoted by

$$\|D\| = \sum_{i=1}^{k} |s_i|$$

Please note, that there is a difference between a set of strings $s_1 = \{\varepsilon\}$ that consists only of the empty string and an empty set of strings $s_2 = \{\}$ that contains no string. The cardinality of both sets differs since $|s_1| = 1$ and $|s_2| = 0$, although $\|s_1\| = \|s_2\| = 0$.

To simplify matters, we introduce the following terms: a consecutive part $t_i \cdots t_j$ of the text is called a *subword* of t. A subword $t_i \cdots t_n$ that extends to the end of the text is called a *suffix* of t, denoted by $\mathbf{suf}_i(t)$. Accordingly, a subword $t_1 \cdots t_i$ that ranges to the first character of the text is called a *prefix* of t, denoted by $\mathbf{pre}_i(t)$. The degenerate case of $\mathbf{suf}_{n+1}$ and $\mathbf{pre}_0$ represent the empty string (ε). A suffix (resp. prefix) of the text t is called *proper* if it is not equal to t. (This is the common definition of 'proper' in this context. May be, it would be more convenient to exclude the empty case too.)

Let $p = p_1 \cdots p_m \in \Sigma^+$ and $t = t_1 \cdots t_n \in \Sigma^+$ be (finite) sequences of characters from this alphabet. The problem to be solved is the question whether the pattern p of length m is a subword of the *text* t of length n, that is, is there a position i in t, such that $\forall k : t_{i+k} = p_k$. An extended version of this problem could ask for all such positions, which we will call *occurrences* of p in t.

Within the first decades of the development of pattern matching theory there have been quite a lot of ideas to approach this central problem, the most prominent ones being the algorithm of Knuth, Morris, and Pratt, the algorithm of Boyer and Moore, the algorithm of Aho and Corasick, as well as the algorithm of Karp and Rabin.

Two general ideas seem to be central concepts within this theory. The first one is the usage of a *preprocessing* step that precedes and simplifies the actual search step, allowing in particular a fast repetition of the search process with a slightly changed setting. The other idea is the usage of *tree structures* to make decisions during the search. Computer science has witnessed the development of numerous variants, such as binary search trees, red-black-trees, AVL-trees, B-trees, (a, b)-trees, and so on (see, e.g., [Knu98, CLRS01]).

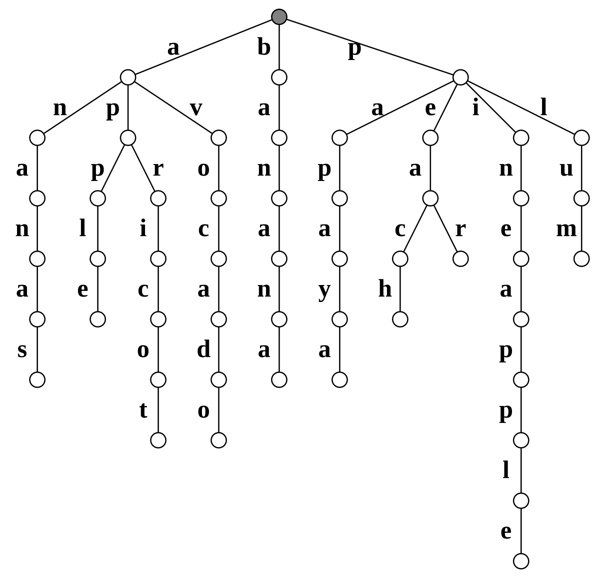


Figure 3.1: The atomic tree (trie) for the words `ananas`, `apple`, `apricot`, `avocado`, `banana`, `papaya`, `peach`, `pear`, `pineapple`, and `plum`.

While these structures are usually used by applying comparisons to the keys as a whole (e.g., numbers or strings), it is also possible to take advantage of the piecewise assembly of each key and to perform the comparison step-by-step by comparing the constituting parts (bits or characters) of the keys (which is called 'digital searching', see [Knu98] for an overview of the methods). Some of the more intricate structures apply the ideas of preprocessing and digital searching using trees to build a so called *index* that provides an accelerated search process. This approach will be discussed in more detail in the sections below.

3.2 Atomic Σ^+-Trees (Tries)

The fundamental structure we utilize in the following is the *tree* having a dedicated root node (or short *rooted tree*). In this context, we assume the usual denotation: For each node, there is exactly one path that connects it to the root node. The edges of this path are considered to point along the directed path from the root to the actual node. The source node is called a *parent* or *ancestor* of the target node. Accordingly, the target node is called a *child* or *successor* of the source node.

A Σ^+*-tree* is a rooted tree that satisfies the following conditions: Each edge connecting two nodes must be labeled with a (non-empty) string over the alphabet Σ. For each node, the edges to its children must start with *different* characters, that is, if u is a Σ^+-tree node with children v and w, then for the labels x and y of the edges $u \to v$ and $u \to w$ always $x_1 \neq y_1$ holds true.

A *trie* is a Σ^+-tree that satisfies the condition that each edge label consists of exactly

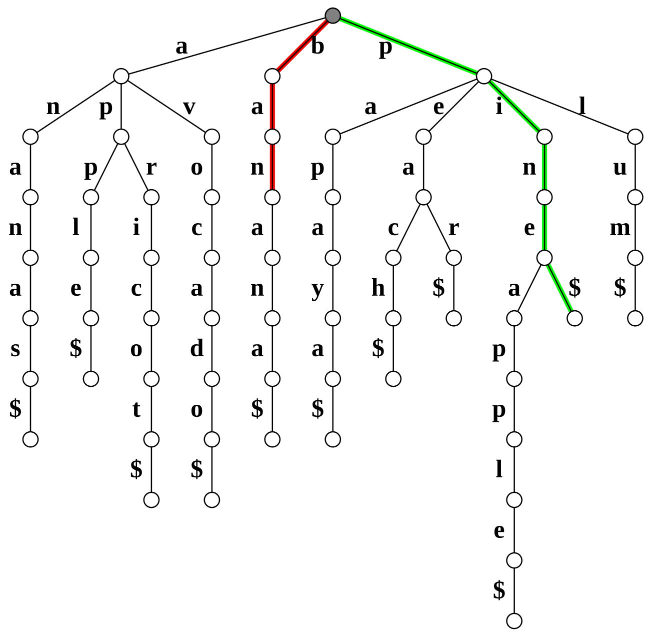


Figure 3.2: The atomic tree (trie) with sentinel. The string `pine` is contained in the database, whereas the string `ban` is not.

one character. Consequently, a trie is also called *atomic* Σ^+*-tree*.

The concept is based on a proposal of DE LA BRIANDAIS [dlB59]. The name “trie” is derived from the term re**trie**val, see FREDKIN [Fre60]. Such a trie can easily be used as a *dictionary*. To build the trie, the (non-empty) words are successively added by matching the beginning of the word going downwards the current trie. If there is no matching edge leaving the current node, then a new branch is added containing edges labeled with the remaining characters of the word.

After the trie has been built, it is possible to answer queries that ask whether a given word $w = w_1 \cdots w_m$ is in the set of (non-empty) strings $D = \{s_1, \ldots, s_k\}$ which are contained in the trie, that is, if there is an $i \in [1, k]$ such that $w = s_i$. The method works similar to the construction. The characters of w are matched against the labels of the trie, beginning at the root node and looking for an edge that is labeled with w_1, then traversing this edge to the child node and again looking for the next character. If the search word is exhausted by matching all symbols downwards along the trie edges, then w must be a prefix of a string in S. Thus, if the last node has any children, then w is a proper prefix of a word in S, otherwise w is obviously equal to one of the strings in S. If the search gets stuck, then w cannot be contained in S, since otherwise the insertion of this word would have produced a branch with the right symbol at the node after the last matching edge.

To state the result in a more explicit way, we emphasize that containment (i.e., equality of the search string with one of the strings) in the trie can be detected correctly (in the prefix case) only if it is guaranteed that for all pairs of strings $(s_i, s_k) \in D \times D$ holds that s_i is not a proper prefix of s_k. Assume the contrary and suppose the word “`pineapple`”

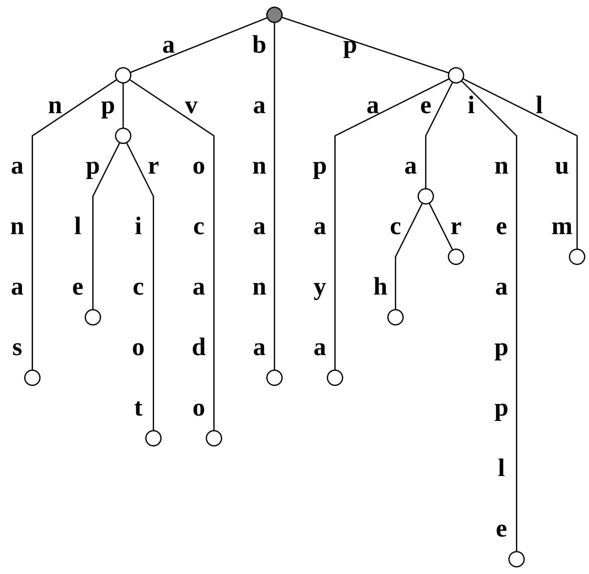


Figure 3.3: The compact Σ^+-tree for the words `ananas`, `apple`, `apricot`, `avocado`, `banana`, `papaya`, `peach`, `pear`, `pineapple`, and `plum`.

is contained in the dictionary. If we are searching the string "`pine`", we end up with a completely matched pattern, but we do not know whether this is because "`pine`" is contained in D, or this is only due to the word "`pineapple`".

An easy solution to this problem is the introduction of a new (auxiliary) character which must not be contained within one of strings in the dictionary (resp. the original alphabet). This character, also called the *sentinel*, guarantees that each word s_i of the dictionary D is represented by a leaf of the trie (see Figure 3.2).

After the trie for a set D of strings has been built in a preprocessing step, it serves the main purpose of answering queries that ask whether a given search pattern p equals one of the strings in D. The (worst case) *time complexity* of these queries depends only on the size of the search pattern, it is independent of the size of the dictionary/database.

3.3 Compact Σ^+-Trees (PATRICIA Trees)

A close relative of the trie is the Σ^+-tree that results from merging the non-branching parts of the trie. Allowing arbitrary (non-empty) edge labels, one can omit all nodes (except the root) having exactly one child. This method is called *path compression*. The resulting path-compressed Σ^+-tree, independently developed at nearly the same time by MORRISON [Mor68] and GWEHENBERGER [Gwe68], is also called *PATRICIA-tree*. Sometimes it is also referred to as the *compact(ed) trie/Σ^+-tree*. An example that relates to the previously given instances of tries, is given in Figure 3.3. Figure 3.4 shows the same tree, but this time using a sentinel character (`$`).

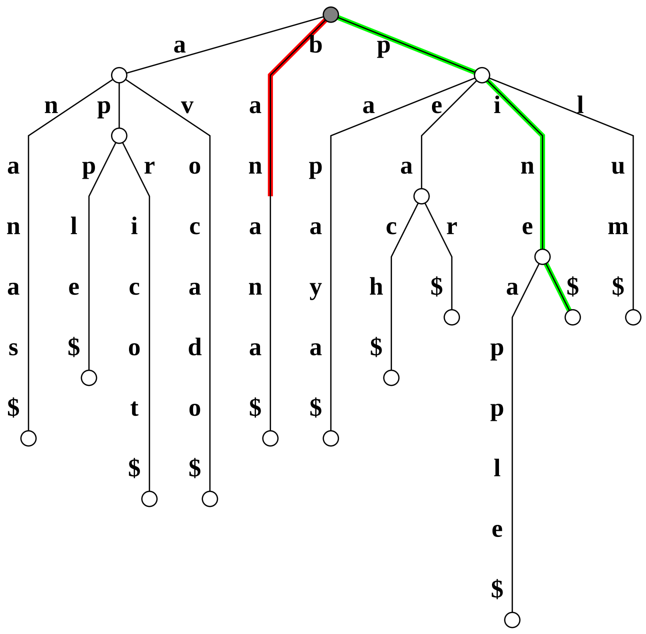


Figure 3.4: The compact Σ^+-tree for the words `ananas`, `apple`, `apricot`, `avocado`, `banana`, `papaya`, `peach`, `pear`, `pineapple`, and `plum` using the sentinel `$`.

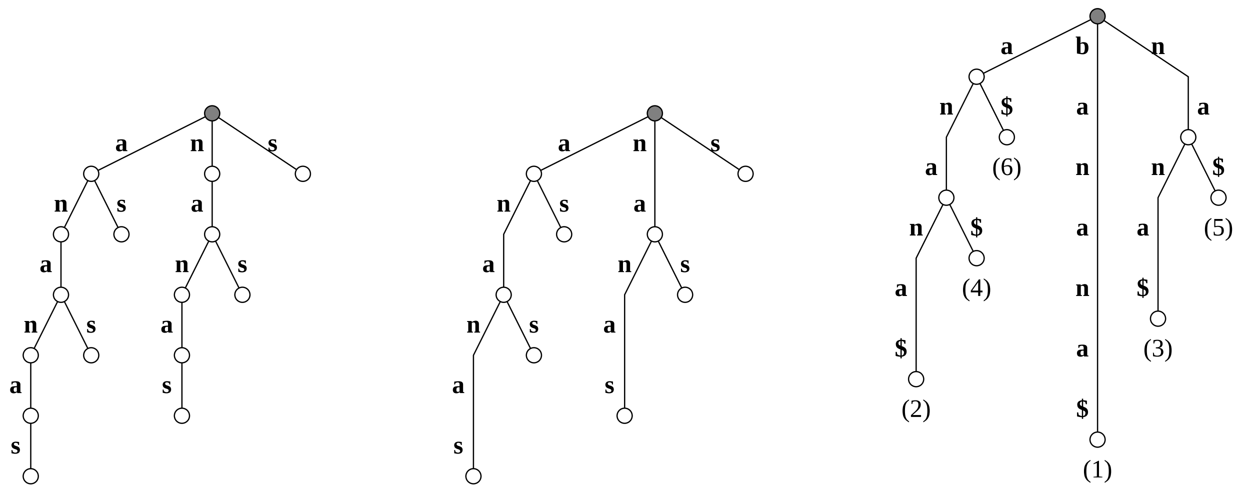
3.4 Suffix Tries and Suffix Trees

Besides deciding on the containedness of entire strings in a dictionary, the data structures of the preceeding sections can be elegantly used to (repeatedly) answer the question whether a given pattern p is contained as a *substring* within a fixed text t. Even further, they can be augmented to data structures that efficiently solve the problem of searching *all occurrences* of a pattern p in a text $t = t_1 \cdots t_n$.

For serving this purpose, it is assumed that a trie is built from a set of strings D that comprises exactly the suffixes of t (including the complete text). The resulting trie is called the *suffix trie* of t. It is easily seen that, using an equivalent matching procedure to the normal trie, one can decide whether a given pattern $p = p_1 \cdots p_m$ is a prefix of a suffix of t, that is, whether p is contained as a *substring* within t. For an example, see Figure 3.5(a).

Once the suffix trie for t has been constructed in a preprocessing step, the complexity of the search procedure does no longer depend on the length of t. It is linear in the length n of the text t. The suffix trie can easily be augmented to also report the position of the first (or last) occurrence of p in t. In this case, this value has to be stored within each of the nodes.

If the positions of all occurrences have to be found, it must be ensured that all occurrences, or more precisely their corresponding suffixes of t, are uniquely represented by a leaf of the trie. This is achieved using the sentinel approach described in the previous section. Now, the whole subtree under the current node after matching all pattern characters has to be traversed. For each leaf, the stored position is reported. A disadvantage of the suffix



(a) The suffix trie for the word `ananas`.

(b) The suffix tree for the word `ananas`.

(c) The suffix tree for the word `banana$`.

Figure 3.5: The suffix trie and the suffix tree for the word `ananas`. The suffix tree for the word `banana$` illustrates how the sentinel assures the existence of a leaf for every suffix.

trie is the fact that the number of nodes within a subtree is not linearly bounded by its number of leaves, which corrupts the linear worst-case complexity of the query procedure in this case.

This drawback can be fixed using the path compression technique of the compact Σ^+-trees (PATRICIA-trees). The tree that results from removing all non-branching nodes but the root is consequently called the *suffix tree*. Except for the leaf nodes and (possibly) the root node, the suffix tree contains only branching nodes, that is, nodes having at least two children (see Figure 3.5(b)). An immediate important implication is the fact, that the number of inner nodes of the tree is bounded by the number of leaves. An equivalent statement holds true for each subtree.

The major drawback of suffix *tries* was the possibly quadratic size of the data structure in terms of the text size. This behavior is shown, for instance, by the class of strings "$\mathtt{a}^i\mathtt{b}^i\$$". We already bounded the number of nodes (and edges) of suffix *trees* by a linear function of the number of leaves, that is, by a linear function of the text size. Anyway, the overall size of the representation can be quadratic since the edge labels do not have constant size. This problem can be solved, if the original string is stored in a normal array, and each edge label is represented by only two numbers: either a start and end index for the original string array, or a start index and the corresponding length that marks a position of the edge label in the array of the original string. Now, all edge labels have *constant size.* It follows that the data structure is represented in memory using an amount of space that is linear in the text size $|t| = n$.

As an instantaneous consequence, the worst-case time complexity for reporting all occurrences of a search pattern $p = p_1 \cdots p_m$ (by traversing the actual subtree after successful matching) is $\mathcal{O}(m + k)$, where k denotes the number of occurrences of p in t, that is, the number of leaves in the subtree of the current node after matching all characters of p.

For the time being, the linear size of the suffix tree does not imply that construction is possible in linear worst-case time. On the contrary, at first sight it seemed to be impossible to construct suffix trees within that time bound. Thus, it was quite surprising when WEINER [Wei73] presented the first linear-time construction algorithm for suffix trees in the early seventies. A little later, MCCREIGHT [McC76] presented a more space-efficient algorithm. A major drawback of MCCREIGHT's algorithm was the fact, that it constructed the suffix tree beginning with the largest suffix, that is, the complete string. A consequence was that the whole string had to be known in advance. This problem was solved two decades later, when UKKONEN [Ukk95] presented another linear-time construction, that iteratively constructs the suffix trees for increasing lengths of the text. Therefore, this algorithm is said to work *online*, because it may compute the suffix tree for a growing text. We will describe the algorithm in more detail within the next chapter where we (ab)use it in a slightly changed variant to search for polypeptide structures.

An excellent review of these algorithms was written by GIEGERICH and KURTZ [GK97]. Further information can also be found in the book of GUSFIELD [Gus97]. For considerations regarding implementation details, we refer to the papers of ANDERSSON/NILSSON [AN95], and KURTZ [Kur99].

Chapter 4

Searching in Protein Structure Databases

The first question any structural biologist asks on
being told that a new structure has been solved
is no longer “What does it *look* like?”;
it is now “What does it look *like*?”.

Gregory A. Petsko [Pet91]

4.1 Previous Work on Structure Searching

In order to make pairwise comparisons practical for larger databases, classical methods like *CE* [SB98], *DALI* [HS94], or *VAST* [GMB96] use a two-phase approach. The first phase constitutes a filter mechanism where a set of candidate proteins is generated. Various strategies exist for this phase, including the alignment of larger parts of the molecule (such as secondary structure elements, SSEs), comparison of intramolecular distance matrices, comparison of feature profiles (mostly geometrical or chemical property vectors), or combinations of the methods. In the following refinement phase, the remaining candidate set is inspected by more intricate and time-consuming algorithms. This requires the candidate set to be not too large. The set of reported positives (hits) is constructed, for example by identifying matching backbone or C_α atoms and computing root-mean-square distance (RMSD) values between the query and all candidate structures. In many cases, this process involves the computation of an optimal alignment of the (pattern, candidate) pairs, which is often implemented by some kind of *dynamic programming* algorithm. Due to the quickly increasing size of structure databases, newer approaches further trade off precision for speed by cutting short this dynamic programming algorithm by certain heuristic stop criteria. This immediately implies an additional loss in sensitivity.

It follows a short description of the most prominent existing tools that feature structure database searching. Please note, that it cannot be finally decided whether the main concern of a tool is structure comparison, structure alignment, structure superimposition, or

Name	Data & Algorithm	Reference
3D-Hit	3D-fragment hashing	[PPvGR02]
CE	combinatorial extension of fragment alignments	[SB98]
DALI	intra-molecular distance matrices, hexapeptide regions, matrix alignment, Monte Carlo method for full alignment	[HS93, HS94, HS96]
DEJAVU	intra-molecular distance matrices, vector-represented SSEs	[KJ97, MK02]
FATCAT	dynamic programming on AFPs	[YG03, YG04a, YG04b]
FoldMiner	alignment of secondary structure elements	[SB04b, SB04a]
Jess	range queries in a kd-tree	[BT03a]
MATRAS	Markov model, environmental score, residue-residue distance score, SSE score	[KN00, Kaw03]
PRIDE, PRIDE2	distribution of $C_i^\alpha - C_{i+k}^\alpha$ distances	[CP02, GVP05]
ProGreSS	multidimensional index of feature vectors extracted from sequence and structure	[BCK$^+$04]
PAST/ProSt	dihedral angle based suffix tree (this work)	[BT03b, BTG03, TBG04]
ProtDex, ProtDex2	fast filtering by indexing structures, precomputed feature vectors	[AFT03, AT04]
ProteMiner	search for similar binding sites	[CCC$^+$04]
PROuST	hash table based on geometric features of SSE triplets	[CGZ04]
PSI	feature vectors on triplets of SSEs	[ÇKS03a, ÇKS03b]
PSIST	normalized local geometric feature vectors indexed by a suffix tree	[GZ05]
SPASM	intra-molecular distance matrices of C_α carbons, DFS	[KJ97, Kle99, MK02]
SSM	alignment of SSEs	[KH04]
TESS	geometric hashing	[WBT97]
TOP	alignment of SSE subsets	[Lu00]
TOPS	comparison of topology diagrams of SSEs	[GWNT99, MTGW04]
TopScan	SSE-based topology strings	[Mar00]
VAST	graph-based SSE alignment	[GMB96]
YAKUSA	C_α virtual bond angles (τ) and virtual bond torsion angles (α), DFA-based scan of the entire database to find SHSPs followed by ranking	[CBP05]

Table 4.1: Overview of Protein Structure Comparison and Searching Tools

structure searching. These topics are tied together rather closely. Thus, some of the tools we mention here are alignment or comparison programs which were (ab)used to perform structure searches.

DALI [HS93, HS94, HS96] (Distance matrix ALIgnment) seems to be one of the most cited and used tools to perform structure alignments, comparisons and searches in the molecular biology community. The method represents each structure by a matrix of intra-molecular atom distances. This idea already dates back to the work of PHILLIPS [Phi70]. The matrix stores the distances of all pairs of atoms within a molecule. To simplify matters and to allow comparison of matrices for structures having different amino acid sequences, not all atoms, but only the C_α atoms (or alternatively all backbone atoms) are used for the distance map. An advantage is the invariance against the frame of reference (the position and orientation in the actual coordinate system). A disadvantage can be the equal representation of enantiomers. Since these isomers are mirror images of each other, they share a common distance behavior and cannot be distinguished by this kind of approach. DALI is used to maintain the **FSSP** [HS97] database by exhaustive all-against-all structure comparison and hierarchical clustering.

Overlaying the matrices of different structures can detect similarity along the main diagonal, which means backbone conformation (that is, secondary structure). Off-diagonal similarity may indicate more or less similar distances of atoms that are more distant within the sequence (that is, tertiary structure). Major advantages are that gaps of arbitrary length are allowed, as well as the reversal of chain direction and free topological connectivity of the aligned segments.

Later on, a modified version using a fast look-up [HS95] was introduced that utilizes alignments of secondary structure elements (SSEs) and shares some similarity with VAST (see below).

SPASM [Kle99, MK02] (SPatial Arrangements of Side-chains and Main-chain) is one of the few (very useful) programs to specialize in finding functional motifs. It is a two phase method, based on intra-molecular distance matrices of C_α carbons and, for non-glycine residues, by pseudo atoms at the center of gravity for each side chain. A simple recursive depth-first search (DFS) algorithm is used, which was originally used for the automatic assignment of two-dimensional and three-dimensional ^1H NMR spectra of proteins, and later in the fold recognition tool **DEJAVU** [KJ97, MK02] using early pruning of the search tree. The SPASM tool allows to apply several options on the search process. The user may choose whether only C_α carbons, only the side-chain pseudo-atoms, or all (pseudo) atoms should be used. The residues may be restricted to be of the same type, to be in a set of allowed substitutions, or of arbitrary types. The search can be further constrained by restricting the search to the same sequence order. Another option restricts the result sets to preserve the direct sequence neighbor relationship. Furthermore, conserved gap lengths between matched residues can be requested. It should be mentioned, that SPASM's ability to handle arrangements of non-consecutive residues is a major advantage in searching functional constitutions such as the catalytic triad.

VAST [GMB96] The Vector Alignment Search Tool is a protein comparison tool that looks for pairs of secondary structure elements (SSEs) that have similar type, relative orientation, and connectivity. The likelihood that this similarity would be seen by chance rates the significance of the considered matches.

YAKUSA [CBP05] (Yet Another K-Uples Structure Analyser) is a software that uses a deterministic finite automaton (DFA) in a BLAST-like search for structural high-scoring pairs (SHSPs). First, the DFA describing similar structures is built. Afterwards, the DFA searches the pattern in all structures of the database, then the found patterns are extended to yield the SHSPs. After selecting compatible SHSPs for each pair of query and database structure, they are ranked according to SHSP similarity, SHSP probabilities, and spatial compatibility of the SHSPs. The method uses discretized virtual bond (τ) and virtual bond torsion (α) angles [Lev76] to describe the protein structure.

For a comparison of different aspects of these approaches, see the reviews of Bourne / Shindyalov [BS03], Novotny et al. [NMK04], and by Sierk and Kleywegt [SK04].

The approaches described above share some common drawbacks. The candidate selection in the filter phase sacrifices accuracy for speed by relying on abstractions like distance matrices, SSE topology or feature vectors. Additionally heuristics are often employed to ignore unlikely database entries. Yet the methods of the refinement phase generally remain very expensive, restricting the filter phase to a small candidate set. As a consequence the user is left with the choice of accepting a very slow response to his query or greatly increasing the risk of pruning true positives in the filter phase. Furthermore, by relying on such features as SSEs, areas of high local similarity (*functional motifs*) may remain unidentified. This is most unfortunate since the functional motifs determine a protein's function to a large degree.

Further methods for aligning, comparing and searching (sub)structures were published in [NO74, RA76, Les79, MR85, ZS89, JOE$^+$94, HLS$^+$95, God96, WLT96, ZKS96, AOI97, Fin97, Toh97, GWT98, LKSD00, PGR$^+$01, CL02, GLZ02, IDP$^+$02, CHTY03, JHF03, OKA03, PRS03a, PRS03b, SH03, Zem03, PR04a, PR04b, PR04c, SDSD$^+$04, YJL04, YG05, CFK$^+$05, ZW05]. The query complexity of the most tools scales linearly with the size of the database. Hence, they are by far **too slow** for interactive working! Often they trade accuracy for speed.

4.2 Drawbacks of Current Methods

As we have seen from the description of the Protein Data Bank in the introductory chapter, there has been accumulated a huge number of structures which is much more than what could be maintained by only a single person. In contrast to the first years of the PDB, it is now completely impossible, even for experts, to know about all structures contained in the PDB that are related to a certain established subject of molecular biology. It has simply become very hard to keep pace with the growth of the database.

This statement implies that all researchers working with PDB data must resort to using automated search tools that filter the content of the database according to specified criteria. These criteria mainly comprise three categories: text searching in the annotation (context information) of the database entries, searching specified amino acid sequences or parameterized sequence patterns, and structure or substructure searching.

4.2.1 Text Searching

The easiest method to find structures that are related to a certain keyword is to apply a full text search on all context information contained in the database. But, everyone who has ever performed such a text search in a real life situation, for instance using one of the WWW search engines (like Google, Yahoo, etc.), knows about the problems that are associated with this approach. Usually, the result set is either empty or very large, and it is not easy to rank the hits such that the most significant ones are presented first. If the user considers searching for several keywords at once, it ends up most of the times that not a single hit is found because in many of the potential hits at least one of the words is missing. Then the question arises which of the subsets of keywords are considered sufficient. The union of all answers is mostly a long term that no one wants to deal with.

In some of the cases a potential target entry is not found due to an ambiguity of the term, that is, the respective document contains only a synonym of the given search term. That is particularly fateful for searching biochemical compounds because there are systematic names and common names that relate to the same molecule. Some of the molecules also have old (obsolete) names besides their currently used denomination. In some cases there are two ore more common names in use, and even systematic names are sometimes ambiguous. Furthermore, there exists an abbreviation (or several of them) for the majority of the molecule names (since the complete name is usually too long). The problem can only be solved by a dictionary of synonyms, but this seems to be an unrealistic wish for the matter of molecular biology (at least from today's point of view).

Sometimes, it happens that the target of the search has no name at all. This is particularly true if we consider searching parts of molecular structures. These *substructures* usually have a dedicated name only in the case where they have an associated function or if they are already known to be frequent substructures.

A particularly nasty problem occurs, if the search term is used within database entries of other structures to state some degree of relatedness or interaction, or, even worse, to state that the search term should not be confused (or does not interfere etc.) with the entry in consideration. Just this remark, that explicitly introduces a distinction to the search term, is taken to be a hit indicator within a full text search. For proteins, there are usually many interaction partners (that are referenced within the entry), which is a major drawback of the text searching approach.

Another problem occurs, if one is interested only in a specific form of a molecule. As an example, hemoglobin is able to bind a different number of oxygen atoms, where the structure differs for each of the binding states. If we wanted to search for all hemoglobin structures where a fixed number of oxygen atoms is bound (that is, with a fixed conformation), we would rely on the correct chemical designation, which leads to a problem already discussed.

4.2.2 Searching Amino Acid Sequences

A second approach to searching in biomolecular databases can be applied if the primary structure (that is, the sequence of bases or nucleic acids, see Chapter 2) of the molecule or a close relative is known in advance. In this case, one of the classical pattern matching and alignment algorithms can be applied to find similar sequences more or less efficiently. Algorithms that solve the exact search problem were listed in the previous chapter. Algorithms that solve an approximate matching problem (that is, finding a similar but not necessarily equal sequence) are usually based on a solution of the optimal pairwise alignment problem, but these algorithms (for instance, the classical algorithms by Needleman and Wunsch [NW70], and by Smith and Waterman [SW81], see also [Got82, GG89]) are in principle very slow. Faster solutions utilize indexing techniques [Ukk93, Cob95, NBYST01, HAI02].

It appeared that popular methods that utilize various heuristics for filtering are much faster in practice (for instance the FASTA family of programs by Lipman and Pearson [LP85, PL88], and BLAST [AGM+90] by Altschul et al.) Their great advantage, the filter step that sacrifices sensitivity of the search for speed, is, at the same time, a great handicap. This means, there is no guarantee to find a near-optimal solution. Unfortunately, it has become a habit in the molecular biology community to rely on the results of these heuristic algorithms without questioning whether the result set contains all possible solutions. It must be emphasized that these programs have significant problems in detecting distant functional relationships of biopolymers where only weak sequence identity is present (q.v. [Pea97, Ros99]). They often miss out the hits that have only few amino acids in common. We learned from the introductory chapter that structure is more preserved than the pure amino acid sequence because **structure determines function**, and quite different sequences may result in very similar structures! According to a statement of Levitt and Gerstein [LG98], the structural comparison can detect approximately twice as many distant relationships as sequence comparison at the same error rate. The evolutionary effect, that development alters the sequence while the structure is kept the same, is called *divergent evolution*. The consequence must be to take structure-based aspects into consideration wherever this is possible.

4.3 Evaluation of Service Quality

At this point we should note, that usually, the performance of search algorithm is measured in terms of quality and time consumption. While the elapsed time for the search process can be easily measured, the (absolute) quality of the result set is hard to assess, since quality is mostly measured by the quantities sensitivity, specificity, and precision (positive predictive value). These are expressed in terms of *true positives* (correctly identified hits), *false positives* (spurious hits), *true negatives* (correctly identified non-hits), and *false negatives* (matching target entries that were not reported as hits). All four values partition the database entries according to the decision of the search procedure, and thus add up to the total number of entries.

$$\text{sensitivity} = \frac{\text{number of true positives}}{\text{number of true positives} + \text{number of false negatives}}$$

$$\begin{aligned}\text{specificity} &= \frac{\text{number of true negatives}}{\text{number of true negatives} + \text{number of false positives}}\\ \text{precision} &= \frac{\text{number of true positives}}{\text{number of true positives} + \text{number of false positives}}\end{aligned}$$

Thus, the calculation of each of the three measures requires knowledge of either the absolute number of true positives or the absolute number of true negatives. In most cases, such a perfect partition of the database is impossible to compute due to complexity of the exact algorithms and the large size of the database. What remains is a direct (relative) comparison of the search results for different tools.

4.4 The Polypeptide Angles Suffix Trees

4.4.1 Construction of the Suffix Tree

For the moment, let us assume we want to search a pattern sequence $p = p_1 \cdots p_m$ of length m within a *single* longer text $t = t_1 \cdots t_n$ of length n. We use the suffix tree construction of UKKONEN that computes the series of suffix trees for the growing text, that is, for $\mathbf{pre}_0(t)$, $\mathbf{pre}_1(t)$,..., $\mathbf{pre}_n(t) = t$. This makes it amenable to a setting where the characters of the string become known one by one over time and the current suffix tree is updated to become the suffix tree for the extended text. This so called *online* property is in contrast to the former linear-time algorithms. MCCREIGHT's algorithm inserts the suffixes in order of decreasing length into the Σ^+-tree, starting with $\mathbf{suf}_n(t)$ and lastly adding $\mathbf{suf}_1(t)$. The intermediate Σ^+-trees do not constitute suffix trees. The same applies to the algorithm of WEINER[1], whereas this algorithm inserts the suffixes in order of increasing length from $\mathbf{suf}_n(t)$ to $\mathbf{suf}_1(t)$. In both, MCCREIGHT's and WEINER's algorithm, the whole string has to be known in advance, since the last character is involved in the first step of the algorithm. Since we will need the (slightly changed) property of online construction, we stick to the UKKONEN algorithm although there are minor performance advantages for the MCCREIGHT algorithm, see [GK97].

For later use, we define a substring s of t to be *right-branching* in t, if there are different characters $x, y \in \Sigma, x \neq y$, such that sx and sy are substrings of t. Obviously, all suffixes $\mathbf{suf}_i(s)$ of a right-branching substring s are right-branching too (with respect to t).

A central role in the concept of all linear-time suffix tree construction algorithms play suffixes that also occur in another position as a substring of t. These suffixes are called *nested*. The crucial point is, that nested suffixes are *not* represented by leaves within the suffix tree. As already discussed in the previous chapter, we therefore assume the text to end with a sentinel character `$`. Thus, no non-empty suffix of t is the prefix of another

[1]Please note, that WEINER used a different naming convention, where the terms suffix tree and prefix tree are interchanged.

suffix, or, using the notation above, the text $t\$$ has no nested suffix, except for the empty string.

A simple observation reveals that all suffixes of a nested suffix $\mathbf{suf}_i(t)$ are also nested, since the occurrence at position k in t

$$\mathbf{suf}_i(t) = \mathbf{pre}_{|\mathbf{suf}_i(t)|}(\mathbf{suf}_k(t)) \qquad (k < i)$$

can be written as

$$t_i \cdots t_n = t_k \cdots t_{k+n-i}$$

and implies that

$$\forall l \in [1, n-i+1] : \quad t_{i+l} \cdots t_n = t_{k+l} \cdots t_{k+n-i}$$

which means

$$\forall l \in [1, |\mathbf{suf}_i(t)|] : \quad \mathbf{suf}_{i+l}(t) = \mathbf{pre}_{|\mathbf{suf}_{i+l}(t)|}(\mathbf{suf}_{k+l}(t))$$

that is, every suffix $\mathbf{suf}_l(\mathbf{suf}_i(t))$ of the suffix $\mathbf{suf}_i(t)$ occurs as a prefix of some suffix of t, namely the suffix $\mathbf{suf}_{k+l}(t)$.

Consequently, the set of nested suffixes can be characterized by the longest nested suffix, which is called the *active suffix*, denoted by $\alpha(t)$. The active suffix can be represented by the respective node in the (atomic) suffix trie that is determined by the usual matching procedure (that is, the node whose path from the root is labeled by the active suffix). But, since we are dealing with (compact) suffix trees, this node might correspond to a (virtual) *implicit node* which is located on an edge because the suffix trie node is not branching. Therefore, this implicit node is described by an (existing) *explicit node* v together with a string x such that the label of the path from the root node to v, concatenated with x (the labels of the path from v to the virtual position) equal the active suffix:

$$\alpha(t) = \mathbf{label}(\text{rootNode}, v) + x$$

The combination of the explicit node v and the label x is called a *reference pair* (v, x). The reference pair having the shortest string $x \in \Sigma^*$, that is, the representation relative to the deepest possible explicit node v, is called the *canonical reference pair*.

To simplify the description, we introduce the following notation: For a string s, $\overline{s}$ denotes the explicit or implicit location whose path from the root node is labeled with s. Furthermore, it is most convenient for the construction and for some applications of suffix trees to extend the nodes of the tree by pointers called *suffix links* [Ukk95]. These pointers implement the *suffix function* for explicit nodes $\overline{s}$, that is defined for a string $s \in \Sigma^*$, as follows:

$$\mathrm{f}(\overline{s}) = \begin{cases} \perp, & \text{if } s = \varepsilon \\ \overline{s'}, \text{ s.t. } s = xs' \text{ with } x \in \Sigma, s' \in \Sigma^*, & \text{otherwise } (s \in \Sigma^+) \end{cases}$$

where $\perp$ is an extra node that can be interpreted as kind of an error state, if the suffix tree is viewed as a deterministic finite automaton (DFA) with states and labeled transitions.

This auxiliary state helps to avoid separate treatment for the root node compared to the other nodes (which corresponds to empty and nonempty suffixes). It has a transition to the root node ($\overline{\varepsilon}$) for each character of the alphabet. The suffix function of the error node $\mathrm{f}(\perp)$ is left undefined.

Suffix links are the concrete implementation of the suffix function for explicit nodes. Hence, the suffix link of the root node is the error node ($\perp$). For a non-empty string $s = xs'$ that consist of a character $x \in \Sigma$ and a string $s' \in \Sigma^*$, let $\overline{xs'}$ be an explicit node of a suffix tree $\mathcal{T}$. The suffix link of $\overline{x} = \overline{xs'}$ is then defined to be

$$\text{suffixlink}(\overline{xs'}) = \overline{s'}$$

It can be shown, that this suffix link is well-defined for the suffix tree of $t\$$, since it is always an explicit node: $\overline{xs'}$ is either a branching node or a leaf. Thus, xs' is either right-branching, then s' is right-branching too, or xs' is a non-nested suffix of $t\$$, then the same holds for s'. Hence, $\overline{s'}$ is an explicit node in the suffix tree of $t\$$. If the sentinel is left out, the suffix function $\mathrm{f}(\overline{s}) = \mathrm{f}(\overline{xs'}) = \overline{s'}$ of a leaf $\overline{s}$ could refer to a substring s' that is nested and not right-branching, such that $\overline{s'}$ is not an explicit node. (All other cases of suffix links remain faultless.)

Before we proceed to the final suffix tree construction, we have a look at the concept using the suffix trie. The algorithm iteratively constructs suffix tries $\mathcal{T}_i'$ for growing prefixes $\mathbf{pre}_i(t)$ of the text, starting with the first trie $\mathcal{T}_0'$ for ε (being just the root node with no edges), transforming it into $\mathcal{T}_1'$ for t_1. In each iteration, $\mathcal{T}_{i+1}'$ is constructed from $\mathcal{T}_i'$ by extending all suffixes by the new character t_{i+1}. To avoid repeated searching for the correct node for each suffix, the suffix links are used that point from $\mathbf{suf}_k(\mathbf{pre}_i(t))$ to $\mathbf{suf}_{k+1}(\mathbf{pre}_i(t))$ after the iteration of prefix $\mathbf{pre}_i(t)$. Thus, we just store the largest suffix and extend the suffixes by traversing them along the suffix links. The traversal ends at the root node, but can be interrupted earlier, if we find a nested suffix. In this case, no further nodes have to be created and all further suffixes are nested too. The concept of one iteration is shown in Algorithm 1.

The algorithm that constructs the suffix tree works, in principle, the same way. Of course, the main difference is, that nodes having only one child are no longer in the tree (except for the root). They are implicit nodes. Thus, we only create new nodes that are leaves or have at least two children. To serve this purpose, the suffixes of each iteration are distinguished in three categories:

1. suffixes, that are nested in $t_1 \cdots t_i$, and in $t_1 \cdots t_{i+1}$ too (irrelevant suffixes),

2. suffixes that are nested in $t_1 \cdots t_i$, but non-nested after the extension by t_{i+1} to $t_1 \cdots t_{i+1}$ (relevant suffixes).

3. suffixes that are not the prefix of another suffix (non-nested suffixes),

The non-nested suffixes are represented by leaves. Since they cannot be nested in later iterations, the only problem to solve is the elongation of edge labels that point to leaves. This was elegantly solved by UKKONEN via the *open edge* concept. The length field or end

Algorithm 1: constructSuffixTrie($\mathcal{T}_i'$, lsuf$_i$)

```
Input:  $t$                                                    // the text
        $\mathcal{T}_i'$                  // the suffix trie of the previous iteration
        lsuf              // the pointer to the longest suffix $\mathrm{suf}_1(\mathrm{pre}_i)$

Result: $\mathcal{T}_{i+1}'$                           // the suffix trie for $\mathrm{pre}_{i+1}(t)$
        lsuf            // the pointer to the longest suffix $\mathrm{suf}_1(\mathrm{pre}_{i+1})$

$v \longleftarrow$ lsuf
$p \longleftarrow \perp$ ;                                     // the predecessor
while $v \neq \perp$ do
    // Extend suffix $v$
    if $\nexists$ child $w$ of $v$ with edge label $t_{i+1}$ then
        Create node $w$ with edge $(v, w)$ labeled by $t_{i+1}$
        if $p \neq \perp$ then
            $(p \rightarrow \text{suffixlink}) \longleftarrow w$
        else
            lsuf $\longleftarrow w$
        $p \longleftarrow w$
        $v \longleftarrow (v \rightarrow \text{suffixlink})$
    else
        if $p \neq \perp$ then
            $(p \rightarrow \text{suffixlink}) \longleftarrow w$
            $v \longleftarrow \perp$
return
```

index of the edge label stores a special flag for indicating the open-edge property, which means, the edge extends to the end of the currently treated suffix. If the suffix tree is updated with the i-th character (t_i), the edge extends exactly to this point. This method prevents the elongation of all leaves in each iteration. Once a leaf is created, it grows without explicit manipulation.

The first type of suffixes are easy to handle. As in the case of suffix tries, nothing has to be done for suffixes that keep being nested during one step.

In each iteration, the empty suffix is added, which is always nested, and thus of the first category. Afterwards it grows, and in the iteration before it gets nested, it is of the second category. In all following iterations it is non-nested, that is, of the third category.

Thus, the critical point is the transition from nested to non-nested suffixes. Therefore, they are called *relevant suffixes*. We only need to store the longest nested suffix (the active suffix, see above), because all shorter suffixes are nested, whereas all longer suffixes are non-nested. As has already been mentioned, the active suffix is represented by the canonical reference pair. The algorithm checks by calling the function `testAndSplit()`, whether the active suffix is nested after the current iteration. This is true, if the node of the reference pair (the reference node, pointer `refnode` in the algorithm) has an edge starting with t_{refidx} that has character t_{i+1} at position `reflen+1`. In this case, the function returns the pointer to the auxiliary node. Otherwise, it creates such a node by splitting the edge that starts with t_{refidx} (if it exists) after `reflen` characters. In the case, where `reflen` is zero, nothing has to be done.

In any case, `testAndSplit()` returns either the auxiliary node ($\perp$), or the node, that should be extended with the new character. In the first case, a match was found, and thus `reflen` is incremented and the reference pair is updated to the new canonical reference pair. In the latter case, a new node is added (as child of the returned node) which represents the relevant suffix. Furthermore, the reference pair is updated to the next (shorter) suffix by ignoring its first character. This can be done by replacing the reference node by the target node of its suffix link. After making the reference pair canonical again, the algorithm restarts with the nested-test. The iteration ends when the active suffix is no longer a relevant one.

Please note: the function that updates the reference pair to yield a canonical reference pair (`canonize()`) has not to look at every character for each traversed edge. Since the reference pair always points to a nested suffix, the respective string already exists in the suffix tree. Hence, only the first character of each traversed edge is needed to find the correct child of the current node. The function descends in the tree by matching the active suffix until this string is exhausted. The node that succeeds the last completely matched edge is made the new reference node (while simultaneously updating the variables `refidx` and `reflen`).

The total number of operations is bounded by a linear function of n:

- There are $n + 1$ iterations (one for each character of t, one for the sentinel).

- `testAndSplit()` needs only constant time for a canonical reference pair.

Algorithm 2: constructSuffixTree($\mathcal{T}_i$, refnode, refidx, reflen)

```
Input: $t$                                                    // the text
       $\mathcal{T}_i$                       // suffix tree of the previous iteration
       refnode        // node of the reference pair for active suffix $i$
       refidx        // index of the reference pair for active suffix $i$
       reflen       // length of the reference pair for active suffix $i$

Result: $\mathcal{T}'_{i+1}$                       // suffix trie for pre$_{i+1}(t)$
        refnode    // node of the reference pair for active suffix $i+1$
        refidx    // index of the reference pair for active suffix $i+1$
        reflen   // length of the reference pair for active suffix $i+1$

current_length $\longleftarrow$ current_length +1 ;       // enlarge the open edges
$v \longleftarrow$ testAndSplit(refnode, refidx, reflen, $t_{i+1}$)
$p \longleftarrow \perp$ ;                                   // the predecessor
while $v \neq \perp$ do
    // Extend suffix $v$ and go to next active suffix
    Create new node $w$
    Establish new open edge $(v, w)$ with label $t_i \cdots t_\infty$
    if $p \neq \perp$ then
        $(p \rightarrow$ suffixlink$) \longleftarrow v$
    $p \longleftarrow v$
    refnode $\longleftarrow$ suffixlink(refnode)
    canonize (refnode, refidx, reflen)
    $v \longleftarrow$ testAndSplit(refnode, refidx, reflen, $t_{i+1}$)
if $p \neq \perp$ then
    $(p \rightarrow$ suffixlink$) \longleftarrow$ refnode ;      // new nodes were added
reflen $\longleftarrow$ reflen + 1
canonize (refnode, refidx, reflen)
return
```

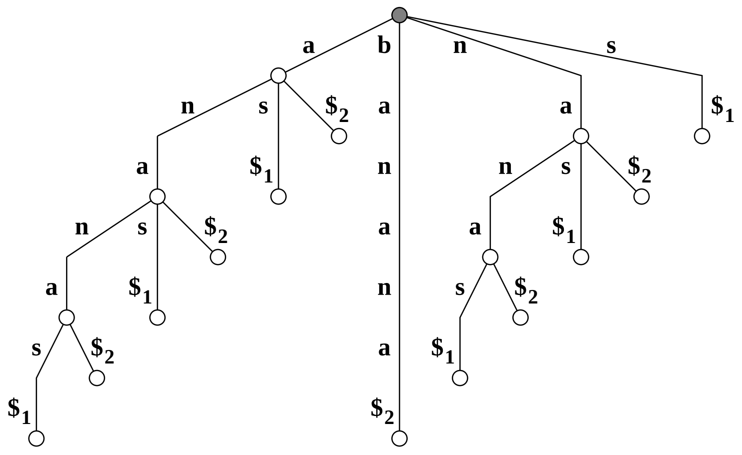


Figure 4.1: The generalized suffix tree for the words `ananas` and `banana`. The sentinel $\$_1$ marks the leaves of `ananas`, whereas $\$_2$ indicates the leaves of `banana`.

- The number of `while`-block executions is $n+1$, because in each iteration, a new leaf is created, and the suffix tree has exactly one leaf per suffix or per symbol.
- `canonize()` is executed $2(n+1)$ times (only once per iteration outside of the `while`-block, and only once per `while`-block execution). Within this function (see Function 4), the `while`-loop decrements the reference length (`reflen`) at least by one. Since each iteration of the main function increments `reflen` (a non-negative variable), there can be at most $n+1$ iterations of the `while`-loop in `canonize()`.
- There are no other non-constant parts of the algorithm.

Hence, the worst-case time complexity of the overall suffix tree construction algorithm is in $\mathcal{O}(n)$.

4.4.2 Generalized Suffix Trees

Suffix trees can be extended to the case where more than one text (that is, a set of sequences) has to be stored for searching. Consequently, these *Generalized Suffix Trees* (abbr. GST) can be used as indexing structures for sequence databases.

To check the existence of substrings in a string contained in the database, it would be sufficient to simply append all strings to a long sequence, and to build the suffix tree for that sequence (using a sentinel after each of the strings). Searching by the usual matching procedure would not only give an answer to the existence of the search pattern, it would also reveal the position within the long string. But, this is not what we want to know. Usually, we are interested in the identifier of the sequence, where the hit occurred, as well as the position with respect to that sequence. It would take too much time to determine these values from the global position. Furthermore, we want to know *all* occurrences of the search pattern.

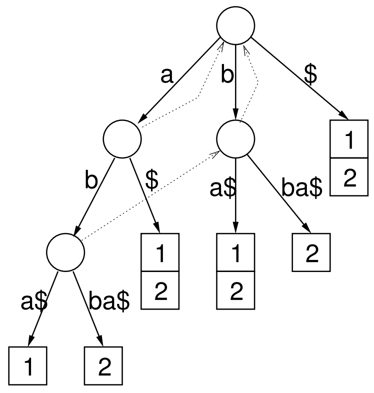


Figure 4.2: Generalized suffix tree for the strings `aba` and `abba`.

The latter problem could be solved by using a different sentinel $\$_i$ for each sequence s_i. Though, this can dramatically increase the size of the resulting alphabet, and thus, lead to an enormous waste of memory. For the moment, we adopt this model. Later, we will show how to implement it in a space-saving way. The suffix tree construction algorithm can now be applied successively to all of the input sequences. Due to the sentinel characters, all suffixes of each sequence are represented by a leaf in the GST. Figure 4.1 illustrates an example of the approach.

The resulting generalized suffix tree can be used, for instance, to search substrings in the usual way, to find longest common substrings of structure sequences, or to compute, for each substring, the number of different sequences that contain this substring [Hui92].

More information on generalized suffix trees can be found in the paper by Bieganski et al. [BRCR94] and in Gusfield's book [Gus97].

4.4.3 Implementation Issues

Sequence identifiers and edge label storage We augment the nodes of the suffix tree by a sequence identifier that gives us access to the desired information of the occurrences. For convenience, we assume that label information of all edges representing the transition from a parent node to a child node, is stored within the child node. This does not violate the condition of constant size nodes, because the underlying graph is a tree, and thus, each node has only one incoming edge from a parent.

Storage of child pointers For the implementation of child edges there are several conceivable variants. The pointers to the children can be implemented by an array: one entry for every possible character of the alphabet (which marks the first character of the edge label). An array allows direct access of the entries by using the character code for indexing, and is thus very fast. From the viewpoint of memory consumption, this variant would be quite efficient for the first (upper) levels of the tree, but it would waste most of

Algorithm 3: GenSuffixTree::insertSequence(newString)

```
Impl. Arg.: strings              // access to all strings of the GST
            s                    // access to the current string
            current_stridx       // the index of the current string
            current_length       // working length of the current string
            refnode              // the node of the reference pair
            refidx               // the start index for the reference pair label
            reflen               // the length of the reference pair label
Expl. Arg.: newString            // the sequence to be added to the GST
Result: The given string is added to the GST.
strings.store (newString)
s ⟵ address of current string
refnode ⟵ rootNode
refidx ⟵ 0
reflen ⟵ 0
rootNode→stridx ⟵ current_stridx
int len ⟵ newString.size ()
for int i ⟵ 0; i < len; i++ do
    current_length ⟵ i
    v ⟵ testAndSplit (s[i])
    p ⟵ errorNode
    while v ≠ errorNode do
        w ⟵ new_STnode ()
        w→stridx ⟵ current_stridx
        w→start ⟵ i
        w→end ⟵ len-1 ;      // for real online application use OPEN_EDGE
        if s[i] = SENTINEL and
        (currentChild ⟵ searchChild (v→children, SENTINEL)) ≠ NULL then
            w→suflink ⟵ currentChild
            replaceChild (&(v→children), SENTINEL, w)
            currentChild→next ⟵ NULL
        else
            w→suflink ⟵ NULL
            insertChild (&(v→children), w)
        if p ≠ errorNode then
            p→suflink ⟵ v
        p ⟵ v
        refnode ⟵ refnode→suflink
        canonize ()
        v ⟵ testAndSplit (s[i])
    if p ≠ errorNode then
        p→suflink ⟵ refnode
    reflen++
    canonize ()
return current_stridx++ ;                // increment after return value!
```

```
Algorithm 4: GenSuffixTree::canonizeRefPair( )
Impl. Arg.: strings                     // access to all strings of the GST
            s                           // access to the current string
            current_stridx              // the index of the current string
            current_length              // working length of the current string
            refnode                     // the node of the reference pair
            refidx                      // the start index for the reference pair label
            reflen                      // the length of the reference pair label
Expl. Arg.: –
Result: The reference pair is made canonical.
if $reflen > 0$ then
    if $refnode = \perp$ then
        refnode ⟵ rootNode
        reflen ⟵ reflen − 1
        refidx ⟵ reflen + 1
    if $reflen > 0$ then
        v ⟵ searchChild(refnode→children, s[refidx])
        if v→end = OPEN_EDGE then
            if v→stridx = current_stridx then
                l ⟵ current_length − v→start +1
            else
                l ⟵ strings[v→stridx].size() − v→start
        else
            l ⟵ v→end − v→start +1
        while (reflen> 0) and ($l \leq$ reflen) do
            refnode⟵ v
            reflen⟵ reflen − l
            refidx⟵ refidx + l
            v ⟵ 0
            childIt ⟵ searchChild(refnode→children, s[refidx])
            if childIt ≠ NULL then
                v ⟵ childIt
                if v→end = OPEN_EDGE then
                    if v→stridx = current_stridx then
                        l ⟵ current_length − v→start +1
                    else
                        l ⟵ strings[v→stridx].size() − v→start
                else
                    l ⟵ v→end − v→start + 1
return
```

Algorithm 5: GenSuffixTree::testAndSplit(x)

```
Impl. Arg.: vector<string>& strings    // access to all strings of the GST
            STnode& refnode            // the node of the reference pair
            int refidx       // the start index for the reference pair label
            int reflen            // the length of the reference pair label
Expl. Arg.: char x                     // the first character of the edge
Output: the parent node for the new child (or the errorNode)
if refnode = errorNode then
    return errorNode
if reflen = 0 then
    if not searchChild(refnode→children, x) or (x = SENTINEL) then
        return refnode                   // no such edge/child or new leaf
    else
        return errorNode                     // normal edge/child exists
curNode ⟵ searchChild(refnode→children, s[refidx])
if curNode ≠ ⊥ then
    if (strings[curNode→stridx][curNode→start + reflen] = x) and
      (x≠SENTINEL) then
        return errorNode                         // match, nothing to do
    else
        splitNode ⟵ new_STnode()
        splitNode→stridx ⟵ curNode→stridx
        splitNode→start ⟵ curNode→start
        splitNode→end ⟵ curNode→start + reflen - 1
        splitNode→next ⟵ ⊥
        splitNode→children ⟵ curNode
        splitNode→next ⟵ curNode→next
        replaceChild(&(refnode→children), s[refidx], splitNode)
        curNode→next ⟵ ⊥
        curNode→start ⟵ curNode→start + reflen
        return splitNode
return refnode
```

the available space in the deeper levels, because the branching degree of the nodes usually decreases with increasing node level. Most of the entries would be empty (meaning there is no such child whose edge label starts with the respective character).

An alternative would be to use a search tree. This option would present a compromise between access time and memory usage. Using a hash map would be possible too.

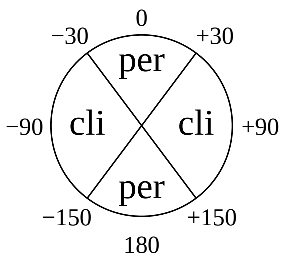
We emphasize the memory aspect and implement the GST using single-link lists of children. They only need space for existing child pointers. For access, we have to go through all entries to find a child pointer (or to learn, that no such child exists) in the worst case; but, as has been mentioned above, most of the nodes that reside in the GST, have only very few children (see also Figure 4.21). Some time can be saved by sorting the pointers according to the encoding of the alphabet. Then the search for a child with first character x on the edge label, can be canceled, if some greater character $y > x$ is encountered. (This introduces slightly more work for the insertion of a new child, but, since we intend to search repeatedly, this approach might pay off.)

Leaf lists Note that, occasionally, two or more sentinels of different sequences could be edge labels for children of the same node. In extreme cases, this would mean to have a lot of child pointers which would be very unfortunate in some of the searching and traversing algorithms. We elude this problem, using the fact that, in this case, these children would be leaves. As luck would have it, leaves usually do not have suffix links stored. Hence, we store only one child that ends with a sentinel; the remaining children of this kind are stored as a linked list via the suffix link pointer.

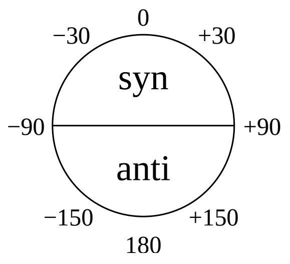
Creation of new nodes Since the number of nodes is not known in advance they must be created dynamically. These memory chunks, usually created with the `new` operator, are allocated in the dynamic memory (the heap). For reasons of memory management, each of these chunks would allocate some additional amount of space to store the size of the data structure. (This is needed for deleting the records afterwards.) This significant overhead of memory usage, has two disadvantages: it is slow and it wastes memory. Therefore we request memory from the system in larger chunks of predefined size, and get an appropriate piece for each new record using the function `new_STnode()`. A pointer to the next free entry is maintained. If a whole chunk has been exhausted, a new one is requested from the system.

Semi-online construction Our setting of the search problem will not require the generalized suffix tree to be constructed online character by character. We only wish to add complete strings in case there are new entries in the database. Therefore, we modify the construction algorithm by storing the full length of the respective string already at the time of creation of the leaves. This avoids the overhead which is due to the treatment of the open edge flag.

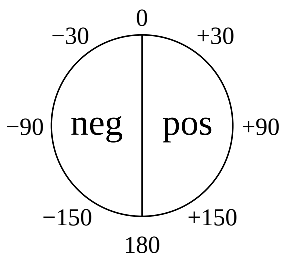
Further information on efficient implementation of suffix trees and related structures can be found in the work of ANDERSSON / NILSSON [AN95], and KURTZ [Kur99].



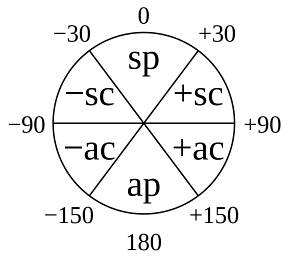
(a) periplanar / clinal



(b) syn / anti



(c) positive / negative



(d) scheme

Figure 4.3: Torsion angle classification by KLYNE and PRELOG.

4.5 Structure and Feature Representations

4.5.1 Computation of Bond Angles

In search for structure descriptions that are independent of the coordinate system, an obvious measure besides atom distances are angles. A simple *bond angle* α between two neighboring bonds $A - B$ and $B - C$ can be calculated from the positions of the three atoms A, B, and C via the normalized dot-product of the two bond vectors:

$$\alpha = \arccos\left(\frac{(\vec{a}-\vec{b})\cdot(\vec{c}-\vec{b})}{||(\vec{a}-\vec{b})||\cdot||(\vec{c}-\vec{b})||}\right)$$

4.5.2 Formal Definition and Computation of Torsion Angles

A *torsion angle* measures the torsion of two atoms that are attached to the opposite ends of a central bond. Looking from one side along the axis of the central bond $B - C$, the torsion angle is the angle by which the projection of the $B - A$ bond into a plane perpendicular to $B - C$ must be turned in order to overlay the projection of the $C - D$ bond. KLYNE and PRELOG [KP60] proposed the syn/anti clinal/periplanar terminology to describe the steric relationship across a single bond of conformational isomers (also called *rotamers*). This scheme is illustrated in Figure 4.3. We follow their convention of giving the torsion a positive sign if the rotation of the $B - A$ bond is clockwise to overlay the $C - D$ bond when looking along the axis from C towards D. Counterclockwise rotation clearly gives a negative value. The torsion angle of the four atoms is closely related to the *dihedral angle*, which measures the angle between the planes ABC and BCD (or between their crossing normals n_1 and n_2). For a formal definition of the dihedral angles the following difference vectors of the atom positions are defined:

$$\begin{aligned} a &= p_B - p_A \\ b &= p_C - p_B \\ c &= p_D - p_C \end{aligned}$$

The normals of the planes defined by ABC and BCD are then given by the cross-products $n_1 = a \times b$ and $n_2 = b \times c$ or, as unit vectors, $\frac{n_1}{|n_1|} = \frac{a \times b}{|a \times b|}$ and $\frac{n_2}{|n_2|} = \frac{b \times c}{|b \times c|}$ (where both are perpendicular to b). According to the intended definition of rotating the first plane into the second (and thus the first normal into the second) by angle α it must hold

$$\begin{aligned} R(b, \alpha)\frac{n_1}{|n_1|} &= \frac{n_2}{|n_2|} \\ R(b, \alpha)\frac{a \times b}{|a \times b|} &= \frac{b \times c}{|b \times c|} \end{aligned}$$

where $R(b, \alpha)$ denotes the matrix that defines the rotation by angle α perpendicular to the (directed) normal b.

The absolute value can be obtained using the normalized dot-product by the following formula

$$\begin{aligned} \cos(\alpha) &= \frac{n_1 \cdot n_2}{|n_1| \cdot |n_2|} \\ |\alpha| &= \arccos\left(\frac{n_1 \cdot n_2}{|n_1| \cdot |n_2|}\right) \end{aligned}$$

The torsion angle is also related to the *argument* θ (also called *phase*) of a complex number $z = x + iy = |z|e^{i\theta}$. It is the polar angle of the complex coordinates (x, y) and can be computed from the following equation

$$\arg(x + iy) = \arctan^*\left(\frac{y}{x}\right)$$

where $\arctan^*(z) \in [-\pi, \pi]$ takes into consideration which quadrant z is lying in.

Now we can derive the formula for computing the torsion angle from the four atom positions:

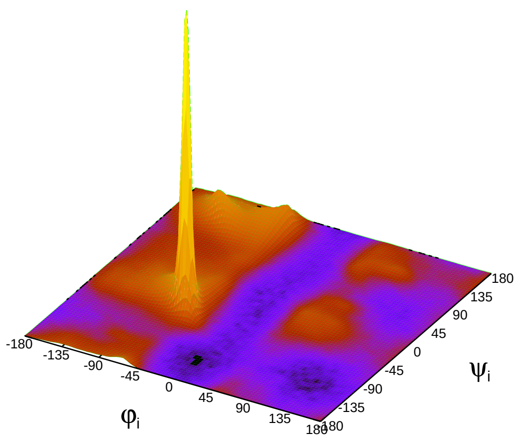
$$\alpha = \arg(-a \cdot c + (a \cdot b)(b \cdot c), a \cdot (b \times c))$$

An (ideal) angle of 180° is in chemistry called a *trans* conformation, where 0° is called *cis*.

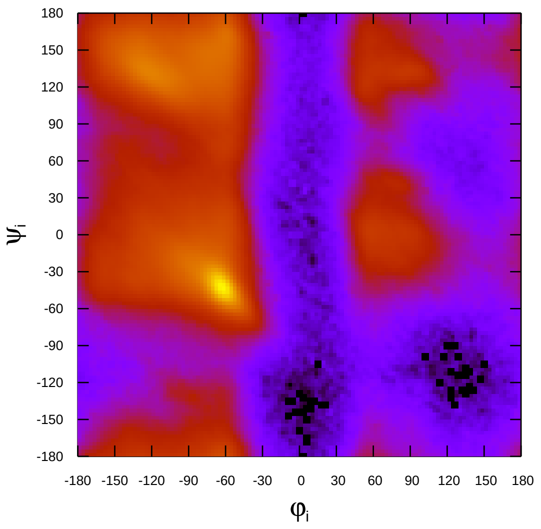
4.5.3 Angle Distributions

We provide now a short visual characterization of the distributions for several (real and virtual) bond and torsion angles. These are, to the best of our knowledge, the first large-scale and high-resolution histogram plots of the entire PDB. They might be of a more general interest to the structural genomics and proteomics community. The plots are in good agreement with the plots published so far (see, for instance, [Lev76, OH94, HSV97, Kle97, LDA$^+$03, SMB04, LC05, AWCJ05]). Much more insight could be given by plots that separate special residues like glycine or proline, or by plots showing histogram data for only one type of residue. (Since this is not the main matter of this work, we omit these figures.)

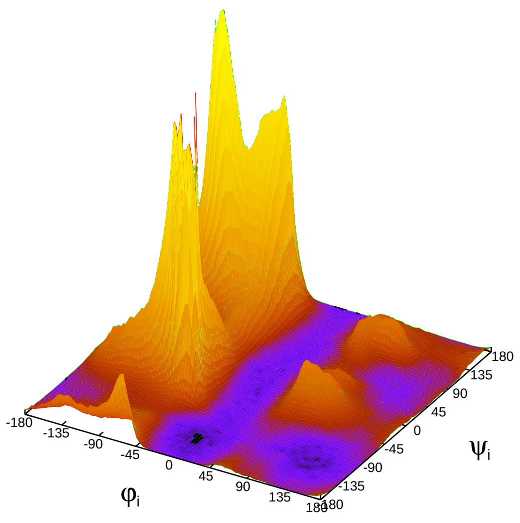
Figures 4.4 and 4.5 show histograms of the ψ and φ backbone torsion angles. While the first figure shows the relation between the angles having an equal index according to the numbering scheme of Figure 2.8(a) (both axes of rotation belong to the same amino acid part), the other figure shows the relation according to Figure 2.8(b) (both axes of rotation belong to the same C_α atom), which is analogous to the original Ramachandran plot [RRS63].



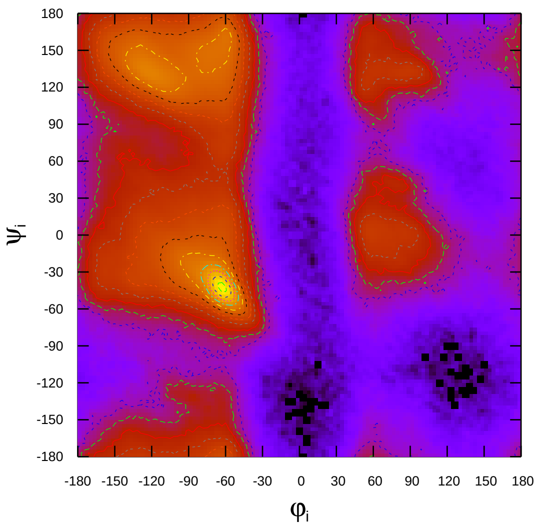
(a) 3D-histogram



(b) Histogram map

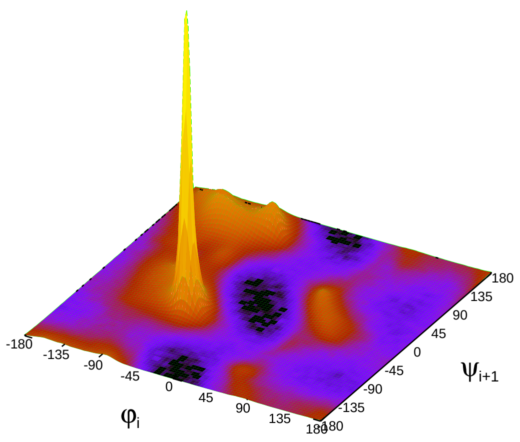


(c) Bounded 3D-histogram

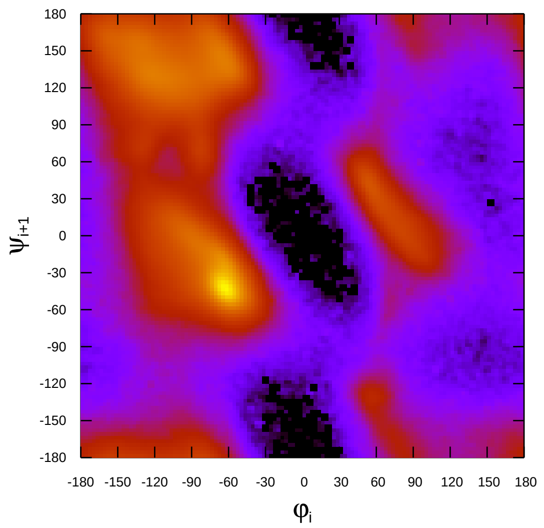


(d) Contour plot

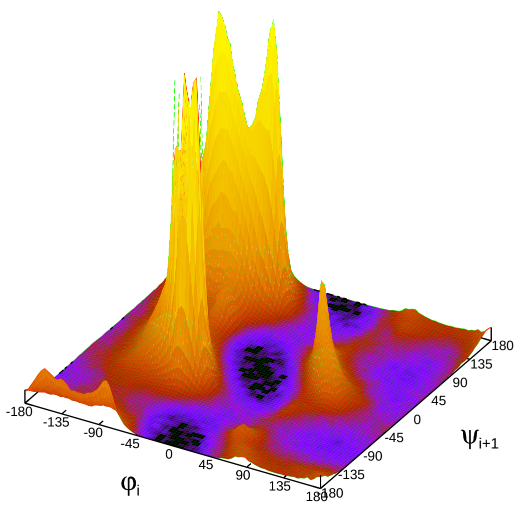
Figure 4.4: Torsion angle histogram of the PDB (φ_i vs. ψ_i). The major peak represents the α-helix, the minor peaks come from the β-strands/sheets and the left-handed helix.



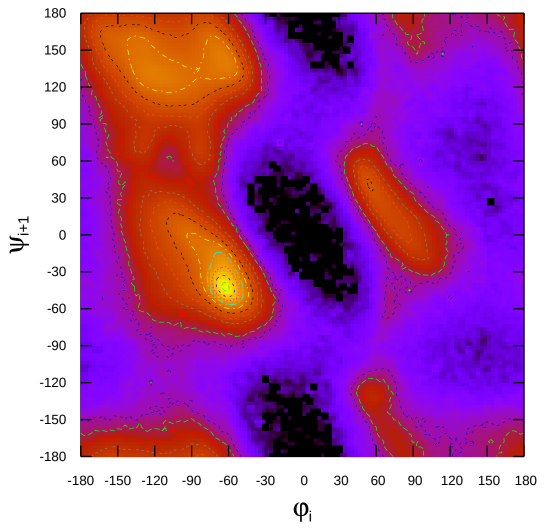
(a) 3D-histogram



(b) Histogram map

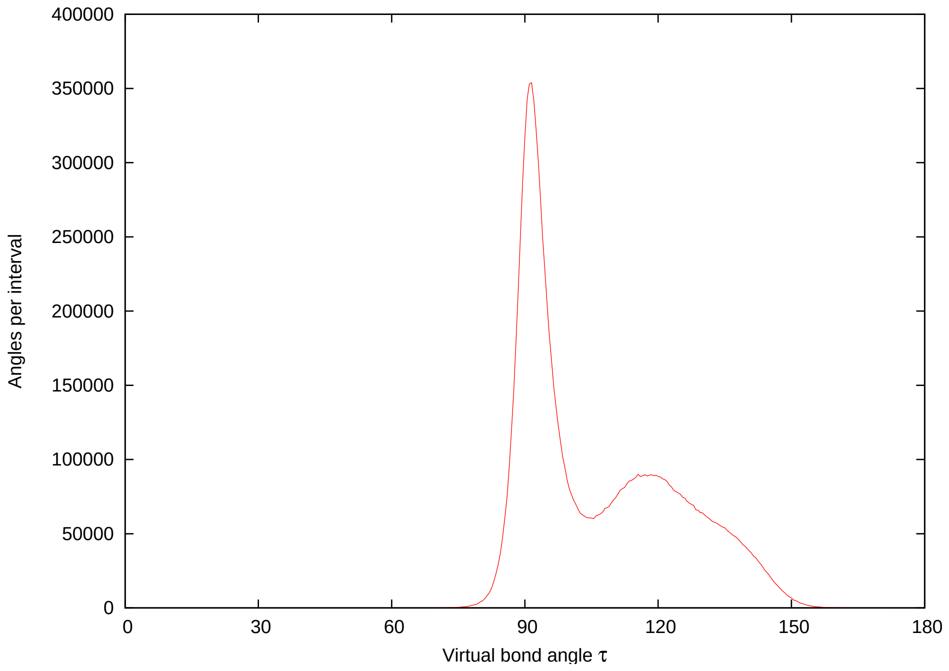


(c) Bounded 3D-histogram

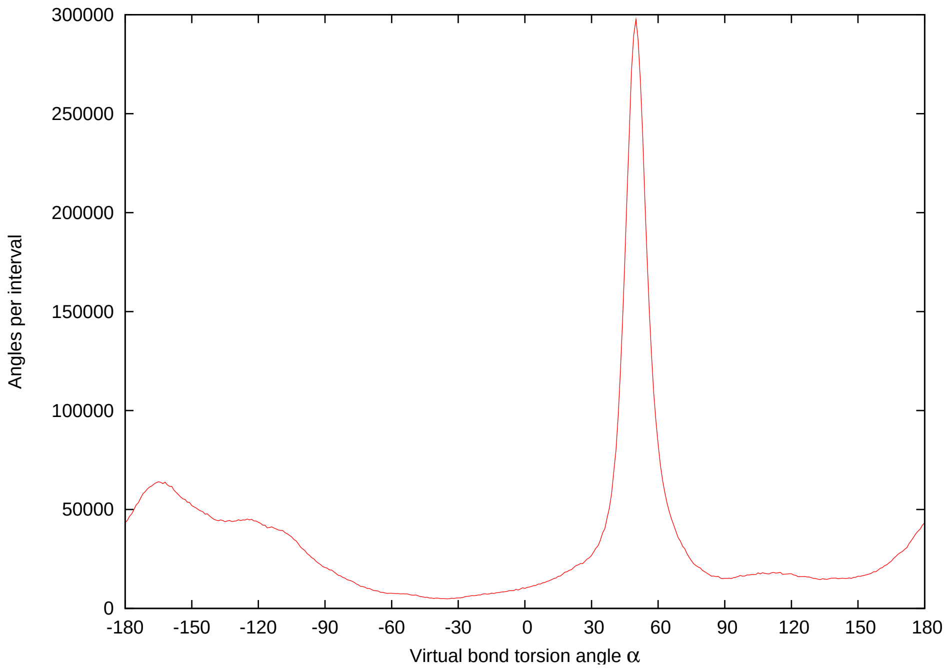


(d) Contour plot

Figure 4.5: Ramachandran-like angle histogram of the PDB (φ_i vs. ψ_{i+1})

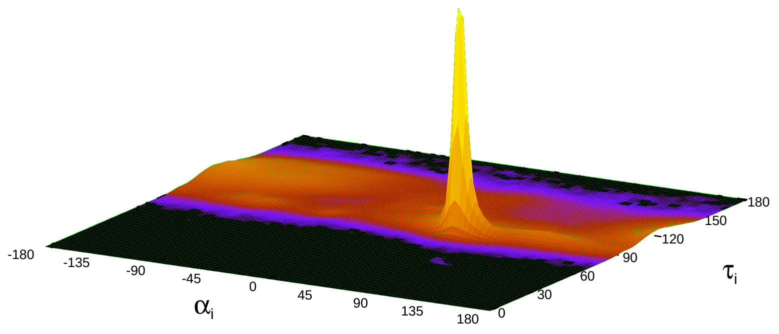


(a) Histogram of the virtual bond angle $\tau_i = \sphericalangle(C_i^\alpha, C_{i+1}^\alpha, C_{i+2}^\alpha)$

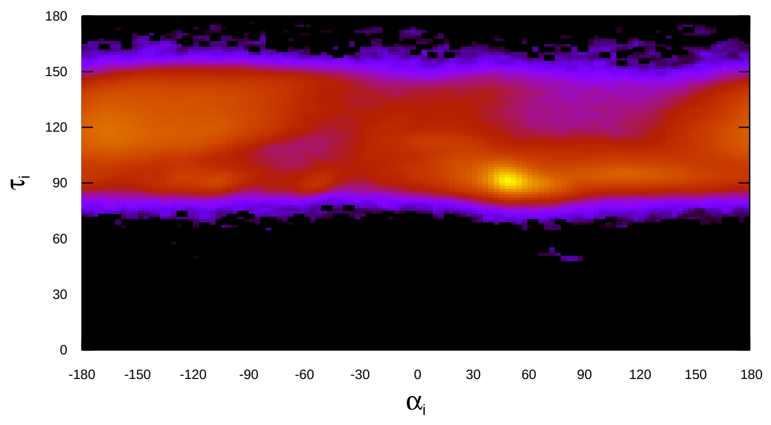


(b) Histogram of the virtual torsion angle $\alpha_i = \circlearrowright(C_i^\alpha, C_{i+1}^\alpha, C_{i+2}^\alpha, C_{i+3}^\alpha)$

Figure 4.6: Histograms of the virtual C_α-bond and torsion angles of the PDB.

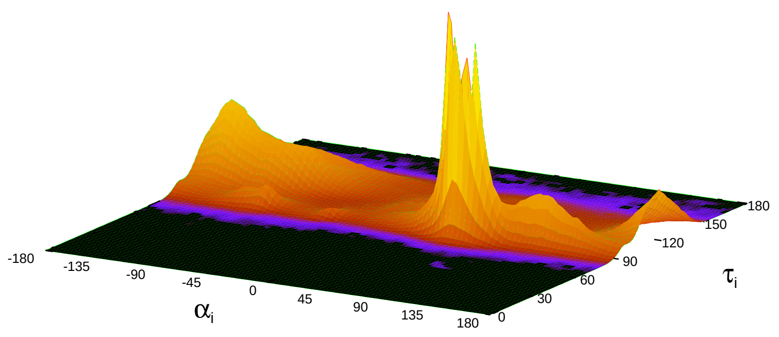


(a) Surface histogram of (τ_i, α_i)

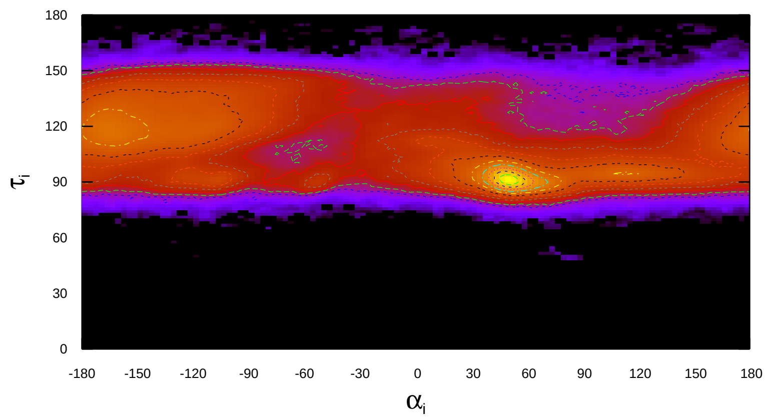


(b) Color plot histogram of (τ_i, α_i)

Figure 4.7: Histogram of the virtual bond angles (τ_i, α_i) in the PDB using a logarithmic color scheme.

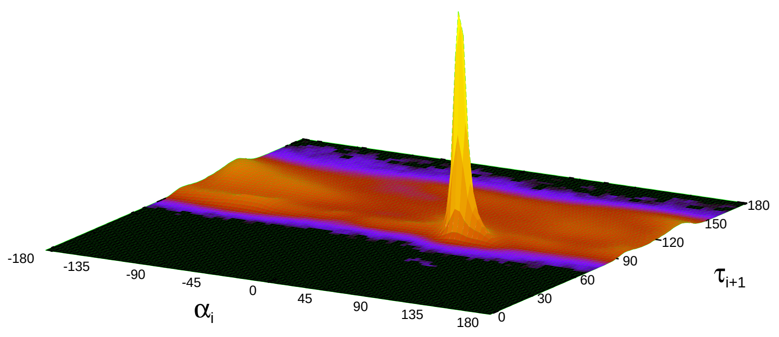


(a) Bounded surface histogram of (τ_i, α_i)

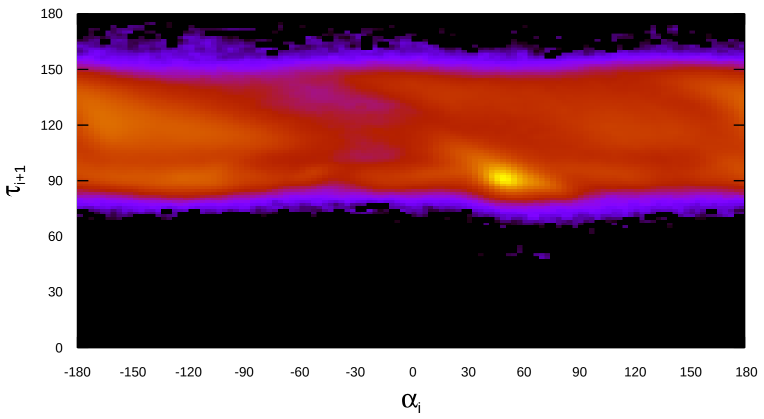


(b) Contour plot of (τ_i, α_i)

Figure 4.8: Histogram of the virtual bond angles (τ_i, α_i) in the PDB using a logarithmic color scheme (continued).

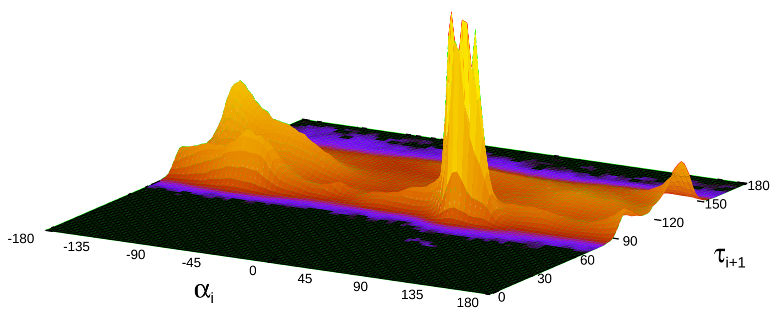


(a) Surface histogram of (τ_{i+1}, α_i)

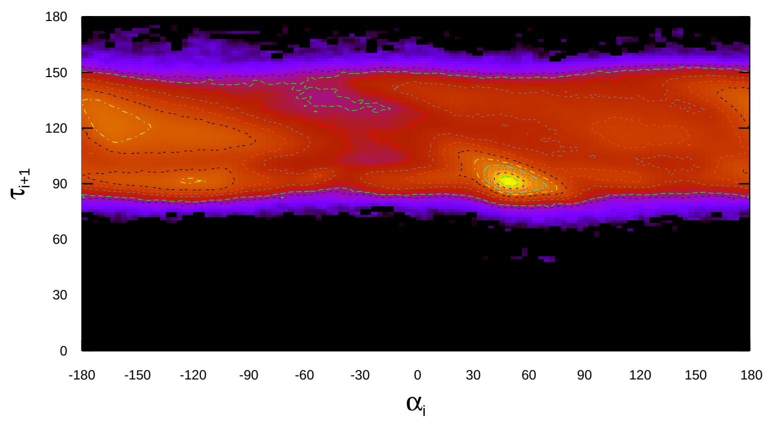


(b) Color plot histogram of (τ_{i+1}, α_i)

Figure 4.9: Histogram of the virtual bond angles (τ_{i+1}, α_i) in the PDB.



(a) Bounded surface histogram of (τ_{i+1}, α_i)



(b) Contour plot of (τ_{i+1}, α_i)

Figure 4.10: Histogram of the virtual bond angles (τ_{i+1}, α_i) in the PDB.

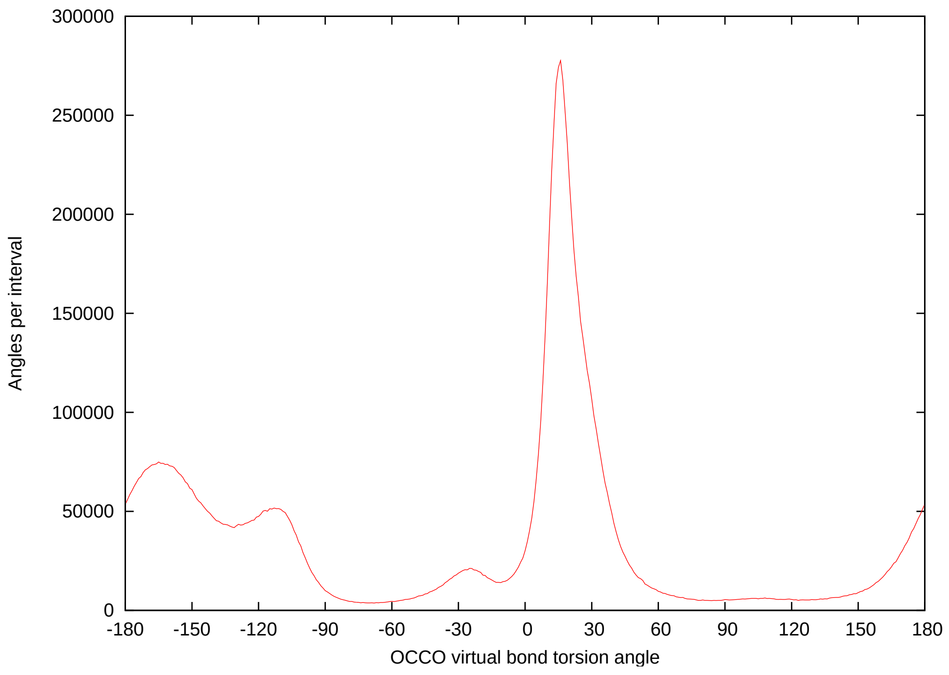
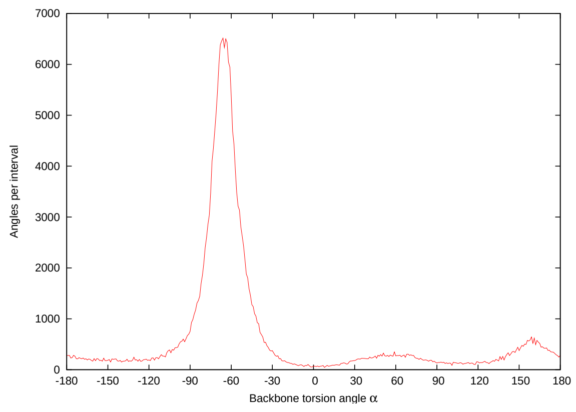
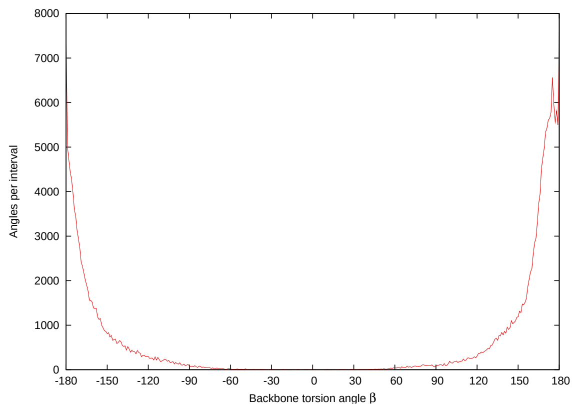


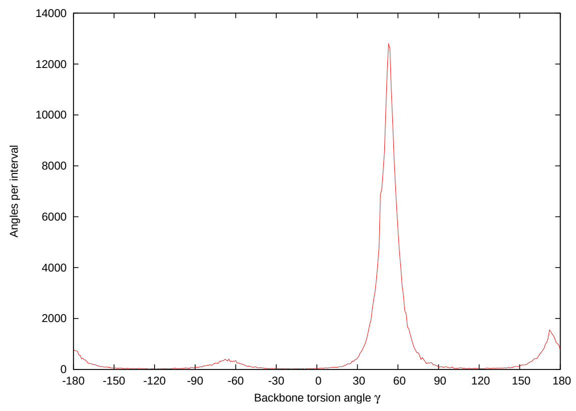
Figure 4.11: Histogram of the $OCCO$ virtual torsion angles in the PDB.



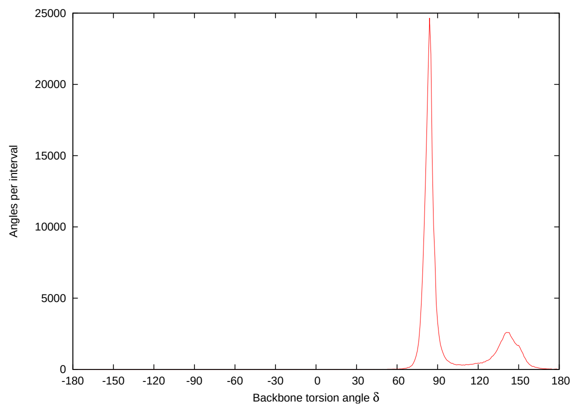
(a) Histogram of torsion angle α



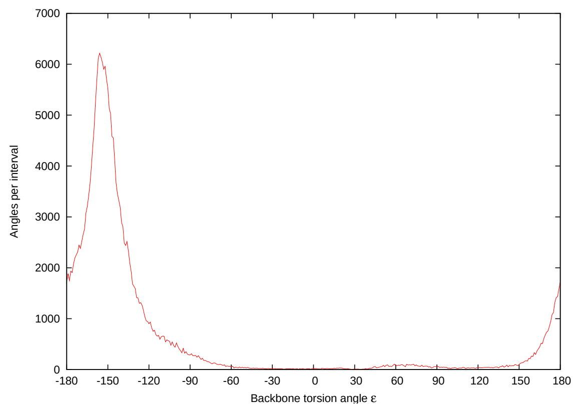
(b) Histogram of torsion angle β



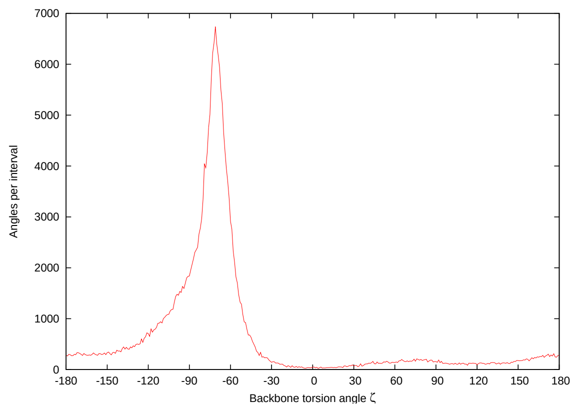
(c) Histogram of torsion angle γ



(d) Histogram of torsion angle δ



(e) Histogram of torsion angle ε



(f) Histogram of torsion angle ζ

Figure 4.12: Histogram of the backbone torsion angles α, β, γ, δ, ε, and ζ of the nucleic acids contained in the PDB.

4.6 Measures of Protein Similarity

We will now briefly discuss the different measures that can be used to describe the similarity of macromolecular biopolymers in general, and in particular of proteins. We usually define the overall distance of two abstract structures (sequences of characters or numbers) in terms of the distances of their constituting elements and operations such as comparisons, substitutions, insertions, and/or deletions. Most of the times, we assume the *distance function* $d(x, y)$ (of sequences or their building blocks) to fulfill the properties of a *metric*, that is, for all assignments to the variables x and y the distance function

- gives a nonnegative value $d(x, y) \geq 0,$
- is symmetric $d(x, y) = d(y, x),$
- obeys the triangle inequality $d(x, y) + d(y, z) \geq d(x, z),$
- gives distance zero for identical arguments $d(x, x) = 0,$
- as well as distance zero implies arguments identity $(d(x, y) = 0) \Rightarrow (x = y).$

Sometimes the last condition is dropped. In this case, $d(x, y)$ is called a *pseudometric*.

Please note, that each of the distance functions can be associated with a corresponding *similarity function*. Thus, distance and similarity can usually be regarded as complementary concepts.

4.6.1 String-Based Similarity Measures

The most widely used similarity measures for biological macromolecules (proteins and nucleic acids) are based on the sequence of their constituting monomers. For two such sequences s and t that are equal in size, the *Hamming distance* [Ham50] counts the number of mismatch positions i: $s_i \neq t_i$.

The *edit distance*, also called *Levenshtein distance* [Lev65], allows insertions and deletions. Hence this measure can be used for calculating the similarity of sequences having different lengths.

4.6.2 Distance-Based Similarity Measures

Root-Mean-Square Distance

The root-mean-square distance (RMSD) of atom positions is by far the most popular (dis)similarity measure for comparing protein structures. It requires a superposition of both structures in the same coordinate system. If the comparison candidates are equal in length and sequence the distance for a certain superposition can be evaluated straightforward in the following way:

$$\begin{aligned} RMSD(A,B) &= \sqrt{\frac{1}{n}\sum_{i=1}^{n} d(a_i,b_i)^2} \\ &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(b_{i,x}-a_{i,x})^2+(b_{i,y}-a_{i,y})^2+(b_{i,z}-a_{i,z})^2} \end{aligned}$$

where $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$ are the sequences of corresponding (three-dimensional) atom positions a_i and b_i, respectively, and n is the number of atoms of each sequence.

An optimal superposition of the two structures is now defined as any superimposition of A and B with minimum root-mean-square distance. The optimal root-mean square distance RMSD* is consequently defined as

$$\begin{aligned} \mathrm{RMSD}^*(A,B) &= \min_{R,t} \mathrm{RMSD}(A, R(B-t))^2 \\ &= \min_{R,t} \sqrt{\sum_{i=1}^{n} d(a_i, R(b_i - t))^2} \end{aligned}$$

where $R \in R^{3\times 3}$ is a rotation matrix and $t \in R^3$ is a three-dimensional translational displacement vector.

This optimum three-dimensional alignment is always a superimposition where the center of gravity for the first structure coincides with the center of gravity for the second structure. This can be proven as follows:

Ignoring the square root and the constant factor do not change the minimum, thus we must minimize the term

$$\begin{aligned} n \cdot RMSD^*(A,B)^2 &= \min_{R,t} RMSD(A, R(B-t))^2 \\ &= \sum_{i=1}^{n} d(a_i, R(b_i - t))^2 \\ &= \sum_{i=1}^{n} (a_i - R(b_i - t))^T (a_i - R(b_i - t)) \end{aligned}$$

Without changing the value, the identity matrix **I** is introduced

$$= \sum_{i=1}^{n} (a_i - R(b_i - t))^T \mathbf{I} (a_i - R(b_i - t))$$

For rotation matrices, the inversion is equal to the transposition: $R^{-1} = R^T$ or $RR^t = \mathbf{I}$

$$= \sum_{i=1}^{n} (a_i - R(b_i - t))^T (RR^T)(a_i - R(b_i - t))$$

Since matrix multiplication is associative

$$= \sum_{i=1}^{n} ((a_i - R(b_i - t))^T R)(R^T (a_i - R(b_i - t)))$$

By $(AB)^T = B^T A^T$ we get

$$\begin{aligned} &= \sum_{i=1}^{n} (R^T a_i - b_i + t)^T (R^T a_i - b_i + t) \\ &= \sum_{i=1}^{n} (t - b_i + R^T a_i)^T (t - b_i + R^T a_i) \\ &= \sum_{i=1}^{n} (t - (b_i - R^T a_i))^T (t - (b_i - R^T a_i)) \\ &= \sum_{i=1}^{n} \| t - (b_i - R^T a_i) \|^2 \end{aligned}$$

Now consider a fixed rotation matrix R, and optimize the translation vector t, i.e. we want to know (setting $c_i = b_i - R^T a_i$)

$$\begin{aligned} & \min_{t \in \mathbb{R}^3} \sum_{i=1}^{n} \| t - c_i \|^2 \\ = & \min_{t \in \mathbb{R}^3} \sum_{i=1}^{n} \sqrt{(t_x - c_{i,x})^2 + (t_y - c_{i,y})^2 + (t_z - c_{i,z})^2}^{\,2} \\ = & \min_{t \in \mathbb{R}^3} \sum_{i=1}^{n} \left[(t_x - c_{i,x})^2 + (t_y - c_{i,y})^2 + (t_z - c_{i,z})^2 \right] \\ = & \min_{t \in \mathbb{R}^3} \left[\sum_{i=1}^{n} (t_x - c_{i,x})^2 + \sum_{i=1}^{n} (t_y - c_{i,y})^2 + \sum_{i=1}^{n} (t_z - c_{i,z})^2 \right] \end{aligned}$$

Setting the first deviation to zero yields the translation values of the minimum:

$$t_x = \frac{1}{n}\sum_{i=1}^{n} c_{i,x} \quad t_y = \frac{1}{n}\sum_{i=1}^{n} c_{i,y} \quad t_z = \frac{1}{n}\sum_{i=1}^{n} c_{i,z}$$

After superimposing the centroids of both structures, the optimum can be found by systematically trying rotations about the three axes. Since rotations cause larger distances for atoms distant from center, it could be useful to consider the atoms in order of decreasing distance from the center.

If the structures differ in their sequence of amino acids, it is not possible to find an equivalent partner for each atom of the side chains. Therefore, it is reasonable to replace each residue by a single virtual atom at the respective center of gravity or to consider only the backbone atoms of both structures.

A major disadvantage of the RMSD measure appears in the case, where both sequences contain a different number of residues. Then, a mapping of equivalent atoms is needed. Another proposition was provided by Carugo and Pongor [CP01], see also [CP02].

For the computation of optimal superpositions and the respective RMSD value, several methods have been published.

Diamond [Dia66] described a procedure for finding a suitable rotation to superpose two structures, but it did not necessarily find the minimum.

McLachlan [McL72] presented an iterative solution where successively rotations are applied. First, a steepest-descent method, according to Jacobi's diagonalization method for real symmetric matrices, is applied. In a second phase, it is switched over to a Newton-Raphson method. An analytic solution is presented too.

Kabsch [Kab76] proposed a direct solution using Lagrange multipliers. Sometimes the method produced improper rotations, which was fixed in a later paper [Kab78].

Ferro and Hermans [FH77] described how the problem of approximating the axis-rotation matrix can be solved iteratively without changing the coordinates. After extracting the inherent information from the coordinates, the effort spent on the approximation is the same for small and large vector sets.

Mackay [Mac84] proposed to use a unit quaternion instead of the 3×3 matrix for representing the rotation. The advantage is that the resulting equations to be solved are linear. A similar proposal had been made by Faugeras and Hebert at a conference in 1983.

Arun, Huang, and Blostein [AHB87] presented a noniterative algorithm that utilizes the Singular Value Decomposition (SVD) of a 3×3 matrix.

Extensive work on a unit quaternion-based and an orthonormal matrices-based solution to the problem was written by Horn [Hor87a, Hor87b].

Diamond[Dia88] showed how the analysis of the problem can be simplified by casting the algebra in terms of only half the required angle of rotation. This led to an unconstrained eigenvalue/vector problem of order 4, or to scalar iteration using the inversion of a third order matrix. A related method using quaternion algebra was published by Kearsley [Kea89]. It requires only the diagonalization of a 4×4 symmetric matrix and, unlike some of the other approaches, it always produces proper rotations, and no special cases have to be considered.

Further work on the problem was published by UMEYAMA [Ume91], by COUTSIAS, SEOK, DILL [CSD04], by THEOBALD [The05], and others [Dia76, CL86, Brü03, JD05].

Other Distance-Based Measures

Other distance-based similarity measures are mostly based on intra-molecular distance matrices, such as the root-mean-square deviation of the corresponding interatomic distances (see LEVITT [Lev76] and SIPPL [Sip82]):

$$D(\overline{S}, S) = \frac{1}{n}\sqrt{\sum_{i,j}(\overline{d}_{i,j} - d_{i,j})^2}$$

SIPPL also defined another measure, that is based on the entries of the k-th order side diagonals of the intramolecular distance matrix:

$$D_k(\overline{S}, S) = \sqrt{\frac{1}{n-k}\sum_{i=1}^{N-k}(\overline{d}_{i,i+k} - d_{i,i+k})^2}$$

The d_k values for small k, that is near the main diagonal, indicate (dis)similarity of the local (secondary) structure, whereas the values for larger k refer to the similarity with respect to the more global (tertiary) structure.

A representation based on distance vectors of neighboring C_α carbons was proposed by CHEW et al. [CHKK99]. Further methods related to structure similarity are discussed in [CB99, BC01, LS04, Lot04, KL04, KP04].

4.6.3 Angle-Based Similarity Measures

Angle-based similarity measures are a straight forward approach, leading to slightly different concepts of similarity. While distance-based measures are mostly a kind of rigid-body superposition, angle-based measures provide inherent flexibility with respect to torsion around bonds. The answer to the question whether this is beneficial depends on the particular comparison problem.

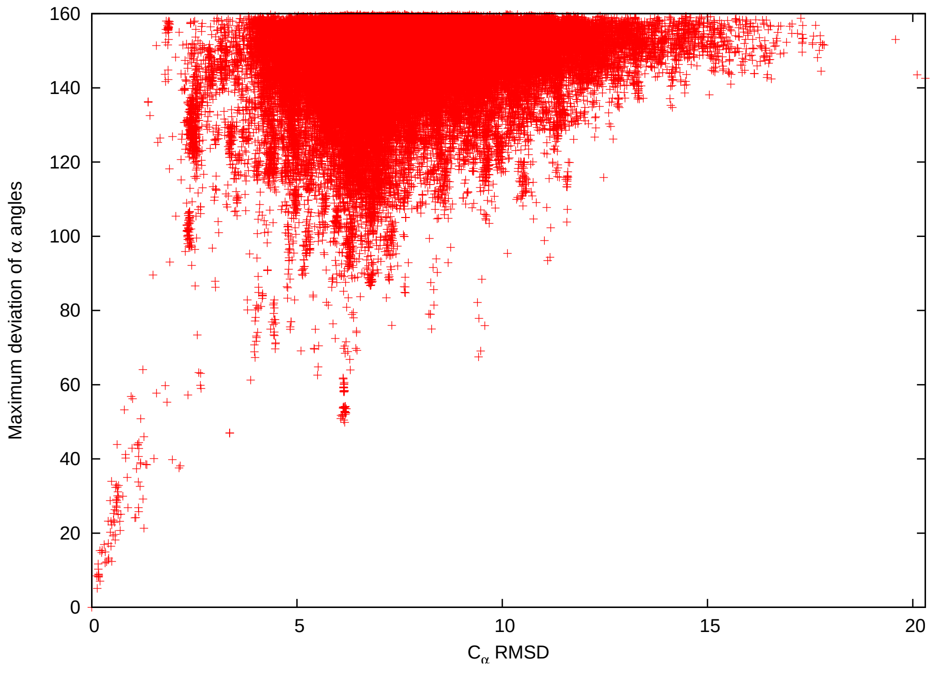
We investigated the relation between the RMSD of an optimal superimposition and the value of different torsion-angle based measures, that vary in the norm used to compute a kind of 'average' angle deviation. In particular, these measures are:

1. the maximum angle deviation (∞-norm),

2. the root-mean-square deviation of angles (2-norm),

3. the average deviation of angles (1-norm), and

4. the 0.5-norm of the angle deviations.



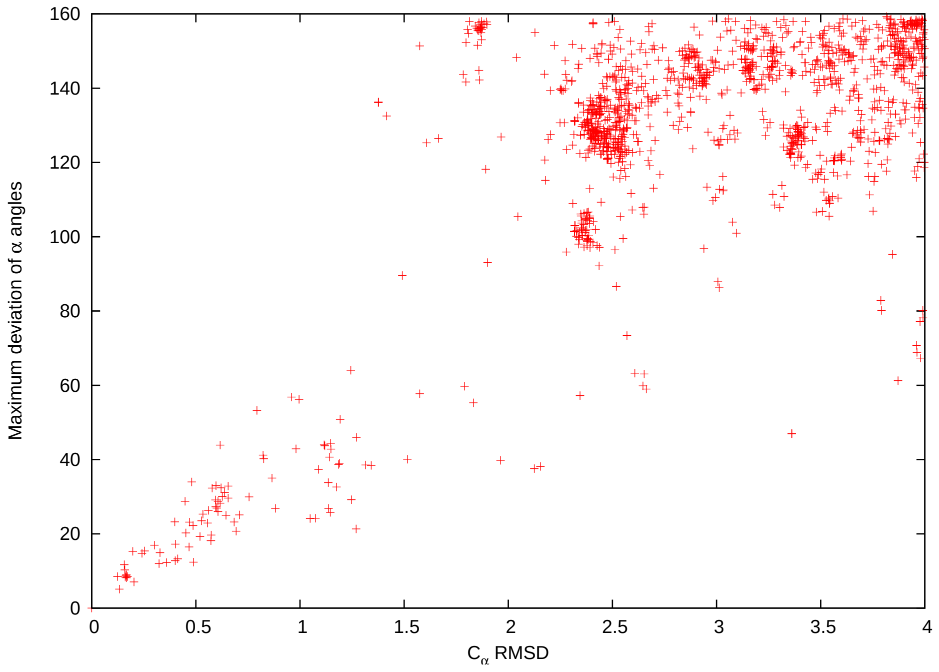


Figure 4.13: Comparison of the coordinates RMSD and the maximum angle deviation.

Symbol	Virtual	Type	Atoms
Polypeptides			
ψ_i	no	torsion angle	$\circlearrowright (N_i, C_i^\alpha, C_i', N_{i+1})$
ω_i	no	torsion angle	$\circlearrowright (C_i^\alpha, C_i', N_{i+1}, C_{i+1}^\alpha)$
φ_i	no	torsion angle	$\circlearrowright (C_i', N_{i+1}, C_{i+1}^\alpha, C_{i+1}')$
τ_i	yes	bond angle	$\sphericalangle(C_i^\alpha, C_{i+1}^\alpha, C_{i+2}^\alpha)$
α_i	yes	dihedral angle	$\circlearrowright (C_i^\alpha, C_{i+1}^\alpha, C_{i+2}^\alpha, C_{i+3}^\alpha)$
	yes	dihedral angle	$\circlearrowright (O_i, C_i', C_{i+1}', O_{i+1})$
Nucleic acids			
α_i	no	torsion angles	$\circlearrowright (O3_{i-1}', P_{i-1}, O5_i', C5_i')$
β_i	no	torsion angles	$\circlearrowright (P_{i-1}, O5_i', C5_i', C4_i')$
γ_i	no	torsion angles	$\circlearrowright (O5_i', C5_i', C4_i', C3_i')$
δ_i	no	torsion angles	$\circlearrowright (C5_i', C4_i', C3_i', O3_i')$
ϵ_i	no	torsion angles	$\circlearrowright (C4_i', C3_i', O3_i', P_{i+1})$
ζ_i	no	torsion angles	$\circlearrowright (C3_i', O3_i', P_{i+1}, O5_{i+1}')$

Table 4.2: Different types of variable angles. (Please note, we use the counting based numbering scheme, see Figure 2.8).

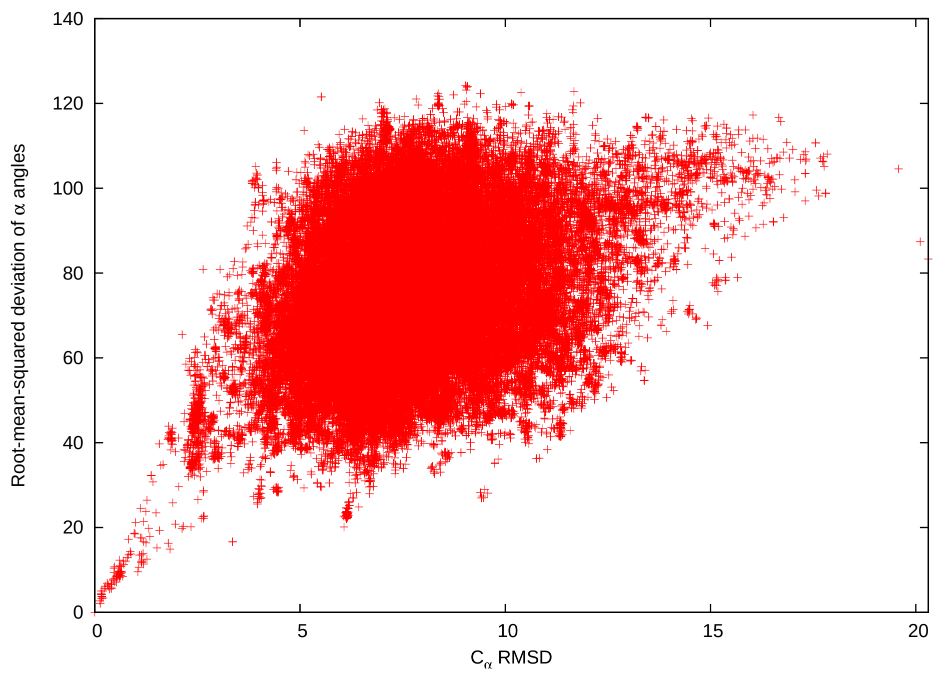
We used an exemplary search of a structure that was described by its α-angles to examine the relation to the RMSD measure. Figure 4.13 shows that there are hit candidate structures with quite acceptable C_α coordinate RMSD values (below 2.0) that exhibit extreme maximum α angle deviations of approximately 160°. But the picture changes dramatically when we switch over from the maximum (∞-)norm to the 2-norm (RMSD, see Figure 4.14), the 1-norm (average deviation, see Figure 4.15), or even the 1/2-norm (see Figure 4.16).

The root-mean-square deviation of ψ- and φ-angles had been used to compare protein structures by Remington and Matthews [RM80] and by Karpen et al. [KdHN89]. Other methods also used dihedral angles for structure comparison, for instance the work of Levitt [Lev76], Levine et al. [LSW84], and others [DFJZK94].

4.6.4 The Arithmetic String Distance

While string-based similarity measures like Hamming distance, Levenshtein distance, and weighted edit distance gained a lot of attention in the subject of pattern matching (in particular for nucleic and amino acid sequence comparison), the focus of structure comparison has always been on the distance-based measures.

In this work, we try to integrate both views by abstracting from an exact measurement to be encoded (for instance, an angle or a distance). To each measurement, we assign a certain category by discretizing the original continuous space. At the same time, we specialize the weighted edit distance to a distance function which has a certain arithmetic or geometric interpretation. The result is a compact code with an associated distance function that parallels the distance in the original metric of the continuous space. The



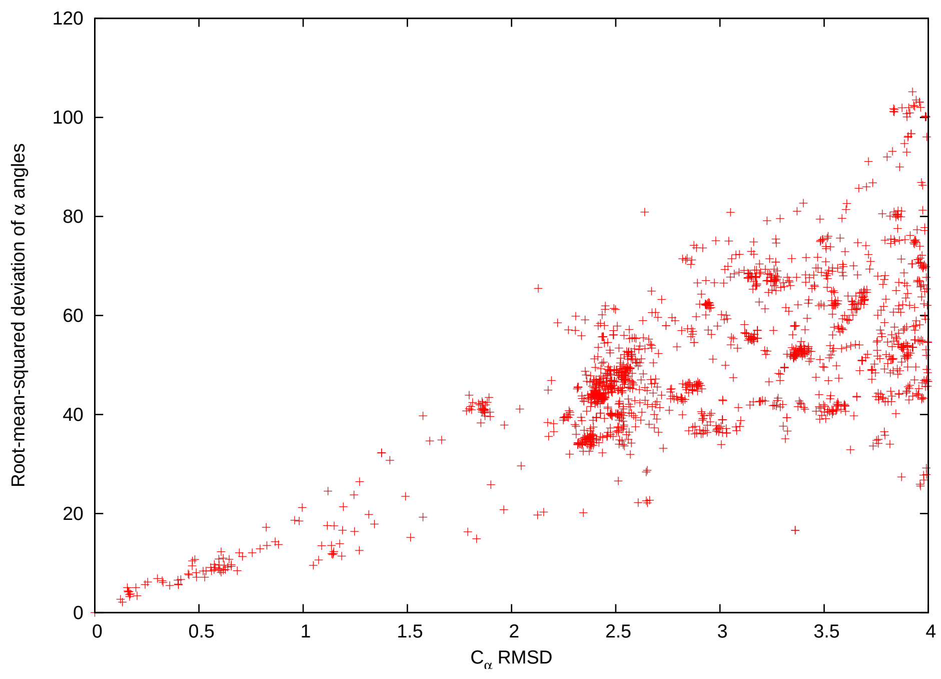


Figure 4.14: Comparison of the coordinates RMSD and the angles RMSD.

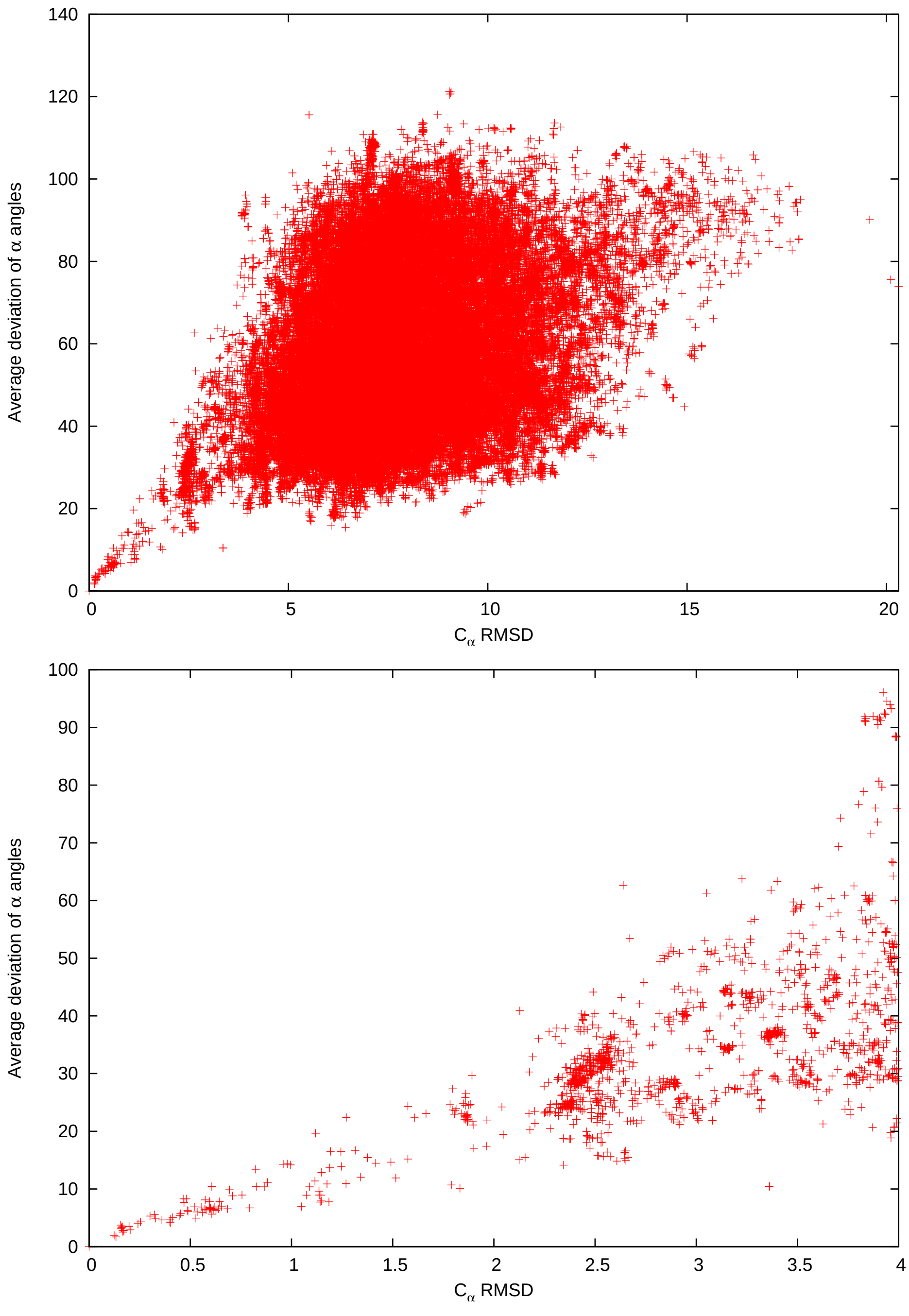


Figure 4.15: Comparison of the coordinates RMSD and the average angle deviation.

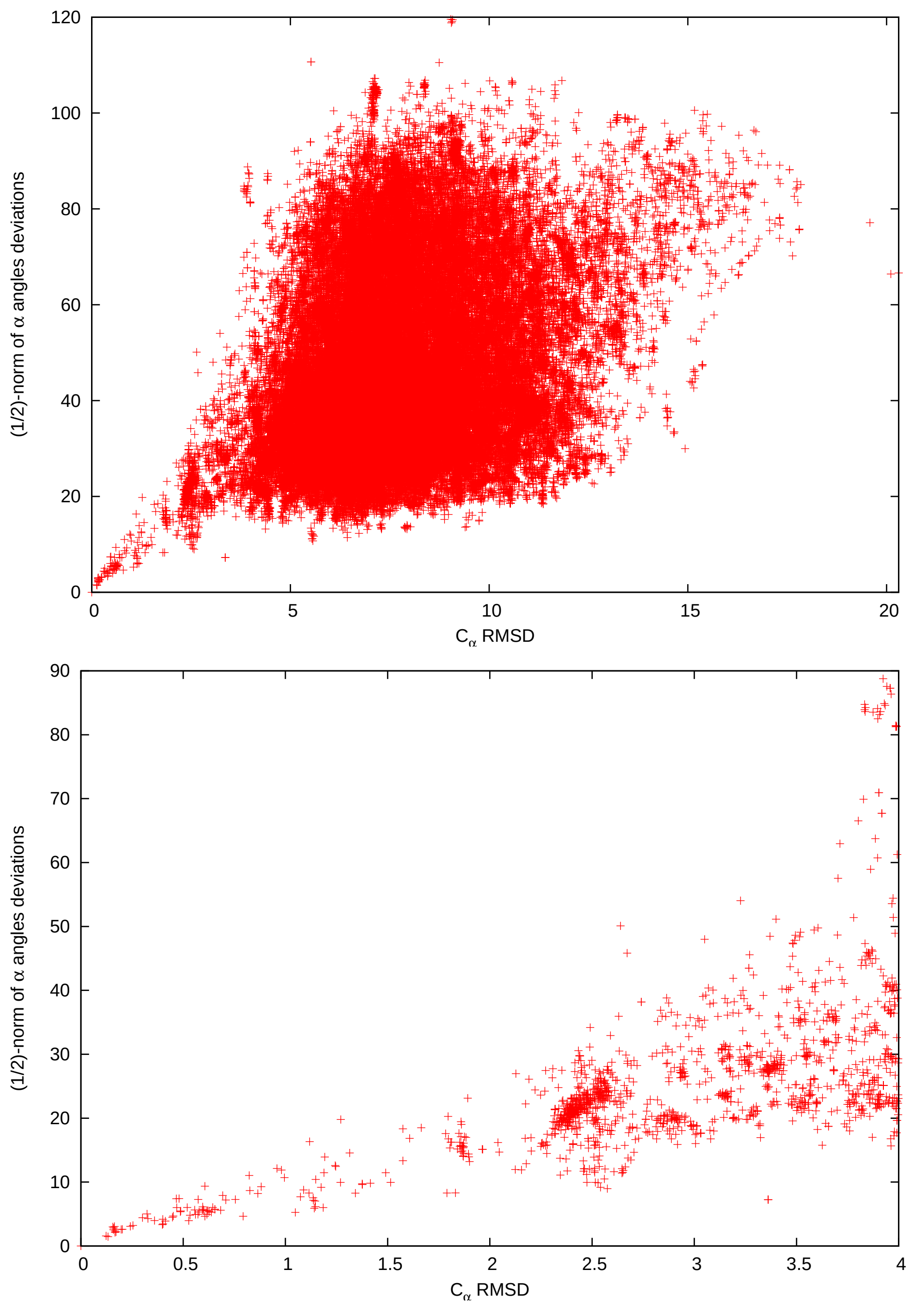


Figure 4.16: Comparison of the coordinates RMSD and the 1/2-norm angle deviation.

distance values are not as accurate as the original ones, but the discrete state allows a compact representation of common parts that are shared by several sequences. This enables the application of pattern matching algorithms, which were originally designed for alphabets of a more categorical nature.

4.7 Structure Searching via Encoded Backbones

We now turn to the application of generalized suffix trees to structure searching. First of all, we describe the creation of a suitable structure representation. Afterwards, we describe the concrete application to the PDB. We briefly describe the resulting indexing structure, and how exact searching can be performed using this data structure. Then we discuss the computation of longest matching substructures, and proceed with a description of deviation-tolerant searching. We conclude the section with a method for searching with insertions and deletions.

4.7.1 Less (Information) is More: The Structure Alphabet

Since the central idea of this thesis is to apply string indexing and pattern matching methods to searching structures and substructures, we need to find representations that encode structure in terms of an alphabet. This could easily be achieved by just taking the internal computer representation of the respective measure, e.g. the C++ or Java binary code for the `float` or `double` value of the actual measure. This provides already the basic precondition of discretizing the continuous measure space, yet the size of the alphabet (and thus the memory consumption) is very large. Moreover, there is surely doubt, whether two very close values should really be encoded by different characters. Consequently, we propose to partition the continuous measure space into several bins each of which a distinct character of the alphabet is assigned to. Please note, that this bin-based approach was already introduced by HOFFMAN [Hof96]. They used a restricted variant with equal-sized sectors of certain torsion angles for accelerating protein structure comparisons. We extend this approach to searching in structure databases (and thus parallel comparison) using arbitrary translation- and rotation-invariant measures with intervals of arbitrary size.

The possible measures include simple distances, bond angles, and dihedral angles. To simplify matters, we exploit the fact that the overall structure of the molecule is dominated by the conformation of the backbone. Thus, its atomic distances and angles make up the basis for the structural alphabet. A rather obvious representation of the backbone structure is the sequence of the torsion angles ψ, ω, and φ (see description in Section 2.2.3). As already mentioned in the introductory chapter, the angle that corresponds to the torsion around the peptide bonds adapts mostly the ideal trans-configuration (corresponding to $\pm 180°$). In rare cases, the cis-configuration ($0°$) occurs, see Figure 4.17, but the intervals $[-150, -10]$ and $[10, 150]$ are almost empty. This means the peptide bond torsion does not contribute much to distinguishing backbone structures since the average information content (the entropy) is very low. To save time and space, the resulting structure descrip-

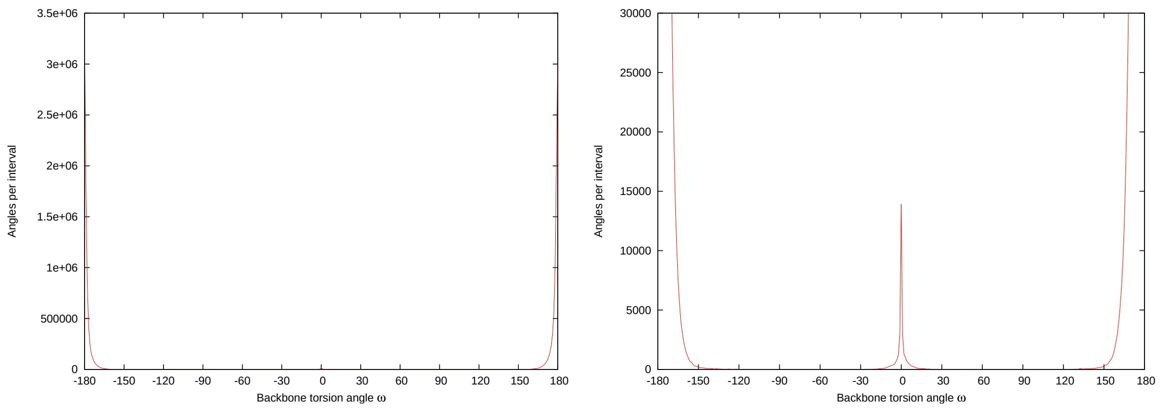


Figure 4.17: Histogram of the torsion around the peptide bond (ω).

	Polypeptides	Nucleic Acids
Positive PDB entries (files)	32.799	2.907
Files containing several MODELs	3.694	490
Positive chains	165.167	15.845
Computation time	55 min	16 min

Table 4.3: Statistical data of the angles extraction process for the PDB.

tion should contain only the values of the other backbone torsion angles ψ and φ. The respective histograms of the PDB are depicted in Figure 4.18.

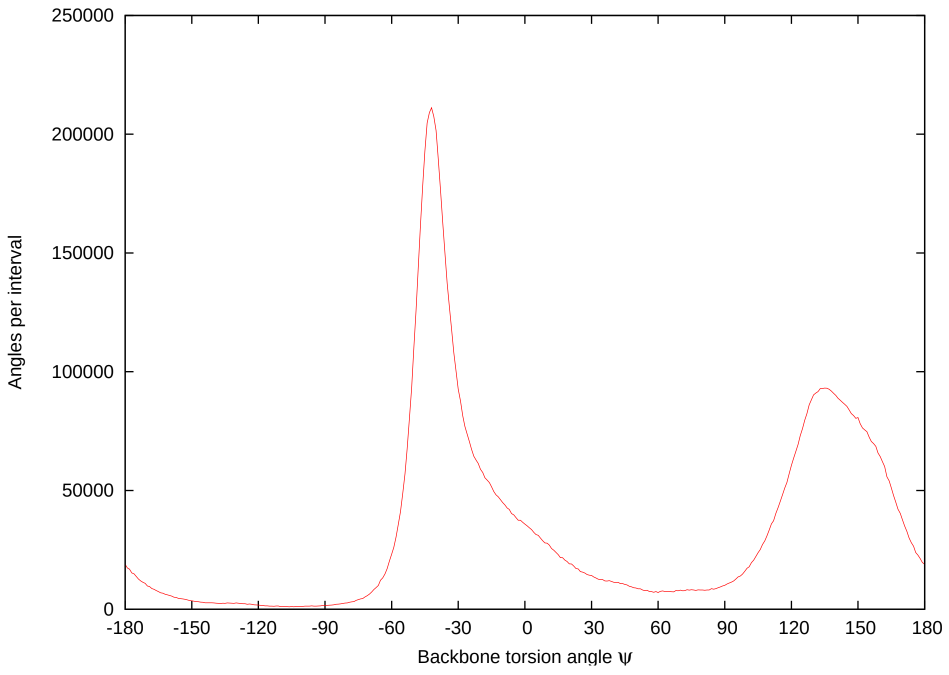
A more condensed description of the backbone structure can be obtained by an abstraction that ignores the existence of the N and C' atoms. Indeed, there is no need to consider the exact position of these atoms, since the side chains (which define the chemical properties at a certain position) are tied to the C_α atoms. Two 'neighboring' atoms $C_{\alpha,i}$ and $C_{\alpha,i+1}$ are considered as being connected by a *virtual bond*. In consequence of that, a backbone structure can be described by the dihedral angles defined by four quasi-consecutive C_α atoms (the *virtual bond torsion angles*) together with the angles defined by three (quasi-)consecutive C_α atoms.

For discretizing τ see [dlCML97].

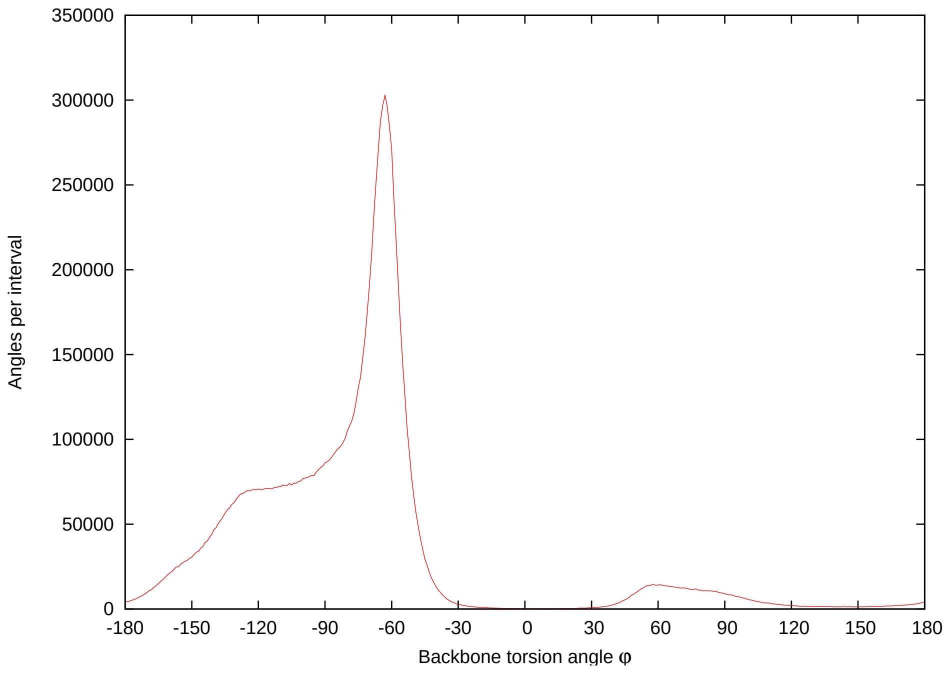
The whole circle from either 0° to 360°, or −180° to +180° can be divided into segments (not necessarily of the same size). Each sector is then encoded by a different character. This *discretization* of the angle space leads to a reasonable abstraction for the purpose of comparing or searching structures.

4.7.2 Construction of the Polypeptide Angles Suffix Tree

After all, we are now prepared to describe how an index of the protein structures in the PDB can be built. First of all, we loaded the current version of the 'database', that is all PDB-format text files, which are located at the `ftp` server of the RCSB. At the end



(a) Histogram of the backbone torsion angle $\psi_i = \circlearrowright(N_i, C_i^\alpha, C_i', N_{i+1})$.



(b) Histogram of the backbone torsion angle $\varphi_i = \circlearrowright(C_i', N_{i+1}, C_{i+1}^\alpha, C_{i+1}')$.

Figure 4.18: Histograms of the backbone torsion angles ψ and φ.

of 2005, these files had a total size of 5.3 gigabytes(!), in packed form, of course. After roughly three quarters of an hour for the decompression, the total consumption of disk space of the unpacked files was **22 GB**. After that, we parsed all files, computed the different types of angles (ψ, φ, α, etc.) and stored the angle sequences in separate files. This took another hour on our (comparatively old-fashioned) computer having a clock rate of 1 GHz. Fortunately, these steps have to be done only once per PDB file. Some context information like C_α carbon positions and amino acid sequences were extracted too. Of course, the same procedure applies to the nucleic acid structures and sequences. Some statistical data on the property extraction process is shown in Table 4.3.

Having the different sequences of angles at hand, we can now build a structure index by deciding

- which type of angles to use as a representation of the structure,
- which kind of entries to use for the index: all models, only one arbitrary model (e.g., the first), or the average structure of all models,
- which discretization scheme to use, that is
  - whether an equidistant discretization should be used or not,
  - which accuracy to use, that is, how many intervals or symbols the alphabet should comprise, and, consequently, how large the different intervals should be,
  - whether different alphabets should be used for encoding different kinds of angles (e.g., characters `a` to `m` for ψ, and characters `n` to `z` for φ).

Please, note that we use normal characters of the English alphabet for demonstration purposes only; the implementation, of course, is not limited to this set of symbols, and it usually starts with ASCII code 1, since code 0 is reserved for the sentinel.

After we made a decision, the respective angle sequences are successively retrieved from the hard disk, and discretized according to the defined bins. Finally, the resulting character sequences are inserted into a generalized suffix tree. Since we mostly use characteristic angles of polypeptides to encode their structure, we refer to this kind of GST as the *Polypeptide Angles Suffix Tree (PAST)*. Building the PAST for all protein structures contained in the PDB, takes (at present) between two and five minutes. The exact time actually depends on the type of angle sequence and on the number of discretization intervals. We do not show the graph of the construction time, since it does not matter whether this can be done within 5 minutes, or 30 seconds. Again, we emphasize the main matter of fast repeated searching, construction time does not matter within certain bounds.

4.7.3 Properties of the PAST

The size of the PAST (in terms of the number of nodes) as a function of the discretization accuracy (number of intervals) is depicted in Figure 4.19. The graph exhibits some similarity with the average size of a generalized suffix tree that is built from random sequences:

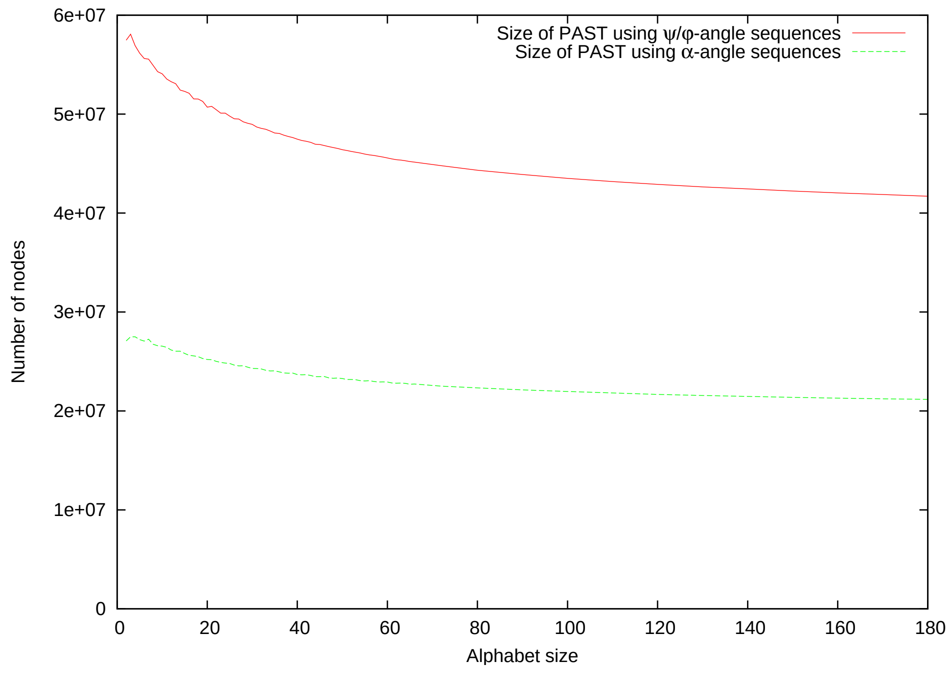


Figure 4.19: The size of the PAST (number of nodes) vs. the size of the alphabet.

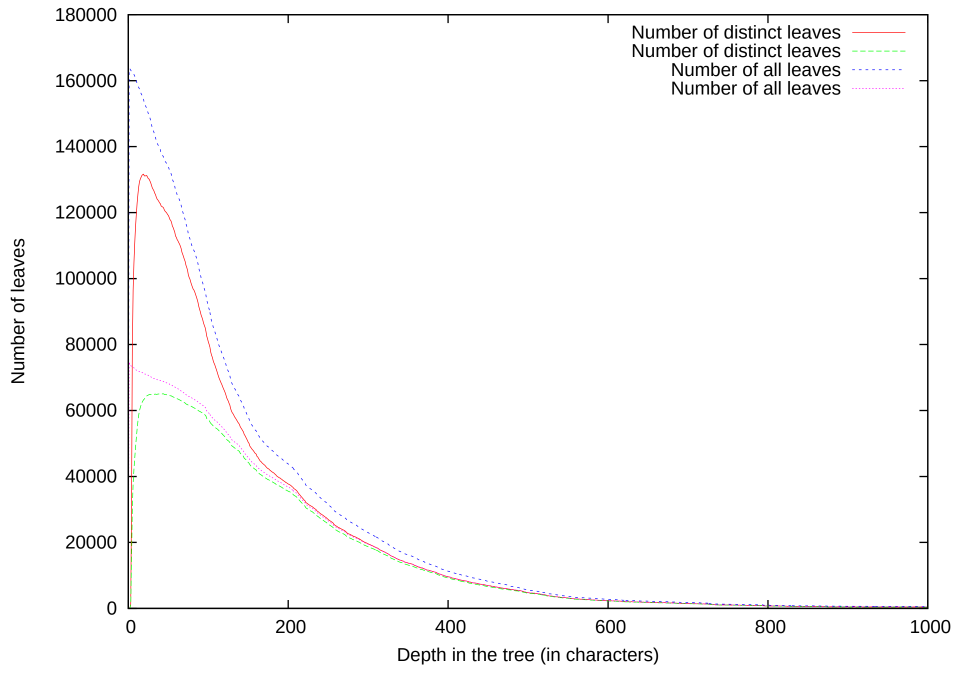
it shows some significant oscillations of decreasing frequency (for growing alphabet size). Formulas of these oscillation parameters and the asymptotical average case behavior were derived using Mellin transforms by Blumer et al. [BEH89].

For the case of several models of the same structure within a PDB entry, it is sometimes most convenient, to include only a single structure, preferably some kind of average or median structure. We show how to compute average structures in Section 5.1.2. Figures 4.20 and 4.21 compare this setting to the case where all models are contained in the PAST.

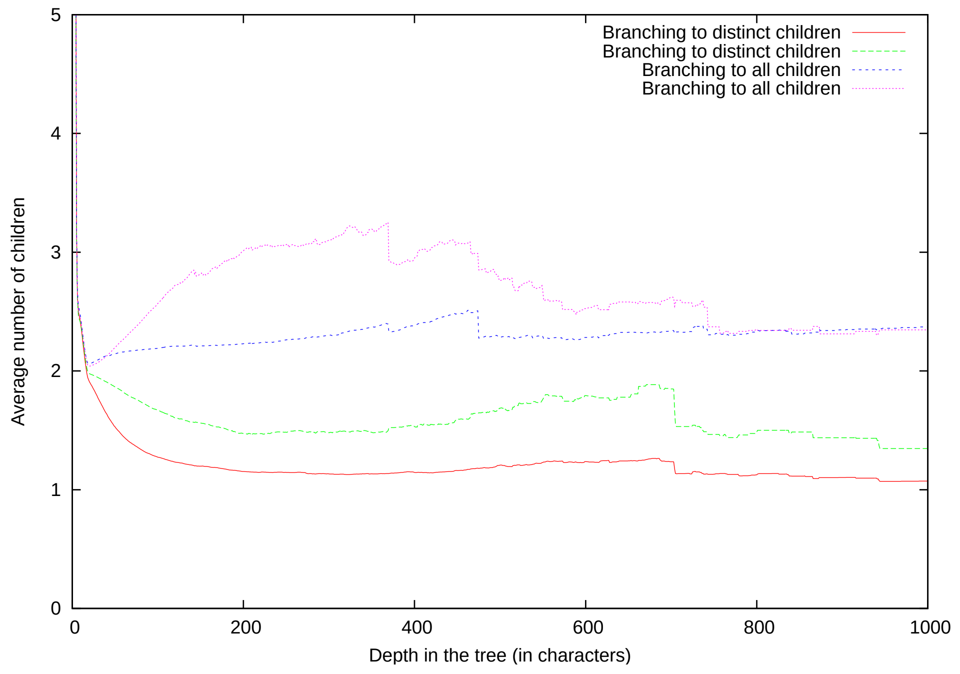
4.7.4 Exact Searching

Having the PAST for all angle sequences of a certain type at hand, it is easy to search for structure that are equal to a given reference structure. We just have to encode the reference as we did for the PDB structures. Afterwards, we search character by character using the standard suffix tree matching algorithm. At the end, we either have a complete match, or the search gets stuck. In the first case, we have at least one matching structure. The respective sequence and position can be found using the sequence identifier and the position field of the last inspected node. By traversing the whole subtree under the last inspected node, we can find all substructures contained in the PDB, that are encoded by the same sequence of characters, that is, whose angles fall into the same intervals. This means, some deviation is allowed for the structures to be retrieved.

At first glance, this seems to be an appropriate and very easy solution to the problem.

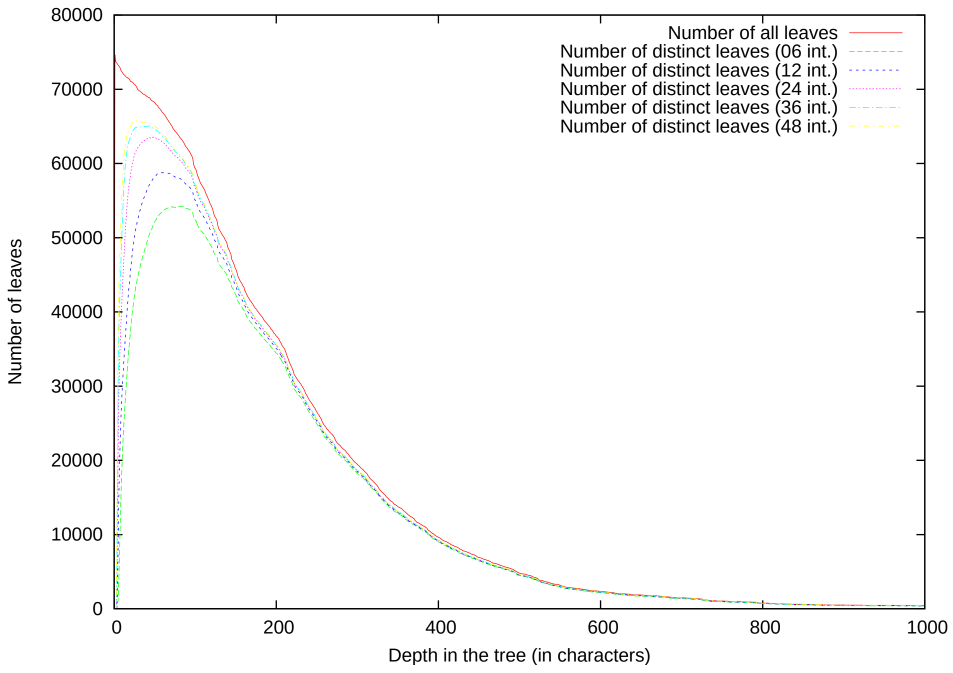


(a) The number of leaves for all models and for average structures only.

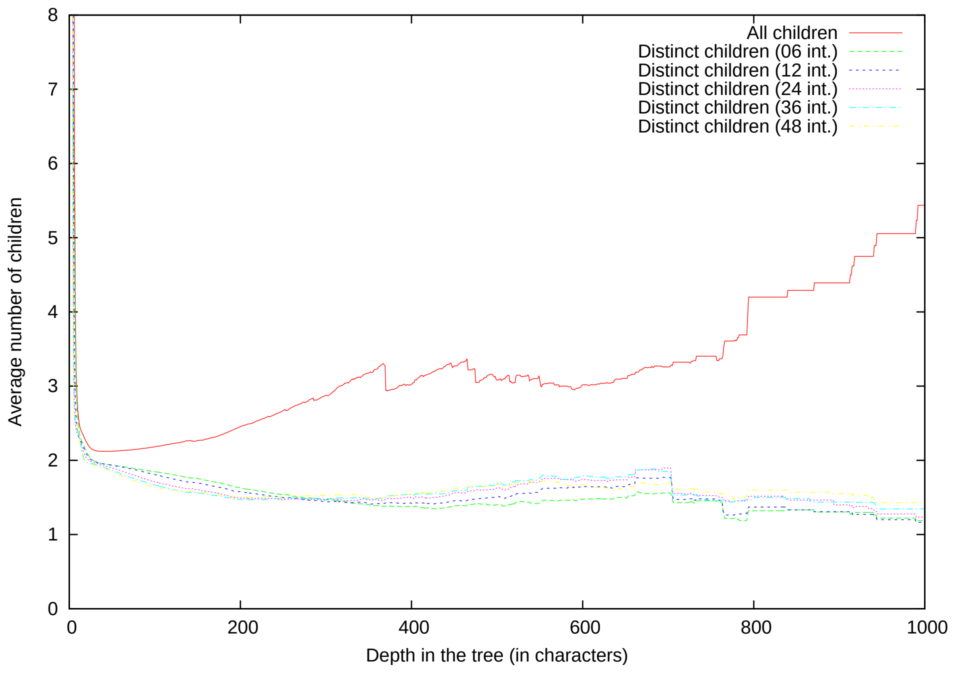


(b) The average number of children for all models and for average structures only.

Figure 4.20: Number of leaves and average branching degree of the PAST. This particular instance was built using an alphabet of 36 characters representing α-angle intervals of 10°.



(a) The number of leaves for different discretization intervals.



(b) The average branching degree for different discretization intervals.

Figure 4.21: Number of leaves and average branching degree for different alphabet sizes (using average structures for models of the same PDB entry). The alphabet (interval) sizes are: 6(60°), 12(30°), 24(15°), 36(10°), 48(7.5°).

As we will see, only the second predicate applies. We test the method by constructing the PAST for protein α-angles, using 36 discretization intervals of 10°. We take the search pattern from PDB entry `1a1f`, chain `A`, residues `137` to `157`, or short `1a1f(A)`:$137-157$. We get the result immediately (after 0.0 seconds), but it is a bit depressing: we found only two hits, and these represent exactly the entry where we took our search pattern from. We got two hits at the same position, because this chain contains two alternate location indicators at residue `123`. Thus, we did not find anything new.

We don't give up and build the same PAST, but now using 24 greater intervals of 15°. This time, we are lucky, finding an additional hit: `1a1k(A)`:$137-157$.

We repeat the experiment with 12 intervals of size 30°. Besides the two entries of `1a1f`, we find the following eight additional entries now: `1a1g(A)`, `1a1k(A)`, `1f2i(I)`, `1g2f(F)`, `1mey(C)`, `1mey(C)`, and `1llm(D)` (two times).

If we reduce the discretization accuracy to 10 intervals of 36°, a set of 29 substructures is returned. Surprisingly, a further reduction to 8 intervals of 45°reduces the size of the result set significantly. The complete results are shown in Table 4.4.

We find, that searching using the standard procedure is unsatisfactory. While at least one of the settings (e.g., alphabet size 10, interval size 36°, see Table 4.4) seems to produce acceptable results, even larger intervals may reduce the result set dramatically. We observe, that presence of a substructure in the result set for a certain discretization does not imply appearance of this hit in a result set for lower accuracy. Thus, the results to be expected are quite unpredictable, because a near-optimal discretization scheme cannot be determined in advance of the search. Also, the query definition is very restricted, since there is the same admissible interval size for each of the angle positions. A very unfortunate situation occurs if a search angle is near the boundary of its encoding interval; then the accepted variance is much greater in one direction compared to the other. What is left, is a very short response time. In any case, we could not measure any delay (0.0 seconds) of the answer. This is, of course, due to the linear worst-case time complexity (see Chapter 3).

4.7.5 Finding the Longest Common Substructure

The exact searching approach has one major advantage: For a given query structure p, the longest substructure, that is contained in the database, can be found very fast. The principle is almost as simple as exact searching itself. From the root node, we search for all suffixes of the query sequence using the standard matching procedure, starting with the complete string. If the search becomes stuck, we remember how many characters matched, and we search for the next suffix. We can save some effort if we do not start the search at the root again. Instead, we use the suffix link of the last matching (explicit) node. This pointer leads us to another explicit node that exactly matches the next suffix to the same position. If the mismatch occurred at an implicit node, only the first character of the edge has to be used for finding the correct child after traversing the suffix link. The search is continued with the character that produced the mismatch. We always keep track of the maximum length match. The traversal can be stopped, if the current suffix is shorter than the actual maximum.

Entry	Ch	Position	Md	Alt	Sequence	36	24	12	10	8
1a1f	A	137 157	1	A	CRICMRNFSRSDHLTTHIRTH	✓	✓	✓	✓	✓
1a1f	A	137 157	1	B	CRICMRNFSRSDHLTTHIRTH	✓	✓	✓	✓	✓
1a1g	A	137- 157	1		CRICMRNFSRSDHLTTHIRTH			✓	✓	
1a1h	A	137- 157	1	A	CRICMRNFSRSDHLTTHIRTH				✓	✓
1a1h	A	137- 157	1	B	CRICMRNFSRSDHLTTHIRTH				✓	✓
1a1i	A	137- 157	1	A	CRICMRNFSRSDHLTTHIRTH				✓	
1a1i	A	137- 157	1	B	CRICMRNFSRSDHLTTHIRTH				✓	
1a1j	A	137- 157	1	A	CRICMRNFSRSDHLTTHIRTH					✓
1a1j	A	137- 157	1	B	CRICMRNFSRSDHLTTHIRTH					✓
1a1k	A	137- 157	1		CRICMRNFSRSDHLTTHIRTH		✓	✓	✓	✓
1a1l	A	137- 157	1	A	CRICMRNFSRSDHLTTHIRTH				✓	✓
1a1l	A	137- 157	1	B	CRICMRNFSRSDHLTTHIRTH				✓	✓
1aay	A	137- 157	1	A	CRICMRNFSRSDHLTTHIRTH				✓	
1aay	A	137- 157	1	B	CRICMRNFSRSDHLTTHIRTH				✓	
1aay	A	165- 185	1	A	CDICGRKFARSDERKRHTKIH				✓	
1aay	A	165- 185	1	B	CDICGRKFARSDERKRHTKIH				✓	
1f2i	G	1137-1157	1		CRICMRNFSRSDHLTTHIRTH				✓	
1f2i	I	3137-3157	1		CRICMRNFSRSDHLTTHIRTH			✓		
1g2d	C	137- 157	1		CRICMRNFSQHTGLNQHIRTH				✓	
1g2d	C	165- 185	1		CDICGRKFATLHTRDRHTKIH				✓	
1g2d	F	237- 257	1		CRICMRNFSQHTGLNQHIRTH				✓	
1g2d	F	265- 285	1		CDICGRKFATLHTRDRHTKIH				✓	
1g2f	C	165- 185	1		CDICGRKFATLHTRTRHTKIH				✓	
1g2f	F	237- 257	1		CRICMRNFSQQASLNAHIRTH			✓		
1g2f	F	265- 285	1		CDICGRKFATLHTRTRHTKIH				✓	
1jk1	A	137- 157	1		CRICMRNFSRSDHLTTHIRTH				✓	
1llm	D	206- 226	1	A	CRICMRNFSRSDHLTTHIRTH			✓	✓	✓
1llm	D	206- 226	1	B	CRICMRNFSRSDHLTTHIRTH			✓	✓	✓
1mey	C	35- 55	1		CPECGKSFSQSSDLQKHQRTH			✓		
1mey	C	63- 83	1		CPECGKSFSRSDHLSRHQRTH			✓		
1p47	A	137- 157	1		CRICMRNFSRSDHLTTHIRTH				✓	
1p47	B	137- 157	1		CRICMRNFSRSDHLTTHIRTH				✓	
1x6h	A	50- 70	7		CSKCGKTFTRRNTMARHADNC				✓	
1zaa	C	37- 57	1		CRICMRNFSRSDHLTTHIRTH				✓	
1zaa	C	65- 85	1		CDICGRKFARSDERKRHTKIH				✓	

Table 4.4: Search test for the exact search.

As for the exact search, the worst-case time complexity of this operation is a linear function of the pattern length. This is due to the fact, that we have to follow at most $m = |p|$ matching (downward) edges, and the number of suffix link transitions is bounded by m too (there are only m suffixes). The number of matched characters is bounded by m, since we match each character only once. The number of mismatches is bounded by the number of traversed suffix links (at most m), and the number of child lookups (which is $\mathcal{O}(|\Sigma|) = \mathcal{O}(1)$ for a constant alphabet) is bounded by the total number of inspected nodes (at most $2m$). Ultimately, the overall worst-case complexity is $\mathcal{O}(m)$.

4.7.6 Tolerant Searching

In Section 4.7.4, we learned that searching using the standard matching procedure of suffix trees does only perform well regarding the response time, but not with respect to the quality of the results. Now, we describe how to improve the sensitivity of the search process while the query time is only slightly increased.

The main problem of exact searching is the fact that some of the query angles fall near the discretization bound of their respective interval. But, on average, we expect angles of related structures to spread to both sides of the query angles. If the related angle of a potential hit is on the wrong side, already a small deviation is sufficient to move it into the next discretization interval, while the difference of a whole interval size is needed to put it into the neighboring interval on the other side of the query angle. Hence, we are facing a complete lack of symmetry for the search tolerance, at least for extreme cases. Note that, already one mismatch discards a related structure. So we might lose a lot of potential hits.

To bring the search angles more to the center of the query range, we allow the potential hits to be in one of the two neighboring intervals. Thus, we have a fixed guarantee for the admissible deviation of the size of one interval. In extreme cases, the query angle falls again near one of the discretization boundaries of its interval. But now, the angle of a possibly related structure is allowed to differ by the size of one interval on the one hand side, and by two intervals on the other side. Yet, there is left a bit of asymmetry, but it seems to be better than in the exact case. The asymmetry can be further reduced by allowing more neighboring intervals for potential hits. While the ratio of the allowed left and right deviation is at least 1:2 for a tolerance of one neighboring interval, the ratio bound changes to 2:3 for two, and to 3:4 for three neighbor intervals, and so on. This way, the necessary symmetry can be adjusted. In the asymptotic case of an infinite number of discretization intervals this approach leads to a perfect symmetry of the search range with regard to the query angles.

While the deviation problem is solved this way, another problem has been introduced at the same time. Instead of traversing just one path down the tree, the search procedure must follow the path to more than just one child, if they fit the tolerance criterion. This alters the worst-case time complexity into an exponential function of the pattern length, it is now $\mathcal{O}(|\Sigma|^{|p|}) = \mathcal{O}(|\Sigma|^m)$. Although this is a much worse result, again, the (worst-case) query time does not depend on the size of the database. And, as we will see, the usual query time is not that bad. It was shown by MAASS [Maa04] that, for a suffix *trie* built

```
Algorithm 6: GenSuffixTree::approxSearch(w, curpos)
Impl. Arg.: string& p                           // reference of the pattern
            int pSize                            // the size of the pattern
            vector<int>& srange            // reference of the search ranges
            vector<string>& strings    // access to all strings of the GST
Expl. Arg.: w                               // the current node of the GST
            curpos                  // the current position in the pattern
Result: calls showLeaves() each time the pattern was found
if curpos < pSize then
    nxt ⟵ p[curpos] ;         // pattern character at the current position
    // Compute the bounds of the accepted range
    firstCh ⟵ ((nxt−1− srange[curpos] + alphabetSize) % alphabetSize) +1
    lastCh ⟵ ((nxt−1+ srange[curpos] ) % alphabetSize) +1
    if 2*srange[curpos]+1 ≥ alphabetSize then                    // full range
        firstCh ⟵ 1
        lastCh ⟵ alphabetSize
    notSplit ⟵ (firstCh ≤ lastCh)
    child ⟵ (w→children);                                   // child iterator
    if notSplit then
        while child ≠ ⊥ and firstCh > strings[child→stridx][child→start] do
            child ⟵ (child→next)
    while child ≠ ⊥ do
        if notSplit and lastCh < strings[child→stridx][child→start] then
            return
        if notSplit or isApproxEqual(nxt, strings[child→stridx][child→start],
                                                         srange[curpos]) then
            v ⟵ child
            last ⟵ curpos + v→end − v→start +1
            if last > pSize then
                last ⟵ pSize
            found ⟵ true
            for (int i⟵curpos; i<last; i++) do
                if not isApproxEqual(p[i], strings[v→stridx][v→start+i-curpos],
                                                                  srange[i]) then
                    found ⟵ false
                    break
            if found then
                if last < pSize then
                    approxSearch (v, last)
                else
                    showLeaves (v, curpos, pSize)
        child ⟵ (child→next)
```

from random angles with a uniform and memoryless distribution, the asymptotic running time would be $\mathcal{O}(|S|^{\log_{|\Sigma|}(2t+1)})$, where $|S|$ is the number of sequences, t denotes the search tolerance, and $2t+1$ is the total number of allowed intervals for each angle of the sequence. We do not claim these conditions to be true. At least, this seems to be a rough estimate for the average run time of the procedure, that is confirmed by our experiments.

4.7.7 Searching with Insertions and Deletions

A problem, that sometimes occurs in searches using the previously described tolerant search method, is that some related structures of the database match the pattern only in non-consecutive parts. This means, larger parts of acceptable similarity are interrupted by substructures, that do not match with respect to their shape (angles), or that represent an insertion or a deletion compared to the query structure. In this case, the search process has to be split into two phases: first, the search for the contiguous matching substructures (called *fragments*), and second, the combination of these substructures to the final result structures (hits). A fragment is a continuous stretch of the sequence occurring in the query and in the database (subject to a given tolerance). We consider maximal fragments, that is, approximate occurrences of the query pattern that cannot be extended without violating one of the fragment requirements.

To define the properties of the expected results, the user has to provide a number of parameters, that describe the accepted kind of results:

- a lower bound for the size of the fragments (minimum fragment size),
- the maximum per-position deviation (search tolerance, acceptable interval neighborhood), and
- the admissible average deviation of a fragment (average search tolerance), expressed as a fraction of the maximum per-position deviation.

The minimum fragment size avoids splitting potential matches into very small pieces. The search tolerance serves the same purpose as in the normal tolerant search, it bounds the local deviation of the measure at each single position. The average search tolerance bounds the sum of the deviations. This measure can be given as an absolute value, but usually it is more convenient to give it as a fraction of the position-based tolerance. In this form, the value is independent of the length of the query. Both parameters together allow larger local deviations if the overall fit of the structures is better on average.

At first, we are looking for all substructures in the database (or, more precisely in the GST), that match a suffix of the query structure having at least the minimum length. The attempt to apply the same method as in the case of the longest common (exact) substructure, that is, to traverse the suffixes by following the path along the suffix links of the last matching node, is not applicable (since we apply tolerant matching). After traversing a suffix link, possible matches with slight deviations, that reside in subtrees which are not ancestors of the current node, would be lost. Hence, the algorithm searches each appropriate suffix by a tolerant search from the root node, and keeps track of all fragments having a length of at least the minimum fragment size. Together with the fragments, their current total sum of deviations is stored.

While RÖMERS [Röm04] proposed a method that stores and updates the found fragments after each edge transition (of course, only if they fulfilled the length and deviation requirements), we propose to use a parameter that indicates one of the following three states during the traversal of the tree:

1. The minimum fragment size has not been reached yet.
2. The minimum fragment size has been reached and the edge label of the current path from the root is within the (per-character) tolerance requirements.
3. The minimum fragment size has been reached, but the edge label of the current path from the root violates the tolerance requirements.

Another parameter should provide the current length of the longest path (from the root) fulfilling the average tolerance criterion.

During the traversal of the PAST, we always know from the new parameters about two conditions:

1. whether the minimum fragment size was reached already, and
2. whether the current path from the root is within the per-character tolerance, or not.

If the first of these two conditions is false, we only follow the branches that meet the local tolerance condition. Traversal of all other branches is canceled. If we find only characters within the tolerance, we continue matching and updating the minimum size flag.

If the first condition is true, that is, we found an acceptable fragment earlier, we must continue the traversal anyway to store the occurrences represented by the leaves. But, in case of mismatching characters (out of tolerance bounds), we must remember that further extension of the hit is forbidden. In the case of acceptable characters, we must update the parameter for the longest acceptable string if the average tolerance condition is met. In both cases, we store the found fragment if we are visiting a leaf node.

In contrast to the approach of RÖMERS, this method allows to store each fragment *only once* (at the respective leaf), and it avoids the cumbersome update procedure. The advancement is greatly appreciated, since the largest part of the runtime for searching with insertions and deletions is due to the fragment finding process (see [Röm04]).

The result of the traversal is a set of fragments of different lengths, fulfilling the length and tolerance requirements. For the combination of the found fragments we refer to the work of RÖMERS [Röm04]. The given method is very fast in practice.

Another approach would be to use dynamic programming. This method could also allow the fragments to adopt a different order compared to the query, which is a phenomenon that frequently occurs in the context of genomes and proteomes. With regard to multi-domain proteins for instance, the actual sequence order of the domains is often not essential to the function of the protein. The important requirement is that the fragments of one structure cover the functional parts of the other structure.

4.8 Applications

4.8.1 Zinc Fingers

One of the most abundant motifs of short length is the class of *zinc fingers*. These substructures consist of two antiparallel β-strands (a β-hairpin) followed by a loop and an α-helix. The structure of the classic zinc finger is usually stabilized by a zinc atom, which is bound to two cysteine residues of the hairpin and to two histidine residues of the helix. In absence of the zinc atom, the structure does not fold into the finger form.

The classic zinc finger is a tandem-repeated motif that recognizes and binds to DNA or RNA sequences. Other variants of zinc fingers bind to specific parts of DNA in monomeric form, or in dimeric form to palindromic parts of nucleic acids. The specificity of DNA-binding changes for different lengths and different residues of the parts between the cysteine and histidine residues.

Interestingly enough, it has been shown by Dahiyat and Mayo [DM97, DSM97], that polypeptides can be designed synthetically, that adopt the zinc finger structure without the help of the zinc atom. They used a special algorithm that distinguishes the amino acids according to their fitness for the core, the boundary, and the surface of the folded protein. As a result, they obtained a polypeptide called FSD-1 (Full Sequence Design), that did not contain any cysteine or histidine residue. Searching the amino acid sequence of FSD-1 against protein sequence databases did not yield any significant similarity to a known protein.

After synthesizing the peptide, the structure was determined using NMR spectroscopy. The more or less surprising result was a quite similar structure compared to the template (Zif 268). The root-mean-square deviation of less than 2 Å for the main chain atoms confirmed a good agreement with the target structure.

As a consequence, the PDB entry that holds the structure of FSD-1, is only reported by structure-based searches. Any (amino acid) sequence-based method (e.g., patterns and profiles of the PROSITE database, see [BFM97, SCH+02]) must inherently fail to find this entry. Thus, it is a great opportunity to prove the usefulness of our structure-based approach.

CATH classifies zinc fingers within the first 4 levels as follows (see Section 5.2.2)

C	Class:	3	Alpha Beta
A	Architecture:	30	2-Layer Sandwich
T	Topology:	160	Double Stranded RNA Binding Domain
H	Homologous Superfamily:	60	Classic Zinc Finger

The further levels **S** (Sequence Family), **N** (Non-identical), **I** (Identical), **D** (Domain) are listed in Table D.1.

4.8.2 Searching Zinc Fingers of the *CCHC*-Type

To demonstrate the capabilities of the PAST, we perform a first search for zinc fingers of the *CCHC*-type (because we want to have a survey result set for the first try). This type

Occurrence			Sequence	Tol.	αRMSD	C_α RMSD
1mfs		13-29	VKCFNCGKEGHIAKNCR	0	0.0	0.00
1a1t	A	13-29	VKCFNCGKEGHIAKNCR	2	5.9	0.42
	A	34-50	KGCWKCGKEGHQMKDCT	3	12.3	0.87
1a6b	B	24-40	DQCAYCKEKGHWAKDCP	3	16.5	1.22
1aaf		13-29	IKCFNCGKEGHIAKNCR	2	12.2	0.71
		34-50	RGCWKCGKEGHQMKDCT	2	12.4	0.79
1bj6	A	13-29	VKCFNCGKEGHTARNCR	3	11.8	0.77
	A	34-50	KGCWKCGKEGHQMKDCT	10	28.4	1.29
1cl4	A	51-67	GLCPRCKRGKHWANECK	14	56.0	1.95
1dsq	A	29-45	PVCFSCGKTGHIKRDCK	4	13.0	0.72
1dsv	A	56-72	GLCPRCKKGYHWKSECK	11	48.0	1.45
1esk	A	13-29	VKCFNCGKEGHTARNCR	2	11.9	0.72
	A	34-50	KGCWKCGKEGHQMKDCT	6	31.9	1.12
1f6u	A	13-29	VKCFNCGKEGHIAKNCR	3	14.1	0.93
	A	34-50	KGCWKCGKEGHQMKDCT	1	7.7	0.53
1hvn	E	1-17	VKCFNCGKEGHIARNCR	3	17.5	1.22
1hvo	E	1-17	VKCFNCGKEGHIARNCR	3	14.7	1.11
1mfs		13-29	VKCFNCGKEGHIAKNCR	0	0.0	0.00
		34-50	KGCWKCGKEGHQMKDCT	1	6.8	0.63
1nc8		7-23	IRCWNCGKEGHSARQCR	4	15.0	0.91
1ncp	C	22-38	KGCWKCGKEGHQMKDCT	10	30.9	1.42
	N	1-17	VKCFNCGKEGHTARNCR	6	21.1	1.41
1u6p	A	24-40	DQCAYCKEKGHWAKDCP	2	6.7	0.56
1wwd	A	24-40	DQCAYCKEKGHWAKDCP	1	7.3	0.59
1wwe	A	24-40	DQCAYCKEKGHWAKDCP	1	7.3	0.59
1wwf	A	24-40	DQCAYCKEKGHWAKDCP	1	7.9	0.55
1wwg	A	24-40	DQCAYCKEKGHWAKDCP	1	6.5	0.59
2znf		1-17	VKCFNCGKEGHIARNCR	3	19.5	1.15

Table 4.5: Hits from the search for zinc fingers of the CCHC type.

of zinc finger contains one histidine and three cysteine residues that bind to the zinc atom.

We built the PAST using α-angles with an alphabet size of 36 characters. Each angle interval has size 10°. After the PAST has been built, be performed several searches using different tolerance settings. We tried to find approximate matches of the $CCHC$-type zinc finger, which was taken from PDB entry `1mfs`, residues `13–29`. (In this case, there is only one chain, which has no identifier.)

The queries consumed only a few seconds. For the first tolerances, the elapsed time was even below one second. The results are shown in Table 4.5. The first column contains the PDB entry and the position of the hits. The second column shows the sequence fragment that was found. The four major conserved residues are shaded. In the third column, we show the tolerance setting, where the particular hit was found for the first time. The fourth column contains the root-mean-square deviation for the original sequence of angles of the search fragment and the angle sequence of the hit. In the last column, we show the RMSD of an optimal superimposition of the C_α atoms.

4.8.3 Searching Classic Zinc Fingers

We will now look at a more extensive example, the search for classic zinc fingers of the C_2H_2-type. We perform queries that demonstrate, how the search time, the result set, and other variables depend on the search tolerance and the discretization accuracy (size of the alphabet / interval size). We further compare the result sets to the entries of the PROSITE, SCOP, and CATH databases, as well as to searches that were performed using the SPASM tool.

All tables show the search for a classic zinc finger that we took from PDB file `1a1j`, chain `A`, residues `137` to `157`. Thus, every result set must contain this PDB entry (with deviation zero for all measures).

Tol.	Accepted interval	Time [s]	Nr. of Results	Out of RMSD	Distinct results	Distinct PDB files
0	10°	0.0	3	0	1	1
1	30°	0.0	57	0	31	16
2	50°	0.7	290	0	76	41
3	70°	4.5	749	1	99	57
4	90°	9.1	1098	15	124	75
5	110°	14.9	1431	199	174	95
6	130°	18.1	1589	395	216	111
7	150°	20.5	1692	906	258	126
8	170°	22.4	1766	1919	275	134
9	190°	26.8	1938	3482	333	169
10	210°	32.7	2146	5957	426	202
13	270°	58.2	2818	48251	848	440

Table 4.6: Query comparison: time and number of results for different tolerances using α-angles.

Search Time Comparison for Varying Tolerances

The running times of queries for different tolerances are given in Table 4.6 (using α-angles), Table 4.7 (using ψ/φ-angles) and Table 4.8 (using *OCCO*-angles). The tables show the search tolerance in the left column, followed by the total accepted angle interval (two times the tolerance plus one, multiplied with the size of one interval). The third column shows the overall search time in seconds. The next two columns show the number of hits (within the RMSD bound) and the number of rejected hit candidates (due to exceeding the RMSD bound). Since we built the PAST using all MODEL entries of the PDB files, the result set is somewhat redundant. We remove this redundancy by counting in the sixth column only one model for each distinct position in the PDB file. For the last column, we count each PDB file only ones, hence giving the number of hit PDB files.

The query times for all three angle types do not exceed one minute. While the times for the ψ, φ- and *OCCO*-angles are below one second for the first five rows (tolerance values 0 to 4), the times for the α-based search jumps to 4.5 seconds at tolerance 3. This is due to the fact, that this query already reports roughly 750 hits, whereas this number is reached much later, at tolerance 7, by the other angle types. If we compare the runtimes for equal sizes of the result set, they seem to be very similar, although the tolerances are very different.

In contrast to the α- and *OCCO*-angle queries, the searches that use the ψ/φ-angles show a significant saturation effect at the tolerances 6 to 10, where the number of results for all three measures (all models, model-reduced hits, and distinct files) grows only moderately. Also, the number of (spurious) hits, which are found to exceed the RMSD bound, grows very slowly. Afterwards, this number increases by a large value, which means,

Tol.	Accepted interval	Time [s]	Nr. of Results	Out of RMSD	Distinct results	Distinct files
0	10°	0.0	3	0	1	1
1	30°	0.0	12	0	6	6
2	50°	0.0	62	0	30	16
3	70°	0.2	87	0	43	21
4	90°	0.4	157	0	59	29
5	110°	1.6	380	0	81	43
6	130°	3.9	655	3	94	53
7	150°	5.0	749	16	98	54
8	170°	5.7	803	29	105	57
9	190°	6.3	844	31	106	58
10	210°	7.1	882	35	108	60
13	270°	12.7	1015	1886	140	86

Table 4.7: Query comparison: time and number of results for different tolerances using ψ/φ-angles.

Tol.	Accepted interval	Time [s]	Nr. of Results	Out of RMSD	Distinct results	Distinct PDB files
0	10°	0	3	0	1	1
1	30°	0	5	0	3	3
2	50°	0	32	1	15	13
3	70°	0,1	78	78	49	32
4	90°	0,4	189	131	89	50
5	110°	1,4	373	282	116	69
6	130°	3,1	579	711	152	91
7	150°	5,3	758	1527	175	103
8	170°	6,9	870	2521	199	117
9	190°	8,5	963	4154	215	124
10	210°	10,5	1066	6284	236	139
11	230°	13,4	1206	9877	280	165
12	250°	18,8	1428	19732	361	218
13	270°	31,5	1753	47641	491	313

Table 4.8: Query comparison: time and number of results for different tolerances using $OCCO$-angles.

the tolerance is getting too large for a clear distinction between query-related and other structures.

Query Results Comparison for Varying Tolerances

The coverage of the different result sets (for varying tolerances) with respect to the entries of the SCOP, CATH, and PROSITE databases is shown in the Tables 4.9, 4.15, and 4.16. Please note: Since all three tables show rather similar results, we only present the table for α-angles right here, the other two are deferred to the end of the chapter.

All three tables are divided into two subtables, the upper part showing statistics about different hit positions, the lower part showing statistics about different hit PDB files (usually, zinc fingers occur more than once per protein).

The left column indicates the search tolerance. The second to fourth columns indicate the percentage of found positions (upper subtable) and found files (lower subtable) in the PROSITE, CATH, and SCOP database. The latter one is represented by two columns, the first of which refers to the SCOP family 'Classic zinc finger, C2H2', and the second refers to the whole superfamily 'C2H2 and C2HC zinc fingers', which includes also the C_2HC-type of zinc fingers. The last column shows the coverage according to the union of all database entries. The numbers in parentheses within the subtable captions refer to the number of relevant entries in the respective database.

The tables show a strong performance of the α-angles. They reach a coverage of more than 90% for all categories using a search tolerance of 13 intervals. This measure should be compared to the coverage that is reached by each single database. We emphasize that these fractions of coverage should be interpreted with care, since the databases do not base on identical sets of PDB files. Thus, sometimes the report of a possible hit may depend on whether the database is up-to-date or not.

Tol.	PROSITE hits		CATH hits		SCOP-Fam hits		SCOP-SF hits		All hits	
	(126)	%	(99)	%	(116)	%	(122)	%	(151)	%
0	1	0,8	1	1,0	1	0,9	1	0,8	1	0,7
1	26	20,6	30	30,3	26	22,4	26	21,3	30	19,9
2	55	43,7	53	53,5	47	40,5	48	39,3	63	41,7
3	64	50,8	58	58,6	55	47,4	57	46,7	76	50,3
4	69	54,8	59	59,6	60	51,7	63	51,6	82	54,3
5	76	60,3	63	63,6	65	56,0	68	55,7	89	58,9
6	93	73,8	72	72,7	79	68,1	83	68,0	106	70,2
7	104	82,5	82	82,8	89	76,7	93	76,2	117	77,5
8	112	88,9	86	86,9	97	83,6	101	82,8	125	82,8
9	114	90,5	87	87,9	99	85,3	103	84,4	127	84,1
10	117	92,9	89	89,9	102	87,9	107	87,7	131	86,8
13	123	97,6	93	93,9	107	92,2	112	91,8	137	90,7

Tol.	PROSITE files		CATH files		SCOP-Fam files		SCOP-SF files		All files	
	(48)	%	(31)	%	(43)	%	(49)	%	(63)	%
0	1	2,1	1	3,2	1	2,3	1	2,0	1	1,6
1	14	29,2	15	48,4	14	32,6	14	28,6	15	23,8
2	27	56,3	21	67,7	25	58,1	26	53,1	31	49,2
3	34	70,8	23	74,2	31	72,1	33	67,3	41	65,1
4	36	75,0	23	74,2	33	76,7	36	73,5	44	69,8
5	40	83,3	24	77,4	36	83,7	39	79,6	48	76,2
6	44	91,7	25	80,6	39	90,7	43	87,8	52	82,5
7	45	93,8	25	80,6	40	93,0	44	89,8	53	84,1
8	46	95,8	25	80,6	41	95,3	45	91,8	54	85,7
9	46	95,8	25	80,6	41	95,3	45	91,8	54	85,7
10	47	97,9	26	83,9	42	97,7	47	95,9	56	88,9
13	48	100,0	26	83,9	42	97,7	47	95,9	57	90,5

Table 4.9: Coverage of PROSITE, CATH, and SCOP entries compared for different tolerances using α-angles. SCOP-Fam represents the SCOP family, whereas SCOP-SF refers to the superfamily.

Database	PROSITE hits		CATH hits		SCOP-Fam hits		SCOP-SF hits		All hits	
	(126)	%	(99)	%	(116)	%	(122)	%	(151)	%
PROSITE	126	100,0	85	85,9	106	91,4	108	88,5	126	83,4
CATH	85	67,5	99	100,0	76	65,5	76	62,3	99	65,6
SCOP	106	84,1	76	76,8	116	100,0	116	95,1	116	76,8
SCOP-SF	108	85,7	76	76,8	116	100,0	122	100,0	122	80,8

Database	PROSITE files		CATH files		SCOP-Fam files		SCOP-SF files		All files	
	(48)	%	(31)	%	(43)	%	(49)	%	(63)	%
PROSITE	48	100,0	23	74,2	38	88,4	40	81,6	49	77,8
CATH	23	47,9	31	100,0	24	55,8	24	49,0	32	50,8
SCOP	38	79,2	24	77,4	43	100,0	43	87,8	44	69,8
SCOP-SF	40	83,3	24	77,4	43	100,0	49	100,0	50	79,4

Table 4.10: Cross comparison of PROSITE, CATH, and SCOP.

Cross-comparison of the Databases

To get an impression of how the coverage of the three databases by the PAST searches should be rated, we estimated the same performance measures for each single database. As can be seen from Table 4.10, the respective measures are not significantly different from the query results for the PAST queries.

Comparison to SPASM

For the purpose of performance evaluation, we also compared the search times and the result sets of the PAST queries to the response times and search results of the SPASM tool (see Table 4.11). We used the same set of PDB files for both tools, though we should note, that SPASM only recognizes the first chain of each file. We used the same PDB entry for searching: `1a1j(A):137` – `157`. SPASM offers several options for customizing the search process. For the first three runs, we used all C_α carbons of the zinc finger. Thus, the actual residue type was ignored. The restriction for the maximum per-residue deviation was set to $(2.5, 2.5)$, $(2.0, 2.5)$, and $(3.0, 3.0)$ for the $C_\alpha \,/\, C_\alpha$ and side-chain / side-chain distances, respectively.

The other two queries were performed giving SPASM only the positions of the highly conserved residues ($CCFHH$), where the Phe (residue `144`) was allowed to be substituted by Leu, Ile, Val, Met, Tyr, Trp, or Cys. The first query was constrained to conserve the neighboring residues, as well as the gap sizes. For the second query, these restrictions were removed.

(a) Time and number of results compared to SPASM searches.

Query	Time [s]	Distinct results	Distinct PDB files
SPASM (a)	658,9	64	45
SPASM (b)	640,5	43	29
SPASM (c)	676,7	69	48
SPASM CCFHH	261,2	106	60
SPASM CCHH	260,5	46	31
PAST, α, $\lvert\Sigma\rvert = 36$, Tol. 3	4,5	99	57
PAST, α, $\lvert\Sigma\rvert = 36$, Tol. 8	22,4	275	134

(b) Coverage of PROSITE, CATH, and SCOP entries compared to SPASM searches.

Query	PROSITE hits		CATH hits		SCOP-Fam hits		SCOP-SF hits		All hits	
	(126)	%	(99)	%	(116)	%	(122)	%	(151)	%
SPASM (a)	30	62,5	17	54,8	27	62,8	28	57,1	33	52,4
SPASM (b)	21	43,8	15	48,4	19	44,2	19	38,8	22	34,9
SPASM (c)	30	62,5	18	58,1	28	65,1	30	61,2	35	55,6
SPASM CCFHH	40	83,3	22	71,0	35	81,4	38	77,6	45	71,4
SPASM CCHH	24	50,0	15	48,4	20	46,5	21	42,9	25	39,7
PAST (α,36,$\pm$3)	34	70,8	23	74,2	31	72,1	33	67,3	41	65,1
PAST (α,36,$\pm$8)	46	95,8	25	80,6	41	95,3	45	91,8	54	85,7

Table 4.11: Query comparison between SPASM and PAST.

$\|\Sigma\|$	angle int.	Tol.	Acc. int.	Nr. of nodes	Constr. [min]	Search [s]	Nr. of Results	Out of RMSD	Distinct results
4	90°	1	270°	46.068.590	01:53	0:51.8	2717	41104	748
6	60°	1	180°	47.055.575	02:05	0:25.5	1918	3294	334
12	30°	2	150°	46.398.855	02:30	0:20.9	1714	877	262
18	20°	3	140°	45.596.240	02:59	0:18.9	1618	510	218
24	15°	4	135°	44.846.093	03:30	0:19.1	1627	433	233
36	10°	6	130°	43.594.121	04:30	0:18.4	1589	395	216
60	6°	10	126°	41.808.012	06:29	0:17.6	1549	339	197

Table 4.12: Query comparison: time and number of results for different alphabet sizes using α-angles.

$\|\Sigma\|$	PROSITE hits		CATH hits		SCOP-Fam hits		SCOP-SF hits		All hits	
	(126)	%	(99)	%	(116)	%	(122)	%	(151)	%
4	124	98,4	93	93,9	108	93,1	113	92,6	138	91,4
6	118	93,7	89	89,9	102	87,9	106	86,9	131	86,8
12	106	84,1	84	84,8	91	78,4	95	77,9	119	78,8
18	95	75,4	74	74,7	80	69,0	84	68,9	108	71,5
24	86	68,3	69	69,7	71	61,2	75	61,5	99	65,6
36	93	73,8	72	72,7	79	68,1	83	68,0	106	70,2
60	86	68,3	69	69,7	71	61,2	75	61,5	99	65,6

Table 4.13: Coverage of PROSITE, CATH, and SCOP entries compared for different alphabet sizes using α-angles.

As can be seen, the results of PAST (for the tolerance of ± 8) show a better coverage than any result of SPASM. But, we believe, this is mainly due to the fact, that SPASM does not consider all chains. Again, we emphasize: we do not claim to produce better results than any other tool, but, we observe that PAST produces comparable results in less time.

Search Time and Results Comparison for Varying Alphabet Size

The influence of the discretization accuracy, that is, the size of the alphabet, is shown in Table 4.12 (for α-angles). We observe that, for reasonable alphabet sizes, there are no large differences in the number of results. While the time for the construction of the PAST increases with the decreasing size of the intervals, the search time shows a contrary behavior.

Query Results Comparison for Different Angle Types

One of the most important questions is: What type of angles should be preferred? The answer is a rather strict statement, which becomes clear from looking at the Tables 4.9, 4.15, and 4.16. The best results are obtained in the case, where the virtual torsion angles α are used to encode the structures.

Unconfirmed Hits

The experiments with the PAST produced a lot of zinc finger hits, which are not contained in either of the databases (at least, at that moment). Some of them (having the best RMSD values) are shown in Table 4.14. We believe, that most of them are new to the PDB, and the classification databases do not contain them yet.

Special attention deserve the entries `1fy7`, `1mja`, `1mjb`, and `1mj9`. They do not contain all of the conserved residues and it would be very hard or impossible to find them by sequence-based methods. SCOP annotates for these entries (in family N-acetyl transferase, NAT; protein Histone acetyltransferase ESA1)

> "...contains a rudiment CCHC zinc-finger (res. 191-220)..."

which confirms the hit as a true positive.

The entries, that represent the synthetic zinc fingers (see Section 4.8.1), are also found using PAST together with α-angles. They exhibit even less detectable sequence similarity to the classic zinc fingers, and cannot be found by sequence-based methods. Entry `1psv` is found with tolerance ± 3, the other entries, `1fsd` and `1fsv` are found using tolerance ± 10 and ± 11, respectively. They are listed in the SCOP family 'Zinc finger based beta-beta-alpha motif'.

4.8.4 Other Applications

There is a vast number of other applications, that can be solved straight forward by using the PAST. For example, searching all maximal helix or strand sequences of a particular subtype (q.v. [HT96]) can be used to compute related statistics that are useful in protein structure prediction. These problems include the following questions (among others):

- How frequent is each of the subtypes of helices and strands?
- How many helices are two-sided polar/hydrophobic?
- What amino acids are preferred in π- or 3_{10}-helices?

Of course, the same applies to small motifs like sandwiches, metal- or nucleic acid-binding structures [SGJT04], or larger functional motifs. In general, the determination of the frequency of occurrence for arbitrary small structure fragments will be an important application, since this measure is an essential part of functions rating the significance of hits as BLAST-like E-values or P-values. Further applications include electron density map interpretation, loop closure, and model building [Gre85, RR85, JK97, CD03, KGLK05, JT86].

PDB file	Chain	First	Last	Amino acid sequence	MaxD	aRMSD	AvgD	CaRMSD
1xf7	A	5	25	CKTCQRKFSRSDHLKTHTRTH	22.5	9.6	7.4	0.39
2cot	A	21	41	CDECGKSFSHSSDLSKHRRTH	16.1	8.1	6.2	0.48
1x6e	A	17	37	CVECGKAFSRSSILVQHQRVH	24.2	10.3	8.2	0.56
1x6f	A	28	48	CKHCDSKLQSTAELTSHLNIH	66.9	22.9	13.4	0.64
2cse	W	183	203	CHVCSAVLFSPLDLDAHVASH	17.5	10.6	9.2	0.65
1ej6	C	183	203	CHVCSAVLFSPLDLDAHVASH	17.5	10.6	9.2	0.65
2cot	A	49	69	CDECGKAFIQRSHLIGHHRVH	27.4	11.9	9.3	0.67
1wjp	A	45	65	CPYCSLRFFSPELKQEHESKC	33.7	13.0	11.0	0.68
1x6e	A	45	65	CLECGKAFSQNSGLINHQRIH	38.3	13.6	9.8	0.68
1x6h	A	50	70	CSKCGKTFTRRNTMARHADNC	31.5	10.3	7.7	0.70
1x5w	A	12	32	CSECSYSCSSKAALRIHERIH	49.8	17.1	11.8	0.71
1rik	A	5	25	CPECPKRFMRSDHLTLHILLH	27.7	14.1	12.0	0.74
2ct1	A	18	38	CYICHARFTQSGTMKMHILQK	22.0	11.6	9.5	0.83
1wjp	A	19	39	CRLCNAKLSSLLEQGSHERLC	48.1	19.5	13.3	0.86
2ctd	A	65	85	CHHCGKQLRSLAGMKYHVMAN	23.2	11.1	8.5	0.87
1fy7	A	198	218	DDFTLQYFGSKKQYERYRKKC	37.9	14.1	10.7	0.87
1mja	A	198	218	DDFTLQYFGSKKQYERYRKKC	37.8	14.0	10.7	0.87
1mjb	A	198	218	DDFTLQYFGSKKQYERYRKKC	37.9	14.1	10.7	0.87
1mj9	A	198	218	DDFTLQYFGSKKQYERYRKKC	30.3	11.7	9.3	0.90
1x5w	A	40	60	CNYCSFDTKQPSNLSKHMKKF	49.4	18.7	12.5	0.90
2csh	A	40	60	CGVCGKKFKMKHHLVGHMKIH	25.7	12.8	10.2	0.90
1rim	A	5	25	CPECPKRFMRSDHLSKHITLH	37.2	15.1	11.8	0.96
2csh	A	68	88	CNICAKRFMWRDSFHRHVTSC	27.3	12.8	10.3	0.97
1psv		5	25	ARIKGRTFSNEKELRDFLETF	45.8	20.6	16.0	0.99
1x6h	A	18	38	CSHCDKTFRQKQLLDMHFKRY	27.3	12.8	9.8	1.02
1wir	A	18	38	CLFCDRLFASAEETFSHCKLE	31.5	15.8	13.1	1.02
1u85	A	10	30	CPDCDWSFSRSDHLALHRKRH	26.2	13.0	11.0	1.04
2ct1	A	48	68	CPHCDTVIARKSDLGVHLRKQ	48.6	19.4	14.5	1.09
1zu1	A	97	117	CPVCNMTFSSPVVAESHYIGK	41.1	18.4	14.1	1.35
1zr9	A	45	65	CLACARYFIDSTNLKTHFRSK	32.3	16.6	13.3	1.38

Table 4.14: Sequences not contained in the results of PROSITE, CATH and SCOP.

Tol.	PROSITE hits (126)	%	CATH hits (99)	%	SCOP-Fam hits (116)	%	SCOP-SF hits (122)	%	All hits (151)	%
0	1	0,8	1	1,0	1	0,9	1	0,8	1	0,7
1	6	4,8	6	6,1	6	5,2	6	4,9	6	4,0
2	28	22,2	29	29,3	25	21,6	25	20,5	30	19,9
3	38	30,2	38	38,4	32	27,6	32	26,2	42	27,8
4	48	38,1	45	45,5	40	34,5	42	34,4	54	35,8
5	58	46,0	53	53,5	49	42,2	52	42,6	66	43,7
6	63	50,0	57	57,6	53	45,7	57	46,7	74	49,0
7	65	51,6	60	60,6	53	45,7	58	47,5	78	51,7
8	68	54,0	63	63,6	54	46,6	59	48,4	81	53,6
9	69	54,8	63	63,6	55	47,4	60	49,2	82	54,3
10	69	54,8	63	63,6	55	47,4	60	49,2	82	54,3
13	74	58,7	65	65,7	60	51,7	65	53,3	87	57,6

Tol.	PROSITE files (48)	%	CATH files (31)	%	SCOP-Fam files (43)	%	SCOP-SF files (49)	%	All files (63)	%
0	1	2,1	1	3,2	1	2,3	1	2,0	1	1,6
1	6	12,5	6	19,4	6	14,0	6	12,2	6	9,5
2	15	31,3	16	51,6	13	30,2	13	26,5	16	25,4
3	19	39,6	17	54,8	17	39,5	17	34,7	20	31,7
4	21	43,8	18	58,1	19	44,2	21	42,9	24	38,1
5	28	58,3	22	71,0	25	58,1	28	57,1	33	52,4
6	32	66,7	24	77,4	28	65,1	32	65,3	39	61,9
7	32	66,7	24	77,4	28	65,1	33	67,3	40	63,5
8	32	66,7	24	77,4	28	65,1	33	67,3	40	63,5
9	33	68,8	24	77,4	29	67,4	34	69,4	41	65,1
10	33	68,8	24	77,4	29	67,4	34	69,4	41	65,1
13	37	77,1	25	80,6	33	76,7	38	77,6	45	71,4

Table 4.15: Coverage of PROSITE, CATH, and SCOP entries compared for different tolerances using ψ/φ-angles.

Tol.	PROSITE hits (126)	%	CATH hits (99)	%	SCOP-Fam hits (116)	%	SCOP-SF hits (122)	%	All hits (151)	%
0	1	0,8	1	1,0	1	0,9	1	0,8	1	0,7
1	3	2,4	3	3,0	3	2,6	3	2,5	3	2,0
2	15	11,9	14	14,1	15	12,9	15	12,3	15	9,9
3	31	24,6	28	28,3	27	23,3	27	22,1	34	22,5
4	49	38,9	45	45,5	41	35,3	43	35,2	55	36,4
5	58	46,0	53	53,5	48	41,4	51	41,8	67	44,4
6	63	50,0	55	55,6	51	44,0	55	45,1	73	48,3
7	66	52,4	58	58,6	52	44,8	57	46,7	77	51,0
8	67	53,2	60	60,6	52	44,8	57	46,7	79	52,3
9	67	53,2	60	60,6	52	44,8	57	46,7	79	52,3
10	67	53,2	61	61,6	52	44,8	57	46,7	80	53,0
11	70	55,6	62	62,6	55	47,4	60	49,2	83	55,0
12	76	60,3	63	63,6	60	51,7	65	53,3	89	58,9
13	81	64,3	66	66,7	65	56,0	70	57,4	95	62,9

Tol.	PROSITE files (48)	%	CATH files (31)	%	SCOP-Fam files (43)	%	SCOP-SF files (49)	%	All files (63)	%
0	1	2,1	1	3,2	1	2,3	1	2,0	1	1,6
1	3	6,3	3	9,7	3	7,0	3	6,1	3	4,8
2	13	27,1	12	38,7	13	30,2	13	26,5	13	20,6
3	20	41,7	18	58,1	19	44,2	19	38,8	22	34,9
4	23	47,9	20	64,5	22	51,2	24	49,0	27	42,9
5	27	56,3	22	71,0	25	58,1	28	57,1	33	52,4
6	32	66,7	24	77,4	28	65,1	32	65,3	40	63,5
7	33	68,8	24	77,4	28	65,1	33	67,3	41	65,1
8	33	68,8	24	77,4	28	65,1	33	67,3	41	65,1
9	33	68,8	24	77,4	28	65,1	33	67,3	41	65,1
10	33	68,8	24	77,4	28	65,1	33	67,3	41	65,1
11	35	72,9	24	77,4	30	69,8	35	71,4	43	68,3
12	40	83,3	24	77,4	34	79,1	39	79,6	48	76,2
13	43	89,6	25	80,6	37	86,0	42	85,7	52	82,5

Table 4.16: Coverage of PROSITE, CATH, and SCOP entries compared for different tolerances using *OCCO*-angles.

Chapter 5

Motifs and Classification

The best way to have a good idea is to have a lot of ideas.

Linus Pauling

5.1 Identification of Motifs

5.1.1 Previous Work on Motif Detection

The detection of similarities in sequences originates from the classic work of Needleman and Wunsch. They applied an iterative matrix calculation procedure to the comparison of amino acid sequences. A number of other heuristic algorithms had been proposed. A general framework for identification of repeated patterns in strings, arrays, and trees was described by Karp, Miller, and Rosenberg [KMR72]. Later, metric measures were investigated to describe similarities or distances of sequence pairs. An algorithm to find segments of maximum similarity for two sequences, allowing for insertions and deletions of arbitrary length, was published by Smith and Waterman [SW81].

The detection of frequent structural motifs has been investigated using graph-theoretic abstractions (e.g. clique detection, see [SVPS95, KLW96, KL97, VBK02]), Geometric Hashing [PA98, LNW01] or a combination of both approaches [WKHK03, WKHK04].

The work of Sagot, Viari, Pothier, and Soldano [SVPS95] attempts to find approximate repetitions in a set of protein structures by defining a similarity relation on the mesh partitions of the Ramachandran map. It extends an earlier algorithm by Soldano, Viari, and Champesme, which itself is an extension of an algorithm for finding exact repetitions by Karp, Miller, and Rosenberg [KMR72]. The method uses the concept of maximal cliques of the similarity relation and is one of the first that does not only apply dynamic programming to a pair of structure sequences.

Koch, Lengauer, and Wanke [KLW96, KL97] tried to find maximal common substructures based on the secondary structure elements. They use a more graph-theoretical approach by modifying an algorithm by Bron and Kerbosch, which enumerates all maximal cliques of a graph. The aim is to find maximal common substructures in two or several graphs, where the nodes represent the secondary structure elements. This is done by reducing the problem to the Maximum Clique Problem of the so-called product graph. By restricting the search to cliques that represent connected substructures the search space can be reduced.

Sagot et al. [SVS97] provide algorithms for finding substrings that occur with a certain error in a given number of sequences of the sequence set. In this case, the methods are only applied to the sequence of amino acids and not to the structure in space. The same applies for a paper by Marsan and Sagot [MS00], but here suffix trees are used in combination with the Hamming distance (instead of the Levenshtein/edit distance).

Escalier et al. [EPSV98] present an algorithm that recursively computes common three-dimensional substructures for two or several proteins. But this method can only be applied to small instances with about 30 atoms. Otherwise, a two step approach with a Branch and Bound technique must be applied.

Chew et al. [CHKK99] present an approach that is capable of computing common geometric substructures of only two molecules, but most interestingly the paper also addresses the problem of detecting common domains, i.e. larger substructures where proximity in space does not coincide with continuousness along the polypeptide chain. Another interesting point is the use of an alternative backbone representation that is based on the virtual-bond vectors between consecutive α-carbon atoms.

Geometric Hashing, a method related to the field of computer vision, was used to solve the structural alignment problem. For an application to the alignment of multiple structures and the detection of common motifs, we refer to the work of Nussinov and Wolfson [LNW01]. This method is capable of searching for common domains that are not necessarily contiguous in one chain. Weskamp et al. [WKHK04] proposed a two-step method that tries to combine the advantages of graph-based clique-detection and geometric hashing.

A closely related problem to the search for frequently occurring motifs is the k-Common Substring Problem, where the aim is to find the length (and an occurrence) of a longest substring common to at least k of the K sequences (for all $k \in [2, K]$). Surprisingly, this problem can be solved in time $O(n)$ where n is the sum of the lengths of all sequences (see [Hui92]). Please note that this problem differs from our setting in that occurrences are counted only once per sequence; we want to count every single occurrence (even multiple times in the same string). However, the results of the two problems should not be too different. A restricting point for the algorithm of Hui is: it applies only to exact occurrences.

A detailed survey on different aspects of structure comparison and patterns is given by Eidhammer, Jonassen, and Taylor [EJT99].

There is a vast number of other publications regarding the topic of motif detection, the discussion of which is far behind the scope of this thesis. For completeness, we just provide some references for the most important papers, see [Ric85, ATG92, BG96, KJ97, BJVU98, Kle99, SCH99, BY99, VG01, LP01, BKB02, NBG$^+$02, Vil02, RDR$^+$02, SS03, CP03, SR03, Mul04, IMP$^+$04, MTT04, Tan04, TXL05, HBW$^+$05, Erd05, PPBS05, JSRS05].

5.1.2 Computation of Average Structures

An important factor regarding the determination of frequent motifs is the question whether the database holds a *representative set* of protein *structures*. In the last years, much effort

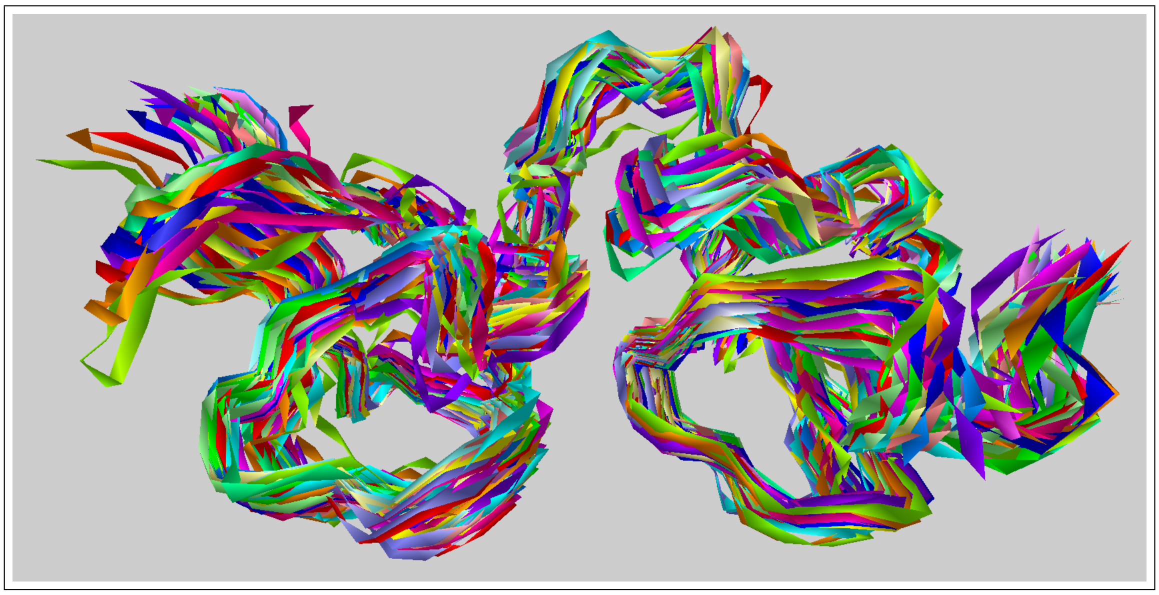
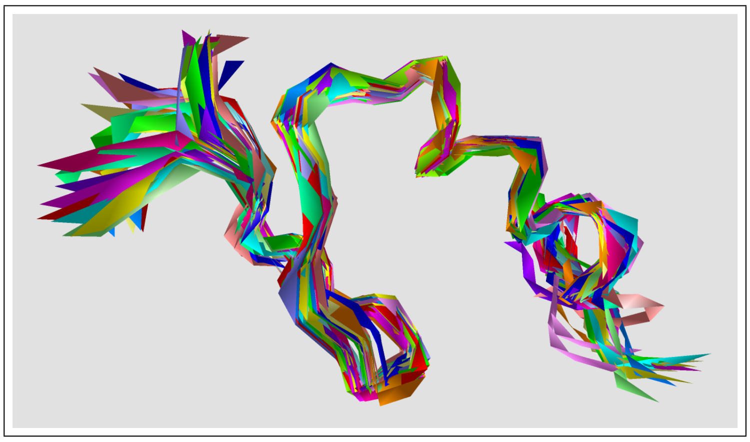


Figure 5.1: 40 different models of antifreeze protein RD3 of *Austrolycichthys brachycephalus* (Antarctic eel pout) [PDB code `1c89`]

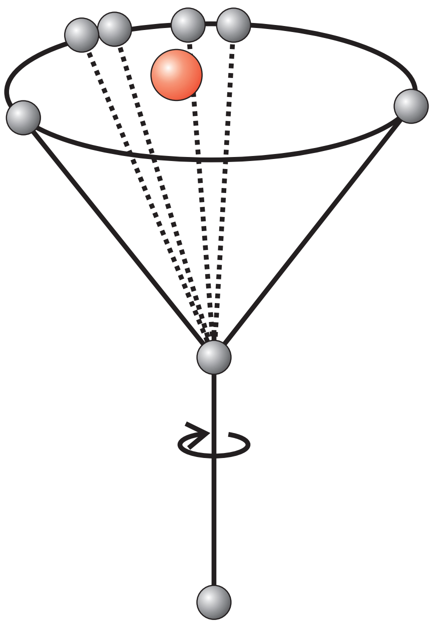
has been put into the creation of non-redundant sets of PDB files, but these projects mostly approached the problem from the viewpoint of *sequence similarity*. It is not clear, whether this provides optimal conditions for motif detection. May be, a structural non-redundant set would produce better results.

Both of these approaches agree in removing the redundancy in the database that is generated by the existence of several measurements or models of the same molecule. These structures have an identical sequence of amino acids, and coincide to a large degree, at least within the structural cores. One solution to the problem could be to simply choose one of the models, but which one should be preferred? Although, in most cases, there seems to be a large structural consensus between all structures, there are some significant deviations at single points for many of the models. Thus, it might be a disadvantage to choose an arbitrary model. Therefore, we propose to use some kind of average structure. As a rather obvious first step, one might try to compute the average coordinates of each atom, that is, the barycenter of the respective atom positions. This approach has been proposed, for instance, by Gerber and Müller, see [GM87]. The method works sufficiently well in the case where not much deviation from the center occurs, but may lead to strange results in the other cases (see Figure 5.2).

An important observation regarding the structures to be superimposed in proteomics is that different models normally differ only with respect to the *torsion angles* of the freely rotatable bonds; they do not disagree much in the bond lengths and bond angles between corresponding atom pairs. As a consequence, the superimposition of different models shows some movement along clearly defined trajectories, at least for the most part of the structures. Hence, the problem is the correct parameterization of the average.

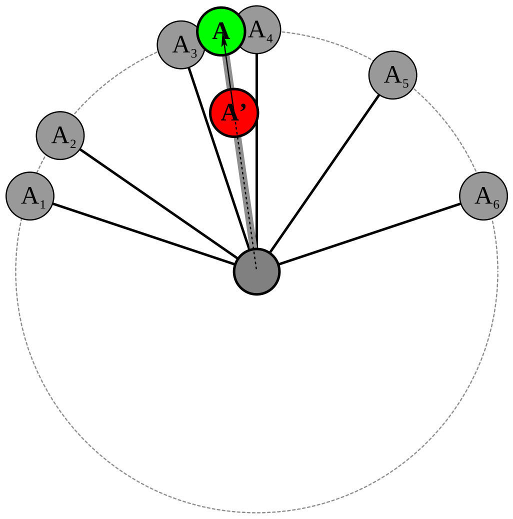


(a) Trajectory example. Note that movement of the chain ends is roughly along a rather specified trace.



(b) Counter-example for computing the consensus structure from the average coordinates.

Figure 5.2: Computing average structures from average coordinates can lead to strange results.



In this figure, the view is along the axis of rotation. A_1,...,A_6 mark the atom positions within the different models, A marks their barycenter, which is apparently no good choice for some kind of average structure.

Figure 5.3: Computing average structures from average angles.

Computing the center of gravity (at average coordinates) may result in an unnatural consensus structure where the resulting point does not agree well with a virtual median of the structures. In extreme cases the point would not lie on the trace of the respective atoms at all, for instance, if the scatter plot is kind of arched or circular, see Figure 5.2. A better approach of computing a consensus structure separates the nearly constant radius of rotation from the comparatively variable torsion angle.

The question immediately arising is: *How can we compute an average angle?* For only two angles φ_1 and φ_2, the answer is simple. Decide, which of the two intervals $[\varphi_1, \varphi_2]$ and $[\varphi_2, \varphi_1]$ is the smaller one, compute the arithmetic mean of the angle values $\bar{\varphi} = (\varphi_1 + \varphi_2)/2$ and add (or subtract, resp.) 180° or π if the mean was in the larger interval (the operation depends on the global angle interval being either from 0° to 360° or from −180° to 180°). The problem gets much more intricate if three or more angles are involved.

It becomes obvious that the arithmetic mean angle is not a promising approach. Also, we notice that there are degenerate instances, where no unique solution can be found. Examples are two angles which are separated by two equal intervals of 180°. For more than three angles, this case occurs, for instance, if the intervals between 'neighboring' angles have all the same size. While this (artificial) problem cannot be solved by any method computing a consensus structure, it would be advantageous to be informed, if a situation close to one of these anomalies is given.

At the end, the solution we propose to the problem is very simple. In principle, the center of gravity was not a completely wrong approach, but it must be applied to the right measure. Instead of computing average atom coordinates, we use it to compute the barycenter of the angles' representations on the unit circle. What we get, is a point somewhere within the circle, that represents the average representation of the angles. This point is also associated with an angle, which can be interpreted in the same way as the input angles (by extending the line from the circle center to the circle). While input angles are always represented by a point on the unit circle (that is, with radius 1), the average angle is associated with a radius between zero and one. The size of this radius is now a measure for the coherence of the input set. This is a very intuitive concept, since a set of very similar angles will produce a barycenter that is close to the circle, and has thus radius 1. An (artificial) set of ideally symmetrical angles will give a barycenter that coincides with the center of the circle, that is radius 0. In this case, no prevalent direction of the angles can be returned, a fact that agrees with the comments made above.

5.1.3 Extraction of Frequent Substructures

The identification of frequently occurring substructures requires an exact definition of the objects to be found. If we define motifs to be substrings of specified length that occur at least in a given number of sequences, then we should apply the approach of Hui [Hui92]. Please note, that many structural patterns are repeated very often within the same chain. Since we want to count different occurrences in the same sequence separately, we could also count the number of leaves in the subtree of each node. Traversing the tree and reporting all nodes having a path label (from the root) of sufficient length would give the most frequent *exactly* repeated strings.

As has been discussed in Sections 4.7.4, the exact matching of structure sequences may lead to quite unsatisfactory results. Therefore, we stick to the approach of tolerant searching. This time, we have to (virtually) apply an all-against-all comparison of all suffixes in the database. This seems to be a hard job, but we can save a lot of time using the PAST.

At first, we have to decide about the alphabet size of the index to be used. We should use a moderate value, since extreme values for the number of intervals could affect run time and results significantly. For the intended comparison process, we assume each node of the generalized suffix tree to be augmented by two counters. After constructing the PAST for a certain alphabet size, the first counter is used to store the number of leaves in the subtree for each node, that is the number of occurrences of the respective path label. This can be done recursively, using a depth-first search (DFS), incrementing the current counter after returning from a descent to a child node. The second counter has to be initialized to zero. It will be used to hold the number of similar structures, that is, the number of angle substrings matching the path label of the node within a given tolerance.

We give now a schematic description of the motif finding process. To compute the counter values of the second kind (approximate match counters), we traverse the suffix tree from left to right (see Algorithm 7). If the length of the path label represented by the current node, is below the desired motif length, we recursively descend to the children (if the node itself is not a leaf). If we visit a node of sufficient depth (a pattern), that is, its path label has length at least the motif length, we use that (entire) path label to search for approximate matches which do not come before the string of the pattern in a lexicographical ordering. This means, we are searching for characters that are equal to or greater than the query characters according to lexicographical sorting. This task is performed by Algorithm 8. We call it with the number of leaves and a reference of the pattern node's own similarity counter as arguments. We must take care whether the current path label is yet equal, or greater than the path label of the calling source node. In the first case, we may traverse the tree by all possible branches, whereas in the second case, we have to restrict the traversal to patterns greater than or equal to the pattern string (with respect to lexicographical ordering).

If we find a matching node of sufficient depth, we increment its similarity counter by the number of nodes of the original calling (pattern) node. The other way round, we increment the similarity counter of the calling node by the number of children of the currently discovered node (via the traceback reference parameter). After returning to the first procedure, we check whether the number of similar structures is greater than a given threshold and display the result if necessary. Please note, that similarity counters cannot be propagated from right to left (due to the traversal of nodes that are never before the pattern in a lexicographical ordering). Thus, after returning to Algorithm 7) from a call to Algorithm 8, we always have the final count of approximate hits stored in the similarity counter.

Consequently, the method returns all motifs fulfilling the requirements of minimum length and minimum number of approximate occurrences.

```
Algorithm 7: GenSuffixTree::motifTraverse(node, prefLen)

Impl. Arg.: motifLength                        // the desired motif length
            motifMinCount            // the minimum number of occurrences
            motifTolerance                           // the search ranges
            greaterThanMotif                         // GST position flag
Expl. Arg.: node                          // the current node of the GST
            prefLen                          // the current prefix length
Result: Traverses the GST and calls updateSimCounters()

v ⟵ node
curPrefLen ⟵ prefLen + edgeLength(node)
if curPrefLen > motifLength or
  (curPrefLen = motifLength and node is no leaf) then
    greaterThanMotif ⟵ false
    updateSimCounters (rootNode, 0, &(strings[v→stridx][v→start - prefLen]),
    v→count, v→simcount)
    if v→simcount ≥ motifMinCount then
        reporter→reportHit(v→stridx, v→start - prefLen, motifLength)
else
    for child ⟵ (v→children); child ≠ NULL; child ⟵ (child→next) do
        motifTraverse (child, curPrefLen)
```

Algorithm 8: GenSuffixTree::updateSimCounters(node, prefLen, p, toAdd, srcCnt)

```
Expl. Arg.: node                          // the current node of the GST
            prefLen                       // the current prefix length
            p                             // pointer to the current pattern
            toAdd                         // the number to be added to all counters
            srcCnt                        // reference to the counter of the calling node
Result: Traverses the GST for lexicographical greater strings
$v \longleftarrow node$
curPrefLen = prefLen + edgeLength($node$)
if $curPrefLen > motifLength$ or
  ($curPrefLen = motifLength$ and $node$ is no leaf) then
    v→simcount += toAdd
    srcCnt+= v→count
else
    c2 ⟵p[curPrefLen]
    for $child \longleftarrow v \rightarrow children$; $child$; $child \longleftarrow child \rightarrow next$ do
        c1 ⟵first character of child
        d ⟵$c1 - c2$
        if ($greaterThanMotif$ or $d \geq 0$) and
          ($d \leq motifTolerance$ or $alphabetSize\text{-}d \leq motifTolerance$ or
          $alphabetSize+d \leq motifTolerance$) and ($c1 \neq SENTINEL$) then
            greaterChangedHere ⟵false
            if (not $greaterThanMotif$) and $d>0$ then
                greaterThanMotif ⟵true
                greaterChangedHere ⟵true
            last ⟵ curPrefLen + child → end − child → start
            if $last \geq motifLength$ then
                last ⟵motifLength-1
            match ⟵true
            for $i \longleftarrow curPrefLen + 1x$; $i \leq last$; $i \longleftarrow i + 1$ do
                c1 ⟵ character $i$ of child
                c2 ⟵p[i]
                d ⟵$c1 - c2$
                if ($d \leq motifTolerance$ or $alphabetSize\text{-}d \leq motifTolerance$ or
                  $alphabetSize+d \leq motifTolerance$) and $c1 \neq SENTINEL$ then
                    match ⟵false
                    break
            if $match$ then
                updateSimCounters($child, curPrefLen, p, toAdd, srcCnt$)
            if $greaterChangedHere$ then
                greaterThanMotif ⟵false
```

5.2 Protein Structure Classification

It is a nearby question to ask whether naturally occurring proteins can be grouped in terms of similar structure. On the one hand, the aim is to derive either possible functions for new proteins having determined structures, or, by contrast, to deduce a probable structure for proteins of known function (by looking at the functions and structures of proteins within the same class). On the other hand classifying proteins into groups of similar (sub)structures may allow conclusions about phylogenetic relationships of structures and species, which is sometimes impossible to derive by sequence-based methods. However, for this last point, we emphasize that structural coincidence is not always due to phylogenetic relationship (that is, by inherited genetic information). Sometimes, the structural properties of unrelated proteins were gradually reduced over time, which is due to the selective pressure of evolution for similar functional requirements. This process is called *convergent evolution*. (It is in contrast to *divergent evolution*, where genes or proteins of a common ancestor are gradually changed into different sequences.)

This kind of structure classification is currently done –at least in large part– manually. The most popular examples of structure classification databases are the hierarchies of SCOP [MBHC95, RB03] and CATH [OMJ$^+$97, OPB$^+$99, OPT03]. Now, it would be a great advancement, to get this job done automatically. This would improve not only the expense of human work, it would also lower the influence of the prejudice due to human interaction.

A further reason to group similar structures is to deduce the main principles that govern the protein folding process, which is essential for structure prediction from sequence, the most-wanted aim of structural genomics.

5.2.1 The SCOP Database

SCOP (Structural Classification of Proteins) [MBHC95, BCHM96, RB03] is a database that aims at providing information on the structural and evolutionary relationships for proteins of known structure. The classification has been created manually by visual inspection, assisted by tools for automatic structure comparison.

Within the hierarchy, the units of categorization are the domains, since they are the typical parts of function and protein evolution

SCOP classifies proteins according to the following hierarchy:

1. **Class**

   There are four major classes, that classify the protein structure according to the main organization of secondary structure elements: all α, all β, α/β, and $\alpha + \beta$. The meaning of the two first classes should be clear, the difference between the two latter classes is: α/β-classified structures consist of a single sheet, where α-helices connect the ends of the strands. Two major forms are seen: in the first variant, the β-sheet is wrapped to form a barrel-like structure that is surrounded by α-helices. In the other variant, the more planar sheet has the helices on the side positions. The members of the α+β-class have the α- and β-parts more separated. The strands are there usually joined by hairpins, not by the helices (thus leading to antiparallel sheets). $\alpha + \beta$-structures may have a (small) cluster of helices, which is tightly packed against the sheet. Additionally to the four major classes, there are some special ones containing multidomain proteins (several domains which are never found separately), membrane proteins, small proteins (stabilized by disulfide bridges or metal ligands rather than hydrophobic cores), coiled coil structures, peptides, and designed proteins.

2. **Fold**

   Proteins are defined as having the same fold, if there is a general structural similarity detected, that is, if they have similar secondary structures arranged in the same way. Proteins of the same fold may or may not have a common evolutionary origin. Often they have peripheral secondary structure elements that differ largely in size and shape.

3. **Superfamily**

   Proteins having only a weak sequence similarity, but significant functional similarities that suggest a common evolutionary origin, are grouped in superfamilies. The common parts may be small if they comprise the important active site(s).

4. **Family**

   SCOP families represent groups of proteins that exhibit a clear evolutionary relationship, that is, they have a high pairwise sequence identity (30% or more), or they exhibit clear similarities in structure and function.

5. **Protein (Domain)** and **Species**

   The unit of function in a particular protein, the actual domain of a certain species, is at the lowest level of the hierarchy.

5.2.2 The CATH Database

CATH [OMJ+97, OPB+99, OPT03] is a hierarchical classification too. The name is an acronym of the four major levels of organization: Class, Architecture, Topology, and Homology. The most general classifier (Class) groups entries according to similar secondary structure composition. The architecture level characterizes the shape, which is made through the orientation and arrangement of the different secondary structure elements. The topology describes the sequential connectivity of the parts. Entries having sufficiently similar structure and function are considered to be evolutionary related, and they are aggregated in the same homology class. If the sequence identity is 35% or above, the structures are put into the same sequence family.

(a) SCOP classification

Class
Fold
Superfamily
Family
Protein Domain
Species
PDB Entry Domain

(b) CATH classification

C	Class
A	Architecture
T	Topology
H	Homologous Superfamily
S	Sequence Family
N	Non-identical
I	Identical
D	Domain

Table 5.1: Classification scheme of SCOP and CATH.

5.2.3 Remarks

Besides SCOP and CATH, there are also other tools and databases that feature the classification of protein structure, such as DALI/FSSP [HS96, HS97], ProtClass [AT05], SARF2 [AF96], and others [Smi04]. They all use more or less different measures and methods, but most of them share one essential property: they try to classify the existing structures according to a *hierarchy*, that is, a tree-like structure. This may or may not be an appropriate approach. It remains unclear so far, whether a hierarchical decomposition does contain all relevant relationships of the vast number of families.

5.3 Containedness and Similarity

A straight forward solution to the problem of classifying structures into groups of high pairwise similarity would be to perform a simple all-against-all structure comparison of the whole database (e.g. using the algorithm from [BYG99]). This would assign a (scalar) measure to each pair, indicating whether these two structures are very similar on the whole or have some common parts (which includes the case that one structure is completely contained within the other). Since we do not want to use sequence similarity measures for known reasons, the similarity of two structures could be measured for example by the RMSD of the (backbone) atom positions. But, there is one hitch: the computation of an RMSD value requires a proper (that is, the best possible or, at least, a near-optimal) assignment of corresponding atoms within the comparison pair. Thus, it remains unclear how this could be done efficiently, since even the number of pairs is quadratic in the (large) number of structures.

Now that we learned that an all-against-all comparison takes too much time, we follow the idea that rating the similarity of structures should be confined to structures that are not completely different. In order to reduce the number of necessary pairwise comparisons, we find, this could be greatly supported by the Polypeptide Angles Suffix Tree. The procedure should be as follows. At first, we build the PAST of all structures as described in Chapter 4. Then we iterate over all structures and search them within the database using the PAST. For all (approximate) matches, we compute the RMSD value and keep the value within the *containedness-similarity matrix*. At the end, we apply a spectral clustering algorithm to the matrix that was originally applied by Ernst to the clustering of gene expression profiles [Ern03]. The result is a partition of the set of structures that should be a good classification scheme of the database. For a sequence-based approach, see [PCCS03].

The algorithm can be further improved, if we apply the same principle of lexicographical traversal as in the algorithm for computing frequent substructures (see Algorithms 7 and 8). Again, we traverse the whole PAST according to a lexicographical ordering, and call a second procedure at each time, when we reached a node of sufficient depth. This depth has to be given as an input to the algorithm. It measures whether the algorithm performs a global or a more local (sub-)structure comparison.

The second function gets a pointer to the calling node and starts (as in the motif detection algorithm) at the root a tolerant search for the path label of the 'calling node', considering only nodes that are greater with respect to the lexicographical ordering. For each leaf having sufficient path length (that is, representing a suffix of sufficient size), the algorithm computes the RMSD of an optimal superimposition with all substructures represented by a leaf in the subtree of the calling node. If this value is below a certain threshold, it is stored in a (sparse) similarity matrix for all sequence pairs. If there has been a successful comparison before, the minimum of both values is kept.

The result is a (sparse) distance matrix of all sequence pairs (with respect to the given substructure length), which can be clustered using, for instance, the algorithm by Ernst.

A more general measure of structural similarity must take into account local as well as global similarity. Thus, the described algorithm has to be performed several times using different settings. The resulting similarity measures (sparse matrices) should be combined by a function that suites the need of the actual application (to emphasize more local or more global similarity).

5.4 Results

First experiments for the identification of frequent substructures indicate running times between several hours and several days, depending on the input parameters (discretization accuracy, required length of the common substructure, and search tolerance). Automatic classification clearly consumes more time, since the root-mean-square deviations for the hits must be computed. To accelerate the RMSD calculation, methods can be used, that do not directly compute the superpositioning rotation. Some of the methods are faster, if only the RMSD value is required. Mostly, some time can be saved if the last square root operation for computing the RMSD is skipped. Instead the squared RMSD is compared to the corresponding threshold.

Although first experiments for these methods led to promising and comprehensible results, we forbear from presenting the large lists in this work. Anyway, these results have to be tested and interpreted by experts of molecular biology, not by computer scientists.

Chapter 6

Conclusion

That's one giant leap for [a] man, one small step for mankind.

The Author

In this thesis, we emphasized the need for fast methods capable of searching huge structure databases like the PDB. We pointed out the drawbacks of currently used methods which sacrifice accuracy for speed.

We proposed a new approach for indexing structure databases using translation- and rotation-invariant measures that are stored in a generalized suffix tree after discretization. We compared different measures according to their suitability for structure searching. Our experiments showed the applicability of the method, by comparing the result sets of the experiments to established tools and databases like PROSITE, SCOP, CATH, and SPASM. As we have seen, the virtual bonds dihedral angle α performs better than other measures. Using this measure, our method achieved comparable results to the other tools, while being much faster—often by an order of magnitude.

In the last part of the thesis, we showed how to apply the new method to other problems of structural biology, such as the identification of frequent substructures (motifs) and the automatic classification of the entries of structural databases. We provided a convenient method for computing average structures from several models of the same molecule, which might be of more general interest to the community.

At the end, this was an essentially interdisciplinary work, that reached from the basic principles of evolution, based on biochemical reactions, to using efficient algorithms and advanced mathematics such as the quaternion concept.

What is left, is to say that all the methods described in this thesis have been implemented in the Protein Structure (ProSt) Project. This includes a WWW service, that allows everyone interested to search for structures in the PDB:

`http://past.in.tum.de/`

Appendix A

List of PAST Query Results

PDB file	Chain	First	Last	Amino acid sequence	MaxD	RmsD	AvgD	C_αRmsD
1a1f	A	137	157	CRICMRNFSRSDHLTTHIRTH	5.5	2.5	2.1	0.16
1a1f	A	165	185	CDICGRKFARSDERKRHTKIH	17.5	7.3	6.0	0.50
1a1g	A	109	129	VESCDRRFSDSSNLTRHIRIH	56.7	25.4	17.7	1.97
1a1g	A	137	157	CRICMRNFSRSDHLTTHIRTH	4.1	2.3	1.9	0.12
1a1g	A	165	185	CDICGRKFARSDERKRHTKIH	13.9	7.3	6.2	0.55
1a1h	A	109	129	VESCDRRFSQSGSLTRHIRIH	74.3	24.9	16.4	2.18
1a1h	A	137	157	CRICMRNFSRSDHLTTHIRTH	3.8	2.0	1.7	0.12
1a1h	A	165	185	CDICGRKFARSDERKRHTKIH	11.9	6.6	5.7	0.54
1a1i	A	109	129	VESCDRRFSRSADLTRHIRIH	74.2	24.0	14.4	1.92
1a1i	A	137	157	CRICMRNFSRSDHLTTHIRTH	6.4	2.8	2.2	0.13
1a1i	A	165	185	CDICGRKFARSDERKRHTKIH	12.0	6.9	6.1	0.54
1a1j	A	109	129	VESCDRRFSRSADLTRHIRIH	75.7	25.0	15.6	1.91
1a1j	A	137	157	CRICMRNFSRSDHLTTHIRTH	0.0	0.0	0.0	0.00
1a1j	A	165	185	CDICGRKFARSDERKRHTKIH	16.5	7.4	6.3	0.59
1a1k	A	109	129	VESCDRRFSRSADLTRHIRIH	65.7	22.6	13.2	1.83
1a1k	A	137	157	CRICMRNFSRSDHLTTHIRTH	6.2	2.9	2.3	0.15
1a1k	A	165	185	CDICGRKFARSDERKRHTKIH	12.4	6.2	5.4	0.55
1a1l	A	109	129	VESCDRRFSRSDELTRHIRIH	67.9	23.3	14.5	1.89
1a1l	A	137	157	CRICMRNFSRSDHLTTHIRTH	6.2	2.5	2.0	0.13
1a1l	A	165	185	CDICGRKFARSDERKRHTKIH	11.9	7.3	6.4	0.56
1a9x	A	714	741	VRAAMEIVYDEADLRRYFQTA	54.2	27.2	21.9	1.68
1a9x	C	2714	2741	VRAAMEIVYDEADLRRYFQTA	51.7	25.6	20.7	1.65
1a9x	E	4714	4741	VRAAMEIVYDEADLRRYFQTA	49.1	24.7	19.7	1.61
1a9x	G	6714	6741	VRAAMEIVYDEADLRRYFQTA	47.1	24.4	19.8	1.58
1aay	A	109	129	VESCDRRFSRSDELTRHIRIH	52.9	19.9	13.2	1.97
1aay	A	137	157	CRICMRNFSRSDHLTTHIRTH	7.5	3.1	2.4	0.17
1aay	A	165	185	CDICGRKFARSDERKRHTKIH	12.4	6.5	5.5	0.47

PDB file	Chain	First	Last	Amino acid sequence	MaxD	RmsD	AvgD	C_αRmsD
1ard		106	126	CEVCTRAFARQEHLKRHYRSH	16.9	8.5	6.9	0.57
1are		106	126	CEVCTRAFARQEALKRHYRSH	17.7	8.8	7.2	0.49
1arf		106	126	CEVCTRAFARQEYLKRHYRSH	21.3	10.3	7.7	0.50
1b70	B	234	254	LFAAGMRPINNVVDVTNYVML	72.0	30.7	22.3	1.83
1b7y	B	234	254	LFAAGMRPINNVVDVTNYVML	77.1	32.0	23.3	1.86
1bbo		4	24	CEECGIRXKKPSMLKKHIRTH	26.4	13.2	10.0	1.01
1bbo		32	52	CTYCNFSFKTKGNLTKHMKSK	41.1	20.8	16.3	1.19
1bfi		68	88	FAEPYNLYSSLKELVLHYQHT	54.0	23.2	18.4	1.90
1bfj		68	88	FAEPYNLYSSLKELVLHYQHT	42.5	21.4	15.9	2.02
1bhi		11	31	APGCGQRFTNEDHLAVHKHKH	65.2	25.6	16.5	1.95
1by8	A	15	35	KKTHRKQYNNKVDEISRRLIW	78.2	28.4	22.7	2.32
1c30	A	714	741	VRPAMEIVYDEADLRRYFQTA	52.1	25.9	20.7	1.81
1c30	C	714	741	VRPAMEIVYDEADLRRYFQTA	51.0	25.0	20.4	1.76
1c30	E	714	741	VRPAMEIVYDEADLRRYFQTA	60.2	27.2	21.6	1.75
1c30	G	714	741	VRPAMEIVYDEADLRRYFQTA	52.8	24.9	19.4	1.73
1c3o	A	714	741	VRPAMEIVYDEADLRRYFQTA	50.7	24.9	19.4	1.80
1c3o	C	714	741	VRPAMEIVYDEADLRRYFQTA	48.2	24.2	19.2	1.81
1c3o	E	714	741	VRPAMEIVYDEADLRRYFQTA	53.4	26.0	20.7	1.78
1c3o	G	714	741	VRPAMEIVYDEADLRRYFQTA	51.1	24.8	19.7	1.74
1ce8	A	714	741	VRPAMEIVYDEADLRRYFQTA	43.6	21.5	16.6	1.63
1ce8	C	714	741	VRPAMEIVYDEADLRRYFQTA	51.5	23.7	18.6	1.54
1ce8	E	714	741	VRPAMEIVYDEADLRRYFQTA	55.4	25.2	19.9	1.57
1ce8	G	714	741	VRPAMEIVYDEADLRRYFQTA	55.0	25.6	20.3	1.55
1cs0	A	714	741	VRPAMEIVYDEADLRRYFQTA	49.7	25.1	19.9	1.78
1cs0	C	714	741	VRPAMEIVYDEADLRRYFQTA	49.4	24.3	19.7	1.77
1cs0	E	714	741	VRPAMEIVYDEADLRRYFQTA	53.6	24.8	19.7	1.75
1cs0	G	714	741	VRPAMEIVYDEADLRRYFQTA	52.4	24.4	18.9	1.74
1cwy	A	418	438	LADWGITFREEEEVPWALMHL	63.4	23.4	17.7	2.37
1ecw	A	87	107	CIHAEEKVKHTEEAKQIVQRH	63.0	23.9	15.9	1.70
1ed1	A	87	107	CIHAEEKVKHTEEAKQIVQRH	67.8	25.5	16.6	1.70
1eiy	B	234	254	LFAAGMRPINNVVDVTNYVML	62.5	30.1	23.2	1.82
1ej6	C	183	203	CHVCSAVLFSPLDLDAHVASH	17.5	10.6	9.2	0.65
1esw	A	418	438	LADWGITFREEEEVPWALMHL	65.7	24.0	18.2	2.39
1f2i	G	1109	1129	VESCDRRFSRSDELTRHIRIH	73.2	24.1	14.2	1.84
1f2i	G	1137	1157	CRICMRNFSRSDHLTTHIRTH	11.6	5.4	4.4	0.46
1f2i	H	2109	2129	VESCDRRFSRSDELTRHIRIH	72.5	24.3	14.3	1.90
1f2i	H	2137	2157	CRICMRNFSRSDHLTTHIRTH	8.0	4.1	3.3	0.26
1f2i	I	3109	3129	VESCDRRFSRSDELTRHIRIH	70.4	23.5	13.0	1.89
1f2i	I	3137	3157	CRICMRNFSRSDHLTTHIRTH	10.6	5.6	4.5	0.34
1f2i	J	4109	4129	VESCDRRFSRSDELTRHIRIH	62.9	21.9	11.8	1.90
1f2i	J	4137	4157	CRICMRNFSRSDHLTTHIRTH	7.8	4.3	3.7	0.28
1f2i	K	5109	5129	VESCDRRFSRSDELTRHIRIH	71.5	24.1	13.4	1.91
1f2i	K	5137	5157	CRICMRNFSRSDHLTTHIRTH	17.8	10.9	9.2	0.48
1f2i	L	6137	6157	CRICMRNFSRSDHLTTHIRTH	10.7	5.0	3.9	0.34

PDB file	Chain	First	Last	Amino acid sequence	MaxD	RmsD	AvgD	C_αRmsD
1fp8	A	418	438	LADWGITFREEEEVPWALMHL	54.9	22.0	17.4	2.40
1fp9	A	418	438	LADWGITFREEEEVPWALMHL	66.2	24.3	18.4	2.43
1fv5	A	11	31	CLPCGIAFSSPSTLEAHQAYY	57.6	18.3	13.7	1.40
1fy7	A	198	218	DDFTLQYFGSKKQYERYRKKC	37.9	14.1	10.7	0.87
1g2d	C	109	129	VESCDRRFSQKTNLDTHIRIH	63.6	17.8	10.7	1.96
1g2d	C	137	157	CRICMRNFSQHTGLNQHIRTH	23.8	8.7	6.4	0.44
1g2d	C	165	185	CDICGRKFATLHTRDRHTKIH	22.3	7.9	6.2	0.49
1g2d	F	209	229	VESCDRRFSQKTNLDTHIRIH	53.6	15.1	9.8	1.88
1g2d	F	237	257	CRICMRNFSQHTGLNQHIRTH	15.9	7.4	6.3	0.45
1g2d	F	265	285	CDICGRKFATLHTRDRHTKIH	20.3	7.1	5.4	0.45
1g2f	C	109	129	VESCDRRFSQKTNLDTHIRIH	63.0	17.5	11.0	1.95
1g2f	C	137	157	CRICMRNFSQQASLNAHIRTH	18.4	6.4	4.6	0.31
1g2f	C	165	185	CDICGRKFATLHTRTRHTKIH	17.2	6.9	5.7	0.47
1g2f	F	209	229	VESCDRRFSQKTNLDTHIRIH	48.0	15.9	10.3	1.94
1g2f	F	237	257	CRICMRNFSQQASLNAHIRTH	15.8	5.3	3.9	0.31
1g2f	F	265	285	CDICGRKFATLHTRTRHTKIH	19.1	7.0	5.6	0.42
1ghs	A	195	215	DQNNGLTYTSLFDAMVDAVYA	59.2	15.9	9.2	1.72
1ghs	B	195	215	DQNNGLTYTSLFDAMVDAVYA	54.8	14.9	9.0	1.72
1hek	A	84	104	AVKXGLQINNVVDGKASFQLL	63.1	27.0	21.3	1.98
1hek	B	84	104	AVKXGLQINNVVDGKASFQLL	54.9	23.6	18.1	2.09
1hiw	A	87	107	CVHQRIDVKDTKEALDKIEEE	60.4	23.8	16.7	2.01
1hiw	B	87	107	CVHQRIDVKDTKEALDKIEEE	57.0	23.7	17.7	2.15
1hiw	C	87	107	CVHQRIDVKDTKEALDKIEEE	61.6	23.3	16.7	1.92
1hiw	Q	87	107	CVHQRIDVKDTKEALDKIEEE	59.2	22.7	15.5	1.99
1hiw	R	87	107	CVHQRIDVKDTKEALDKIEEE	60.2	23.5	16.7	2.01
1hiw	S	87	107	CVHQRIDVKDTKEALDKIEEE	57.1	23.4	16.6	1.98
1jdb	K	713	740	VRPAMEIVYDEADLRRYFQTA	54.5	26.2	21.3	1.57
1jk1	A	109	129	VESCDRRFSRSAELTRHIRIH	68.4	22.3	13.8	1.96
1jk1	A	137	157	CRICMRNFSRSDHLTTHIRTH	3.2	1.7	1.4	0.14
1jk1	A	165	185	CDICGRKFARSDERKRHTKIH	14.1	7.3	6.4	0.55
1jk2	A	109	129	VESCDRRFSRSAELTRHIRIH	55.6	25.7	17.4	1.88
1jk2	A	137	157	CRICMRNFSRSDHLTTHIRTH	5.1	2.3	1.9	0.13
1jk2	A	165	185	CDICGRKFARSDERKRHTKIH	16.0	7.4	6.0	0.53
1jkz	A	7	27	ADTYRGVCFTNASCDDHCKNK	81.6	35.5	27.4	2.46
1jn7	A	11	31	CSTCDISFNYVKTYLAHKQFY	41.6	15.3	11.9	1.17
1kan	A	178	198	GLHHRICYTTSASVLTEAVKQ	64.9	24.6	17.6	1.61
1kan	B	178	198	GLHHRICYTTSASVLTEAVKQ	45.8	20.9	15.8	1.60
1kee	A	714	741	VRPAMEIVYDEADLRRYFQTA	46.9	23.9	19.1	1.81
1kee	C	714	741	VRPAMEIVYDEADLRRYFQTA	48.0	24.2	19.4	1.78
1kee	E	714	741	VRPAMEIVYDEADLRRYFQTA	49.3	24.0	19.3	1.79
1kee	G	714	741	VRPAMEIVYDEADLRRYFQTA	50.4	24.4	19.5	1.78
1klr	A	5	25	CQYCEFRSADSSNLKTHIKTK	43.4	18.8	14.4	0.95
1kls	A	5	25	CQYCELRSADSSNLKTHIKTK	37.0	16.3	13.0	0.89
1kny	A	178	198	GLHHRICYTTSASVLTEAVKQ	57.7	24.8	18.1	1.67
1kny	B	178	198	GLHHRICYTTSASVLTEAVKQ	59.2	26.1	18.7	1.60

PDB file	Chain	First	Last	Amino acid sequence	MaxD	RmsD	AvgD	C_αRmsD
1l6n	A	87	107	CVHQRIDVKDTKEALDKIEEE	41.7	22.8	18.2	1.73
1lhp	A	155	175	ELLTGRKIHSQEEALEVMDML	66.7	24.6	16.5	2.21
1lhp	B	155	175	ELLTGRKIHSQEEALEVMDML	64.8	23.9	16.2	2.17
1lhr	A	155	175	ELLTGRKIHSQEEALEVMDML	67.7	25.4	17.9	2.21
1lhr	B	155	175	ELLTGRKIHSQEEALEVMDML	69.0	23.8	16.3	2.17
1llm	C	106	126	CRICMRNFSRSDHLTTHIRTH	6.2	3.3	2.7	0.19
1llm	C	134	154	CDICGRKFARSDERKRHRDIQ	42.5	13.1	9.1	0.97
1llm	D	206	226	CRICMRNFSRSDHLTTHIRTH	6.9	2.9	2.4	0.13
1llm	D	234	254	CDICGRKFARSDERKRHRDIQ	38.8	12.5	8.7	0.96
1m36	A	10	30	CEFCLKYMKSRTILQQHMKKC	21.3	8.9	6.5	0.63
1mey	C	7	27	CPECGKSFSQSSNLQKHQRTH	12.8	6.3	4.9	0.38
1mey	C	35	55	CPECGKSFSQSSDLQKHQRTH	12.7	5.2	4.1	0.39
1mey	C	63	83	CPECGKSFSRSDHLSRHQRTH	14.8	7.9	6.9	0.52
1mey	F	7	27	CPECGKSFSQSSNLQKHQRTH	19.2	8.5	5.9	0.47
1mey	F	35	55	CPECGKSFSQSSDLQKHQRTH	12.4	6.4	5.3	0.32
1mey	F	63	83	CPECGKSFSRSDHLSRHQRTH	29.5	10.7	8.5	0.45
1mey	G	63	83	CPECGKSFSRSDHLSRHQRTH	20.5	9.4	7.4	0.46
1mj9	A	198	218	DDFTLQYFGSKKQYERYRKKC	30.3	11.7	9.3	0.90
1mja	A	198	218	DDFTLQYFGSKKQYERYRKKC	37.8	14.0	10.7	0.87
1mjb	A	198	218	DDFTLQYFGSKKQYERYRKKC	37.9	14.1	10.7	0.87
1mla		230	250	NNVDVKCETNGDAIRDALVRQ	55.3	19.3	13.0	2.25
1njq	A	8	28	CSFCKREFRSAQALGGHMNVH	43.4	14.8	10.0	1.00
1nm2	A	230	250	SNKDGRAVASGTEVLDRLVGQ	39.3	19.0	15.8	2.00
1p47	A	109	129	VESCDRRFSRSDELTRHIRIH	83.1	27.3	16.3	2.03
1p47	A	137	157	CRICMRNFSRSDHLTTHIRTH	9.8	3.8	3.0	0.19
1p47	A	165	185	CDICGRKFARSDERKRHTKIH	22.9	8.9	6.6	0.51
1p47	B	109	129	VESCDRRFSRSDELTRHIRIH	60.7	23.1	15.2	1.94
1p47	B	137	157	CRICMRNFSRSDHLTTHIRTH	16.1	7.3	6.1	0.34
1p47	B	165	185	CDICGRKFARSDERKRHTKIH	21.3	9.6	7.5	0.56
1p7a	A	14	34	CPDCDRSFSRSDHLALHRKRH	24.5	12.2	9.9	0.79
1paa		134	154	CGLCNRAFTRRDLLIRHAQKI	25.8	11.5	8.5	1.00
1pci	A	26	46	MLNHNKFYENVDEKLYRFEIF	56.0	18.1	13.2	2.25
1pci	B	26	46	MLNHNKFYENVDEKLYRFEIF	56.0	18.1	13.2	2.25
1pci	C	26	46	MLNHNKFYENVDEKLYRFEIF	56.0	18.1	13.2	2.25
1psv		5	25	ARIKGRTFSNEKELRDFLETF	45.8	20.6	16.0	0.99
1qd1	A	286	306	CEKENLFLLQDEHRIRLVVNR	38.5	18.3	14.6	1.74
1qd1	B	2286	2306	CEKENLFLLQDEHRIRLVVNR	47.7	20.5	15.4	1.79
1rft	A	155	175	ELLTGRKIHSQEEALEVMDML	69.4	23.1	15.8	2.15
1rfu	A	155	175	ELLTGRKIHSQEEALEVMDML	68.8	25.9	18.7	2.16
1rfu	B	155	175	ELLTGRKIHSQEEALEVMDML	69.8	26.0	19.0	2.18
1rfu	C	155	175	ELLTGRKIHSQEEALEVMDML	71.0	26.1	18.8	2.18
1rfu	D	155	175	ELLTGRKIHSQEEALEVMDML	70.5	26.4	19.0	2.17
1rfu	E	155	175	ELLTGRKIHSQEEALEVMDML	68.5	26.4	19.2	2.18
1rfu	F	155	175	ELLTGRKIHSQEEALEVMDML	69.8	25.7	18.4	2.17
1rfu	G	155	175	ELLTGRKIHSQEEALEVMDML	70.1	26.4	19.2	2.18
1rfu	H	155	175	ELLTGRKIHSQEEALEVMDML	69.7	26.1	18.9	2.18

PDB file	Chain	First	Last	Amino acid sequence	MaxD	RmsD	AvgD	C_αRmsD
1rfv	A	155	175	ELLTGRKIHSQEEALEVMDML	67.0	24.1	16.0	2.19
1rfv	B	155	175	ELLTGRKIHSQEEALEVMDML	68.2	23.7	15.4	2.17
1rik	A	5	25	CPECPKRFMRSDHLTLHILLH	27.7	14.1	12.0	0.74
1rim	A	5	25	CPECPKRFMRSDHLSKHITLH	37.2	15.1	11.8	0.96
1sp1		5	25	CPECPKRFMRSDHLSKHIKTH	26.7	13.1	11.2	0.83
1sp2		7	27	WSYCGKRFTRSDELQRHKRTH	72.2	27.9	18.5	2.16
1srk	A	10	30	CRICLSAFTTKANCARHLKVH	23.2	10.5	8.1	0.61
1t36	A	714	741	VRPAMEIVYDEADLRRYFQTA	47.3	25.0	20.0	1.74
1t36	C	714	741	VRPAMEIVYDEADLRRYFQTA	41.4	22.7	18.5	1.73
1t36	E	714	741	VRPAMEIVYDEADLRRYFQTA	55.7	25.2	20.1	1.72
1t36	G	714	741	VRPAMEIVYDEADLRRYFQTA	48.7	23.6	18.7	1.71
1td2	A	151	171	EILCEHAVNNVEEAVLAAREL	71.4	29.7	20.1	2.01
1td2	B	151	171	EILCEHAVNNVEEAVLAAREL	67.8	27.3	18.2	2.03
1tf3	A	17	37	FADCGAAYNKNWKLQAHLSKH	58.3	23.6	16.6	1.87
1tf3	A	47	67	EEGCEKGFTSLHHLTRHSLTH	70.8	23.8	14.5	2.05
1tf3	A	77	97	SDGCDLRFTTKANMKKHFNRF	83.2	31.9	21.7	2.09
1tf6	A	17	37	FADCGAAYNKNWKLQAHLCKH	79.0	25.4	14.8	2.04
1tf6	A	47	67	EEGCEKGFTSLHHLTRHSLTH	87.7	28.9	18.2	2.20
1tf6	A	77	97	SDGCDLRFTTKANMKKHFNRF	67.1	27.6	22.2	1.63
1tf6	A	109	129	FENCGKAFKKHNQLKVHQFSH	56.5	19.6	13.3	1.85
1tf6	A	139	159	HEGCDKRFSLPSRLKRHEKVH	67.9	25.9	17.3	2.08
1tf6	D	17	37	FADCGAAYNKNWKLQAHLCKH	79.7	25.4	14.9	2.02
1tf6	D	47	67	EEGCEKGFTSLHHLTRHSLTH	87.6	29.2	18.4	2.21
1tf6	D	77	97	SDGCDLRFTTKANMKKHFNRF	64.4	26.6	21.5	1.62
1tf6	D	109	129	FENCGKAFKKHNQLKVHQFSH	59.7	20.0	13.2	1.87
1tf6	D	139	159	HEGCDKRFSLPSRLKRHEKVH	67.9	26.3	17.0	2.08
1tt9	A	286	306	CDKEKLFVLEEEHRIRLVVNR	48.0	22.2	17.6	1.73
1tt9	B	286	306	CDKEKLFVLEEEHRIRLVVNR	48.1	22.2	17.6	1.73
1tt9	C	286	306	CDKEKLFVLEEEHRIRLVVNR	48.1	22.2	17.6	1.73
1tt9	D	286	306	CDKEKLFVLEEEHRIRLVVNR	47.9	22.2	17.6	1.73
1u85	A	10	30	CPDCDWSFSRSDHLALHRKRH	26.2	13.0	11.0	1.04
1u86	A	12	32	WPDCDRSFSRSDHLALHRKRH	46.5	19.4	15.3	1.98
1ubd	C	327	347	CAECGKAFVESSKLKRHQLVH	21.1	10.6	8.0	0.62
1ubd	C	357	377	FEGCGKRFSLDFNLRTHVRIH	77.5	27.6	16.7	2.02
1un6	B	109	129	FENCGKAFKKHNQLKVHQFSH	80.0	29.7	17.7	2.04
1uph	A	87	107	CVHQRIDVKDTKEALDKIEEE	69.2	22.9	16.2	1.92
1va1	A	541	561	IQGCGKVYGKTSHLRAHLRWH	34.3	16.3	13.1	1.84
1va2	A	571	591	WSYCGKRFTRSDELQRHKRTH	50.2	19.6	14.6	1.52
1va3	A	599	619	CPECPKRFMRSDHLSKHIKTH	36.2	13.2	10.4	0.69
1vi9	A	151	171	EILCEHAVNNVEEAVLAAREL	86.7	33.1	21.9	2.04
1vi9	B	151	171	EILCEHAVNNVEEAVLAAREL	74.5	30.5	20.3	2.04
1vi9	C	151	171	EILCEHAVNNVEEAVLAAREL	72.7	29.1	19.3	2.04
1vi9	D	151	171	EILCEHAVNNVEEAVLAAREL	72.7	29.9	19.7	2.08
1vl2	C	235	255	NLKDGTEKTDPLELFEYLNEV	62.8	22.5	15.1	2.48
1wir	A	18	38	CLFCDRLFASAEETFSHCKLE	31.5	15.8	13.1	1.02

PDB file	Chain	First	Last	Amino acid sequence	MaxD	RmsD	AvgD	C_αRmsD
1wjp	A	19	39	CRLCNAKLSSLLEQGSHERLC	48.1	19.5	13.3	0.86
1wjp	A	45	65	CPYCSLRFFSPELKQEHESKC	33.7	13.0	11.0	0.68
1wjp	A	72	92	CLECMRTFKSSFSIWRHQVEV	30.8	17.8	15.2	1.07
1wty	A	46	66	LELQGLEARSPRAAIRGAFQV	49.9	21.6	15.1	1.97
1wty	B	46	66	LELQGLEARSPRAAIRGAFQV	59.8	24.6	18.3	2.02
1wty	C	46	66	LELQGLEARSPRAAIRGAFQV	72.7	25.0	17.9	2.13
1wty	D	46	66	LELQGLEARSPRAAIRGAFQV	70.8	23.5	16.3	2.05
1wwp	A	46	66	REKEGLEGASPKGVIRLAREV	36.3	16.7	13.3	1.86
1wwp	B	46	66	REKEGLEGASPKGVIRLAREV	33.2	16.7	13.9	1.82
1x3c	A	32	52	HQGCFAAFTIQQNLILHYQAV	81.2	29.1	19.4	2.13
1x5w	A	12	32	CSECSYSCSSKAALRIHERIH	49.8	17.1	11.8	0.71
1x5w	A	40	60	CNYCSFDTKQPSNLSKHMKKF	49.4	18.7	12.5	0.90
1x6e	A	17	37	CVECGKAFSRSSILVQHQRVH	24.2	10.3	8.2	0.56
1x6e	A	45	65	CLECGKAFSQNSGLINHQRIH	38.3	13.6	9.8	0.68
1x6f	A	28	48	CKHCDSKLQSTAELTSHLNIH	66.9	22.9	13.4	0.64
1x6h	A	18	38	CSHCDKTFRQKQLLDMHFKRY	27.3	12.8	9.8	1.02
1x6h	A	50	70	CSKCGKTFTRRNTMARHADNC	31.5	10.3	7.7	0.70
1xf7	A	5	25	CKTCQRKFSRSDHLKTHTRTH	22.5	9.6	7.4	0.39
1xhx	D	180	200	QFKQGLDRMTAGSDSLKGFKD	81.1	29.4	19.8	2.48
1xkg	A	12	32	KKAFNKSYATFEDEEAARKNF	52.7	21.3	15.3	1.97
1y0j	B	11	31	CLPCGIAFSSPSTLEAHQAYY	55.9	19.8	15.8	1.30
1y6e	B	57	77	YIDGDVKLTQSMAIIRYIADK	82.5	33.6	24.7	2.27
1ygj	A	155	175	ELLTGRKIHSQEEALEVMDML	67.0	26.0	18.9	2.19
1ygk	A	155	175	ELLTGRKIHSQEEALEVMDML	63.7	24.9	18.0	2.15
1yhj	A	155	175	ELLTGRKIHSQEEALEVMDML	63.6	24.1	17.4	2.13
1yui	A	36	56	CPICYAVIRQSRNLRRHLELR	25.0	12.5	9.4	0.96
1yuj	A	36	56	CPICYAVIRQSRNLRRHLELR	24.0	12.4	9.7	0.88
1zaa	C	9	29	VESCDRRFSRSDELTRHIRIH	71.3	23.6	15.4	1.97
1zaa	C	37	57	CRICMRNFSRSDHLTTHIRTH	8.5	4.1	3.5	0.19
1zaa	C	65	85	CDICGRKFARSDERKRHTKIH	13.9	7.5	6.2	0.49
1zfd		46	66	HPGCDKAFVRNHDLIRHKKSH	61.4	24.8	17.4	2.19
1zgm	A	64	84	DRERNFAISQXPAIAIYLGER	34.8	17.8	13.3	2.07
1zgm	B	64	84	DRERNFAISQXPAIAIYLGER	36.3	18.4	14.0	2.10
1znf		3	23	CGLCERSFVEKSALSRHQRVH	47.5	16.7	10.4	0.65
1zr9	A	45	65	CLACARYFIDSTNLKTHFRSK	32.3	16.6	13.3	1.38
1zu1	A	36	56	CKVCSAVLISESQKLAHYQSR	41.1	19.3	15.0	1.76
1zu1	A	97	117	CPVCNMTFSSPVVAESHYIGK	41.1	18.4	14.1	1.35
2ab3	A	7	27	FENCGRSFNDRRKLNRHKKIH	38.2	12.7	9.6	2.19
2ab7	A	7	27	FENCGRSFNDRRKLNRHKKIH	68.8	25.9	17.7	2.23
2abq	A	186	206	SELVSKPIASIEDAIPHVQRL	72.6	26.0	18.4	2.46
2adr		106	126	CEVCTRAFARQEHLKRHYRSH	21.4	12.0	9.6	0.59
2adr		134	154	CGLCNRAFTRRDLLIRHAQKI	30.7	13.5	10.3	0.89
2ajp	A	155	175	ELLSGRKIHSQEEALRVMDML	79.7	25.2	16.4	2.26
2ajp	B	155	175	ELLSGRKIHSQEEALRVMDML	77.0	26.9	18.9	2.33

PDB file	Chain	First	Last	Amino acid sequence	MaxD	RmsD	AvgD	C_αRmsD
2cot	A	21	41	CDECGKSFSHSSDLSKHRRTH	16.1	8.1	6.2	0.48
2cot	A	49	69	CDECGKAFIQRSHLIGHHRVH	27.4	11.9	9.3	0.67
2cs0	A	76	96	IPGEKVAHTSLDALVTFHQQK	83.3	35.1	23.7	1.93
2cse	W	183	203	CHVCSAVLFSPLDLDAHVASH	17.5	10.6	9.2	0.65
2csh	A	40	60	CGVCGKKFKMKHHLVGHMKIH	25.7	12.8	10.2	0.90
2csh	A	68	88	CNICAKRFMWRDSFHRHVTSC	27.3	12.8	10.3	0.97
2ct1	A	18	38	CYICHARFTQSGTMKMHILQK	22.0	11.6	9.5	0.83
2ct1	A	48	68	CPHCDTVIARKSDLGVHLRKQ	48.6	19.4	14.5	1.09
2ctd	A	65	85	CHHCGKQLRSLAGMKYHVMAN	23.2	11.1	8.5	0.87
2drp	A	113	133	CKVCSRVYTHISNFCRHYVTS	22.4	10.2	8.6	0.87
2drp	A	143	163	CPFCFKEFTRKDNMTAHVKII	35.5	13.1	10.4	0.91
2drp	D	113	133	CKVCSRVYTHISNFCRHYVTS	22.3	9.5	7.5	0.84
2drp	D	143	163	CPFCFKEFTRKDNMTAHVKII	36.8	11.2	7.4	0.99
2gli	A	174	194	FEGCRKSYSRLENLKTHLRSH	86.9	27.6	17.1	2.10
2gli	A	235	255	LPGCTKRYTDPSSLRKHVKTV	64.5	26.7	19.6	2.17
2hmx		88	108	CVHQRIDVKDTKEALDKIEEE	49.5	23.1	17.1	1.72
3znf		5	25	CSYCNFSFKTKGNLTKHMKSK	47.0	22.2	16.9	1.35
4znf		5	25	CSYCNFSFKTKGNLTKHMKSK	52.4	24.5	17.9	1.28
5znf		5	25	CQYCEYRSADSSNLKTHIKTK	37.1	19.1	15.0	0.96
7pck	A	15	35	KKTHRKQYNNKVDEISRRLIW	50.9	16.9	12.0	2.14
7pck	B	15	35	KKTHRKQYNNKVDEISRRLIW	51.3	16.8	11.9	2.12
7pck	C	15	35	KKTHRKQYNNKVDEISRRLIW	50.8	17.1	12.6	2.16
7pck	D	15	35	KKTHRKQYNNKVDEISRRLIW	50.6	18.5	13.8	2.11
7znf		5	25	CQYCEKRFADSSNLKTHIKTK	41.1	19.4	14.2	0.95

Appendix B

List of PROSITE Entries

1A1F	A	105- 134
		135- 162
		163- 186
1A1G	A	105- 134
		135- 162
		163- 186
1A1H	A	105- 134
		135- 162
		163- 187
1A1I	A	105- 134
		135- 162
		163- 187
1A1J	A	105- 134
		135- 162
		163- 186
1A1K	A	105- 134
		135- 162
		163- 187
1A1L	A	105- 134
		135- 162
		163- 187
1AAY	A	105- 134
		135- 162
		163- 187
1ARD		104- 130
1ARE		104- 130
1ARF		104- 130
1BBO		2- 29
		30- 57
1BHI		7- 31
1F2I	G	1105-1134
		1135-1158
	H	2105-2134
		2135-2158
	I	3105-3134
		3135-3158
	J	4105-4134
		4135-4158
	K	5105-5134
		5135-5158
	L	6105-6134
		6135-6158
1FV5	A	9- 36
1G2D	C	105- 134
		135- 162
		163- 189
1G2D	F	205-234
		235-262
		263-288
1G2F	C	105-134
		135-162
		163-189
	F	205-234
		235-262
		263-288
1JK1	A	105-134
		135-162
		163-187
1JK2	A	105-134
		135-162
		163-187
1LLM	C	104-131
		132-155
	D	204-231
		232-255
1LU6	A	322-351
		352-381
		382-409
		413-442
1NCS		32- 60
1NJQ	A	6- 33
1P47	A	105-134
		135-162
		163-188
	B	105-134
		135-162
		163-186
1P7A	A	12- 34
1PAA		132-159
1SP1		3- 29
1SP2		3- 31
1SRK	A	8- 35
1TF3	A	13- 42
		43- 72
		73-101
1TF6	A	13- 42
		43- 72
		73- 98
		105-134
		135-161
1TF6	D	13- 42
		43- 72
		73- 98
		105-134
		135-161
1UBD	C	296-320
		325-352
		353-382
		383-408
1UN6	B	105-134
		135-161
	C	105-134
		135-161
	D	135-161
1VA1	A	537-565
1VA2	A	567-595
1VA3	A	597-623
1WJP	A	70- 98
1Y0J	B	9- 36
1ZAA	C	5- 34
		35- 62
		63- 87
1ZFD		42- 70
1ZNF		1- 25
1ZNM		2- 28
2ADR		104-131
		132-160
2DRP	A	111-139
		141-165
	D	111-139
		141-166
2GLI	A	104-136
		137-169
		170-199
		200-230
		231-257
3ZNF		3- 30
4ZNF		3- 30
5ZNF		3- 30
7ZNF		3- 30

Table B.1: Zinc fingers that match the PROSITE domain profile (matrix) PS50157.

Appendix C

List of SCOP Entries

PDB file	Chain	First	Last
1a1f	A	103	131
1a1f	A	132	159
1a1f	A	160	186
1a1g	A	103	131
1a1g	A	132	159
1a1g	A	160	186
1a1h	A	103	131
1a1h	A	132	159
1a1h	A	160	187
1a1i	A	103	131
1a1i	A	132	159
1a1i	A	160	187
1a1j	A	103	131
1a1j	A	132	159
1a1j	A	160	186
1a1k	A	103	131
1a1k	A	132	159
1a1k	A	160	187
1a1l	A	103	131
1a1l	A	132	159
1a1l	A	160	187
1aay	A	103	131
1aay	A	132	159
1aay	A	160	187
1ard			
1are			
1arf			
1bbo		1	28
1bbo		29	57
1bhi			
1f2i	G	1093	1131
1f2i	G	1132	1158
1f2i	H	2093	2131
1f2i	H	2132	2158
1f2i	I	3093	3131
1f2i	I	3132	3158
1f2i	J	4093	4131
1f2i	J	4132	4158
1f2i	K	5096	5131
1f2i	K	5132	5158
1f2i	L	6093	6131
1f2i	L	6132	6158
1jk1	A	103	131
1jk1	A	132	159
1jk1	A	160	187
1jk2	A	103	131
1jk2	A	132	159
1jk2	A	160	187

Table C.1: SCOP entries of the superfamily 'C2H2 and C2HC zinc fingers'.

PDB file	Chain	First	Last
1klr	A		
1kls	A		
1llm	C	101	128
1llm	C	129	156
1llm	D	201	228
1llm	D	229	256
1ncs			
1p47	A	102	131
1p47	A	132	159
1p47	A	160	188
1p47	B	103	131
1p47	B	132	159
1p47	B	160	186
1p7a	A		
1paa			
1rmd		87	116
1sp1			
1sp2			
1srk	A		
1tf3	A	1	40
1tf3	A	41	70
1tf3	A	71	101
1tf6	A	10	40
1tf6	A	41	70
1tf6	A	71	100
1tf6	A	101	131
1tf6	A	132	160
1tf6	A	161	188
1tf6	D	7	40
1tf6	D	41	70
1tf6	D	71	100
1tf6	D	101	131
1tf6	D	132	160
1tf6	D	161	188
1ubd	C	295	322
1ubd	C	323	350
1ubd	C	351	380
1ubd	C	381	408
1un6	B	104	131
1un6	B	132	160
1un6	B	161	190
1un6	C	104	131
1un6	C	132	160
1un6	C	161	190
1un6	D	133	160
1un6	D	161	190
1yui	A		
1yuj	A		
1zaa	C	3	31
1zaa	C	32	59
1zaa	C	60	87
1zfd			
1znf			
2adr		102	130
2adr		131	161
2drp	A	103	139
2drp	A	140	165
2drp	D	102	139
2drp	D	140	166
2gli	A	103	134
2gli	A	135	167
2gli	A	168	197
2gli	A	198	228
2gli	A	229	257
3znf			
4znf			
5znf			
7znf			
1njq	A		
1m36	A		
1fv5	A		
1fu9	A		
1uw2	A		
1jn7	A		

Table C.2: SCOP entries of the superfamily 'C2H2 and C2HC zinc fingers' (continued).

Appendix D

List of CATH Entries

S	N	I	D
1	1	1	1llmC1, 1llmD1, 1a1hA2, 1a1iA2, 1aayA2, 1jk2A2, 1a1gA2, 1a1kA2, 1jk1A2, 1a1jA2, 1a1fA2, 1zaaC2, 1p47A2, 1p47B2, 1a1lA2
		2	1f2iG2, 1f2iH2, 1f2iI2, 1f2iJ2, 1f2iK2, 1f2iL2
	2	1	1a1hA3, 1a1iA3, 1aayA3, 1a1gA3, 1a1kA3, 1jk1A3, 1a1jA3, 1a1fA3, 1zaaC3, 1p47A3, 1p47B3, 1a1lA3
	3	1	1g2fC2, 1g2fF2
	4	1	1g2fC3, 1g2fF3
		2	1g2dC3, 1g2dF3
	5	1	1g2dC2, 1g2dF2
2	1	1	1a1hA1
	2	1	1a1iA1, 1a1kA1, 1a1jA1
		2	1jk2A1, 1jk1A1
	3	1	1aayA1, 1zaaC1, 1p47A1, 1p47B1, 1a1lA1, 1f2iG1, 1f2iH1, 1f2iI1, 1f2iJ1, 1f2iK1, 1f2iL1
	4	1	1a1gA1, 1a1fA1
	5	1	1g2fC1, 1g2fF1, 1g2dC1, 1g2dF1
	6	1	1ubdC4
3	1	1	1rmd02
4	1	1	1meyC1, 1meyF1
		2	1meyC2, 1meyF2
	2	1	1meyC3, 1meyF3, 1meyG0
	3	1	1ubdC2
	4	1	1bbo01
5	1	1	1ubdC1
	2	1	2gliA5
6	1	1	1ubdC3
	2	1	2gliA3
	3	1	1tf3A2
7	1	1	2gliA1
8	1	1	2gliA2
9	1	1	2gliA4
10	1	1	2drpA1, 2drpD1
11	1	1	2drpA2, 2drpD2
	2	1	2adr02
12	1	1	1b69A0, 1tn9A0
		2	1bb800, 2bb800
13	1	1	1bbo02
14	1	1	1ncs00
15	1	1	1tf3A1
16	1	1	1tf3A3
17	1	1	1yuiA0, 1yujA0
18	1	1	2adr01

Table D.1: Classification of Zinc Fingers in the CATH hierarchy.

Appendix E

List of Abbreviations / Acronyms

AFP Aligned Fragment Pair

ASCII American Standard Code for Information Interchange

BLAST Basic Local Alignment Search Tool

DFA Deterministic Finite Automaton

DFS Depth-First Search

DNA DeoxyriboNucleic Acid

GST Generalized Suffix Tree

HGP Human Genome Project

NMR Nuclear Magnetic Resonance

PAST Polypeptide Angles Suffix Tree

PDB Protein Data Bank

RCSB Research Collaboratory for Structural Bioinformatics

RMSD Root-Mean-Square Deviation or Root-Mean-Square Distance

RNA RiboNucleic Acid

SHSP Structural High Scoring Pairs

SSE Secondary Structure Element

SVD Singular Value Decomposition

Bibliography

[3DP02] *Proceedings of the 1st International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT'02)*, June 2002.

[AD03] Russ B. Altman and Jonathan M. Dugan. Defining bioinformatics and structural bioinformatics. In Bourne and Weissig [BW03], pages 3–14.

[AF96] Nickolai N. Alexandrov and Daniel Fischer. Analysis of topological and non-topological structural similarities in the PDB: New examples with old structures. *Proteins: Structure, Function, and Bioinformatics*, 25(3):354–365, July 1996.

[AFT03] Zeyar Aung, Wei Fu, and Kian-Lee Tan. An efficient index-based protein structure database searching method. In DASFAA 2003 [DAS03], pages 311–318.

[AGM$^+$90] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, October 1990.

[AHB87] K. S. Arun, Thomas S. Huang, and Steven D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5):698–700, September 1987.

[AMM44] Oswald T. Avery, Colin M. MacLeod, and Maclyn McCarty. Studies on the chemical nature of the substance inducing transformation of pneumococcal types: Induction of transformation by a desoxyribonucleic acid fraction isolated from pneumococcus type III. *The Journal of Experimental Medicine*, 79(2):137–158, February 1944.

[AN95] Arne Andersson and Stefan Nilsson. Efficient implementation of suffix trees. *Software: Practice and Experience*, 25(2):129–141, February 1995.

[AOI97] Tatsuya Akutsu, Kentaro Onizuka, and Masato Ishikawa. Rapid protein fragment search using hash functions based on the Fourier transform. *Computer Applications in the Biosciences*, 13(4):357–364, August 1997.

[ASPM97] Enrique E. Abola, Joel L. Sussman, Jaime Prilusky, and Nancy O. Manning. Protein data bank archives of three-dimensional macromolecular structures. In Carter and Sweet [CS97], pages 546–556.

[AT04] Zeyar Aung and Kian-Lee Tan. Rapid 3D protein structure database searching using information retrieval techniques. *Bioinformatics*, 20(7):1045–1052, May 2004.

[AT05] Zeyar Aung and Kian-Lee Tan. Automatic 3d protein structure classification without structural alignment. *Journal of Computational Biology*, 12(9):1221–1241, November 2005.

[ATG92] Nickolai N. Alexandrov, Katsutoshi Takahashi, and Nobuhiro Gō. Common spatial arrangements of backbone fragments in homologous and non-homologous proteins. *Journal of Molecular Biology*, 225(1):5–9, May 1992.

[AWCJ05] Robert J. Anderson, Zhiping Weng, Robert K. Campbell, and Xuliang Jiang. Main-chain conformational tendencies of amino acids. *Proteins: Structure, Function, and Bioinformatics*, 60(4):679–689, September 2005.

[BB03] Stephen K. Burley and Jeffrey B. Bonanno. Structural genomics. In Bourne and Weissig [BW03], pages 591–612.

[BBM$^+$97] Philip E. Bourne, Helen M. Berman, Brian McMahon, Keith D. Watenpaugh, John D. Westbrook, and Paula M. D. Fitzgerald. Macromolecular crystallographic information file. In Carter and Sweet [CS97], pages 571–590.

[BC01] Somenath Biswas and Samarjit Chakraborty. Fast algorithms for determining protein structure similarity. In HiPC 2001 [HiP01].

[BCHM96] Steven E. Brenner, Cyrus Chothia, Tim J. P. Hubbard, and Alexey G. Murzin. Understanding protein structure: Using Scop for fold interpretation. In Doolittle [Doo96], pages 635–643.

[BCK$^+$04] A. Bhattacharya, Tolga Can, Tamer Kahveci, Ambuj K. Singh, and Yuan-Fang Wang. ProGreSS: Simultaneous searching of protein databases by sequence and structure. In PSB 2004 [PSB04], pages 264–275.

[BEH89] Anselm Blumer, Andrzej Ehrenfeucht, and David Haussler. Average sizes of suffix trees and DAWGs. *Discrete Applied Mathematics*, 24(1):37–45, 1989.

[BFM97] Siegfried Böhm, Dmitrij Frishman, and Hans-Werner Mewes. Variations of the C2H2 zinc finger motif in the yeast genome and classification of yeast zinc finger proteins. *Nucleic Acids Research*, 25(12):2464–2469, June 1997.

[BG96] Peer Bork and Toby J. Gibson. Applying motif and profile searches. In Doolittle [Doo96], pages 162–184.

[BGR02] *Proceedings of the 3rd International Conference on Bioinformatics of Genome Regulation and Structure (BGRS'02)*, volume 3, July 2002.

[BIB03] *Proceedings of the 3rd IEEE Symposium on Bioinformatics and Bioengineering (BIBE'03)*, March 2003.

[BJVU98] Alvis Brāzma, Inge Jonassen, Jaak Vilo, and Esko Ukkonen. Pattern discovery in biosequences. In ICGI 1998 [ICG98], pages 257–270.

[BKB02] Phil Bradley, Peter S. Kim, and Bonnie Berger. Trilogy: Discovery of sequence-structure patterns across diverse proteins. In RECOMB 2002 [REC02], pages 77–88.

[BOB$^+$92] Helen M. Berman, Wilma K. Olson, David L. Beveridge, John D. Westbrook, Anke Gelbin, Tamas Demeny, Shu-Hsin Hsieh, Annankoil R. Srinivasan, and Bohdan Schneider. The nucleic acid database: A comprehensive relational database of three-dimensional structures of nucleic acids. *Biophysical Journal*, 63(3):751–759, September 1992.

[BRCR94] Paul Bieganski, John Riedl, John V. Carlis, and Ernest F. Retzel. Generalized suffix trees for biological sequence data: Applications and implementation. In HICSS 1994 [HIC94], pages 35–44.

[Brü03] Rafael Brüschweiler. Efficient RMSD measures for the comparison of two molecular ensembles. *Proteins: Structure, Function, and Genetics*, 50(1):26–34, January 2003.

[BS03] Philip E. Bourne and Ilya N. Shindyalov. Structure comparison and alignment. In Bourne and Weissig [BW03], pages 321–337.

[BT99] Carl Branden and John Tooze. *Introduction to Protein Structure.* Garland Publishing, New York, second edition, 1999.

[BT03a] Jonathan A. Barker and Janet M. Thornton. An algorithm for constraint-based structural template matching: application to 3d templates with statistical analysis. *Bioinformatics*, 19(13):1644–1649, September 2003.

[BT03b] Arno Buchner and Hanjo Täubig. A fast method for motif detection and searching in a protein structure database. Technical Report TUM-I0314, Inst. f. Informatik, TU München, September 2003.

[BTG03] Arno Buchner, Hanjo Täubig, and Jan Griebsch. A fast method for motif detection and searching in a protein structure database. In GCB 2003 [GCB03], pages 186–188.

[BW03] Philip E. Bourne and Helge Weissig, editors. *Structural Bioinformatics*, volume 44 of *Methods of Biochemical Analysis.* Wiley-Liss, 2003.

[BWF+00] Helen M. Berman, John D. Westbrook, Zukang Feng, Gary L. Gilliland, T. N. Bhat, Helge Weissig, Ilya N. Shindyalov, and Philip E. Bourne. The protein data bank. *Nucleic Acids Research*, 28(1):235–242, January 2000.

[BWF+03] Helen M. Berman, John D. Westbrook, Zukang Feng, Lisa Iype, Bohdan Schneider, and Christine Zardecki. The nucleic acid database. In Bourne and Weissig [BW03], pages 199–216.

[BY99] Gill Bejerano and Golan Yona. Modeling protein families using probabilistic suffix trees. In RECOMB 1999 [REC99], pages 15–24.

[BYG99] Ricardo Baeza-Yates and Gaston H. Gonnet. A fast algorithm on average for all-against-all sequence matching. In SPIRE 1999 [SPI99], pages 16–23.

[CB99] Samarjit Chakraborty and Somenath Biswas. Approximation algorithms for 3-d common substructure identification in drug and protein molecules. In WADS 1999 [WAD99], pages 253–264.

[CBP05] Mathilde Carpentier, Sophie Brouillet, and Joël Pothier. YAKUSA: A fast structural database scanning method. *Proteins: Structure, Function, and Bioinformatics*, 61(1):137–151, October 2005.

[CCC+04] Darby Tien-Hau Chang, Chien-Yu Chen, Wen-Chin Chung, Yen-Jen Oyang, Hsueh-Fen Juan, and Hsuan-Cheng Huang. ProteMiner-SSM: a web server for efficient analysis of similar protein tertiary substructures. *Nucleic Acids Research*, 32(Web Server Issue):W76–W82, July 2004.

[CD03] Adrian A. Canutescu and Roland L. Dunbrack, Jr. Cyclic coordinate descent: A robotics algorithm for protein loop closure. *Protein Science*, 12(5):963–972, May 2003.

[CFK+05] Brian Y. Chen, Viacheslav Y. Fofanov, David M. Kristensen, Marek Kimmel, Olivier Lichtarge, and Lydia E. Kavraki. Algorithms for structural comparison and statistical analysis of 3d protein motifs. In PSB 2005 [PSB05], pages 334–345.

[CGZ04] Matteo Comin, Concettina Guerra, and Giuseppe Zanotti. PROuST: A comparison method of three-dimensional structures of proteins using indexing techniques. *Journal of Computational Biology*, 11(6):1061–1072, December 2004.

[CHKK99] L. Paul Chew, Dan Huttenlocher, Klara Kedem, and Jon Kleinberg. Fast detection of common geometric substructure in proteins. In RECOMB 1999 [REC99], pages 104–113.

[Cho84] Cyrus Chothia. Principles that determine the structure of proteins. *Annual Review of Biochemistry*, 53:537–572, July 1984.

[CHTY03] Chem-Hooi Chionh, Zhiyong Huang, Kian-Lee Tan, and Zhen Yao. Augmenting SSEs with structural properties for rapid protein structure comparison. In BIBE 2003 [BIB03], pages 341–350.

[CIB04] IEEE Computational Intelligence Society. *Proceedings of the IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB'04)*, October 2004.

[ÇKS03a] Orhan Çamoğlu, Tamer Kahveci, and Ambuj K. Singh. PSI: indexing protein structures for fast similarity search. *Bioinformatics*, 19(Suppl. 1):i81–i83, July 2003.

[ÇKS03b] Orhan Çamoğlu, Tamer Kahveci, and Ambuj K. Singh. Towards index-based similarity search for protein structure databases. In CSB 2003 [CSB03], pages 148–158.

[CL86] Cyrus Chothia and Arthur M. Lesk. The relation between the divergence of sequence and structure in proteins. *The EMBO Journal*, 5(4):823–826, April 1986.

[CL02] Alberto Caprara and Giuseppe Lancia. Structural alignment of large-size proteins via lagrangian relaxation. In RECOMB 2002 [REC02], pages 100–108.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Cliff Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[Cob95] Archie L. Cobbs. Fast approximate matching using suffix trees. In CPM 1995 [CPM95], pages 41–54.

[CP01] Oliviero Carugo and Sándor Pongor. A normalized root-mean-square distance for comparing protein three-dimensional structures. *Protein Science*, 10(7):1470–1473, July 2001.

[CP02] Oliviero Carugo and Sándor Pongor. Protein fold similarity estimated by a probabilistic approach based on C_α-C_α distance comparison. *Journal of Molecular Biology*, 315(4):887–898, January 2002.

[CP03] Matt Coatney and Srinivasan Parthasarathy. MotifMiner: A general toolkit for efficiently identifying common substructures in molecules. In BIBE 2003 [BIB03], pages 336–340.

[CPM92] *Proceedings of the 3rd Symposium on Combinatorial Pattern Matching (CPM'92)*, volume 644 of *Lecture Notes in Computer Science*. Springer, April/May 1992.

[CPM93] *Proceedings of the 4th Symposium on Combinatorial Pattern Matching (CPM'93)*, volume 684 of *Lecture Notes in Computer Science*. Springer, June 1993.

[CPM95] *Proceedings of the 6th Symposium on Combinatorial Pattern Matching (CPM'95)*, volume 937 of *Lecture Notes in Computer Science*. Springer, July 1995.

[CPM04] *Proceedings of the 15th Symposium on Combinatorial Pattern Matching (CPM'04)*, volume 3109 of *Lecture Notes in Computer Science*. Springer, July 2004.

[Cre93] Thomas E. Creighton. *Proteins: Structures and Molecular Properties*. W. H. Freeman and Company, second edition, 1993.

[Cri58] Francis H. Crick. On protein synthesis. In *Symp. of the Society for Experimental Biology XII, The Biological Replication of Macromolecules*, pages 138–163, 1958.

[Cri70] Francis H. Crick. Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature*, 227(5258):561–563, August 1970.

[CS97] Charles W. Carter, Jr. and Robert M. Sweet, editors. *Macromolecular Crystallography*, volume 277 B of *Methods in Enzymology*. Academic Press, 1997.

[CSB03] IEEE Computer Society. *Proceedings of the IEEE Computational Systems Bioinformatics Conference (CSB'03)*, August 2003.

[CSB05] IEEE Computer Society. *Proceedings of the IEEE Computational Systems Bioinformatics Conference (CSB'05)*, August 2005.

[CSD04] Evangelos A. Coutsias, Chaok Seok, and Ken A. Dill. Using quaternions to calculate RMSD. *Journal of Computational Chemistry*, 25(15):1849–1857, November 2004.

[DAS03] IEEE. *Proceedings of the 8th International Conference on Database Systems for Advanced Applications (DASFAA'03)*, March 2003.

[DC93] Nigel J. Darby and Thomas E. Creighton. *Protein Structure*. In Focus. Oxford University Press, 1993.

[DEX03] *Proceedings of the 14th International Conference on Database and Expert Systems Applications (DEXA'03)*, volume 2736 of *Lecture Notes in Computer Science*. Springer, September 2003.

[DEX04] *Proceedings of the 15th International Conference on Database and Expert Systems Applications (DEXA'03)*, volume 3180 of *Lecture Notes in Computer Science*. Springer, August/September 2004.

[DFJZK94] Ding Da-Fu, Qian Jiang, and Feng Zu-Kang. A differential geometric treatment of protein structure comparison. *Bulletin of Mathematical Biology*, 56(5):923–943, September 1994.

[Dia66] R. Diamond. A mathematical model-building procedure for proteins. *Acta Crystallographica*, 21(2):253–266, August 1966.

[Dia76] R. Diamond. On the comparison of conformations using linear and quadratic transformations. *Acta Crystallographica*, A32(1):1–10, January 1976.

[Dia88] R. Diamond. A note on the rotational superposition problem. *Acta Crystallographica*, A44(2):211–216, March 1988.

[dlB59] Rene de la Briandais. File searching using variable length keys. In *Proceedings of the Western Joint Computer Conference*, pages 295–298, March 1959.

[dlCML97] Xavier F. de la Cruz, Michael W. Mahoney, and Byungkook Lee. Discrete representations of the protein C_α chain. *Folding and Design*, 2(4):223–234, August 1997.

[DM97] Bassil I. Dahiyat and Stephen L. Mayo. De novo protein design: Fully automated sequence selection. *Science*, 278(5335):82–87, October 1997.

[Doo96] Russell F. Doolittle, editor. *Computer Methods for Macromolecular Sequence Analysis*, volume 266 of *Methods in Enzymology*. Academic Press, 1996.

[DSM97] Bassil I. Dahiyat, Catherine A. Sarisky, and Stephen L. Mayo. De novo protein design: towards fully automated sequence selection. *Journal of Molecular Biology*, 273(4):789–796, November 1997.

[Eis03] David Eisenberg. The discovery of the α-helix and β-sheet, the principal structural features of proteins. *Proceedings of the National Academy of Sciences of the USA*, 100(20):11207–11210, September 2003.

[EJT99] Ingvar Eidhammer, Inge Jonassen, and William R. Taylor. Structure comparison and structure patterns. Technical Report 174, Department of Informatics, University of Bergen, Bergen, Norway, July 1999.

[EPSV98] Vincent Escalier, Joël Pothier, Henry Soldano, and Alain Viari. Pairwise and multiple identification of three-dimensional common substructures in proteins. *Journal of Computational Biology*, 5(1):41–56, 1998.

[Erd05] Michael A. Erdmann. Protein similarity from knot theory: Geometric convolution and line weavings. *Journal of Computational Biology*, 12(6):609–637, July 2005.

[Ern03] Jens Ernst. *Similarity-Based Clustering Algorithms for Gene Expression Profiles.* PhD thesis, Fakultät für Informatik, Technische Universität München, Garching / München, Germany, 2003.

[FAK02] Michel N. Fodje and Salam Al-Karadaghi. Occurrence, conformational features and amino acid propensities for the π-helix. *Protein Engineering*, 15(5):353–358, May 2002.

[FH77] Dino R. Ferro and Jan Hermans. A different best rigid-body molecular fit routine. *Acta Crystallographica*, A33:345–347, 1977.

[Fin97] Barry C. Finzel. LORE: Exploiting database of known structures. In Carter and Sweet [CS97], pages 230–242.

[Fre60] Edward Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, September 1960.

[GCB03] *Proceedings of the German Conference on Bioinformatics (GCB'03)*, October 2003.

[GCB04] *Proceedings of the German Conference on Bioinformatics (GCB'04)*, volume P-53 of *Lecture Notes in Informatics.* Köllen Verlag, October 2004.

[GG89] Zvi Galil and Raffaele Giancarlo. Speeding up dynamic programming with applications to molecular biology. *Theoretical Computer Science*, 64(1):107–118, April 1989.

[GK97] Robert Giegerich and Stefan Kurtz. From Ukkonen to McCreight and Weiner: A unifying view of linear-time suffix tree construction. *Algorithmica*, 19(3):331–353, November 1997.

[GLZ02] Concettina Guerra, Stefano Lonardi, and Giuseppe Zanotti. Analysis of secondary structure elements of proteins using indexing techniques. In 3DPVT 2002 [3DP02], pages 812–823.

[GM87] Paul R. Gerber and Klaus Müller. Superimposing several sets of atomic coordinates. *Acta Crystallographica*, A43(3):426–428, May 1987.

[GMB96] Jean-Francois Gibrat, Thomas Madej, and Stephen H. Bryant. Surprising similarities in structure comparison. *Current Opinion in Structural Biology*, 6(3):377–385, June 1996.

[God96] Adam Godzik. The structural alignment between two proteins: Is there a unique answer? *Protein Science*, 5(7):1325–1338, July 1996.

[Got82] Osamu Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, December 1982.

[GPK00] Kunchur Guruprasad, Maheshuni S. Prasad, and Gundu R. Kumar. Database of structural motifs in proteins. *Bioinformatics*, 16(4):372–375, April 2000.

[Gre85] Jonathan Greer. Computer skeletonization and automatic electron density map analysis. In Wyckoff et al. [WHT85], pages 206–224.

[GSC$^+$96] Anke Gelbin, Bohdan Schneider, Lester Clowney, Shu-Hsin Hsieh, Wilma K. Olson, and Helen M. Berman. Geometric parameters in nucleic acids: Sugar and phosphate constituents. *Journal of the American Chemical Society*, 118(3):519–529, January 1996.

[Gus97] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences – Computer Science and Computational Biology.* Cambridge University Press, 1997.

[GVP05] Zoltán Gáspári, Kristian Vlahovicek, and Sándor Pongor. Efficient recognition of folds in protein 3d structures by the improved PRIDE algorithm. *Bioinformatics*, 21(15):3322–3323, August 2005.

[Gwe68] Gernot Gwehenberger. Anwendung einer binären Verweiskettenmethode beim Aufbau von Listen (Use of a binary tree structure for processing files). *Elektronische Rechenanlagen*, 10(5):223–226, October 1968.

[GWNT99] David R. Gilbert, David R. Westhead, Nozomi Nagano, and Janet M. Thornton. Motif-based searching in TOPS protein topology databases. *Bioinformatics*, 15(4):317–326, April 1999.

[GWT98] David R. Gilbert, David R. Westhead, and Janet M. Thornton. A constraint-based system for protein motif-searching, pattern discovery and structure comparison. In *Proc. of the ERCIM/COMPULOG Workshop on Constraints*, September 1998.

[GZ05] Feng Gao and Mohammed J. Zaki. PSIST: Indexing protein structures using suffix trees. In CSB 2005 [CSB05], pages 212–222.

[HAI02] Ela Hunt, Malcolm P. Atkinson, and Robert W. Irving. Database indexing for large DNA and protein sequence collections. *The VLDB Journal*, 11(3):256–271, November 2002.

[Ham50] Richard W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, April 1950.

[HBW$^+$05] Jun Huan, Deepak Bandyopadhyay, Wei Wang, Jack Snoeyink, Jan Prins, and Alexander Tropsha. Comparing graph representations of protein structure for mining family-specific residue-based packing motifs. *Journal of Computational Biology*, 12(6):657–671, July 2005.

[HC52] Alfred D. Hershey and Martha Chase. Independent functions of viral protein and nucleic acid in growth of bacteriophage. *The Journal of General Physiology*, 36(1):39–56, September 1952.

[HIC94] IEEE. *Proceedings of the 27th Annual Hawaii International Conference on System Sciences (HICSS'94)*, volume V (Biotechnology Computing), 1994.

[HiP01] *Proceedings of the Workshop on Bioinformatics and Computational Biology at the 8th International Conference on High Performance Computing (HiPC 2001)*, December 2001.

[HLS$^+$95] Doug L. Hoffman, S. Laiter, Raj K. Singh, I. I. Vaisman, and Alexander Tropsha. Rapid protein structure classification using one-dimensional structure profiles on the BioSCAN parallel computer. *Computer Applications in the Biosciences*, 11(6):675–679, December 1995.

[Hof96] Doug L. Hoffman. *Comparison of Protein Structures by Transformation into Dihedral Angle Sequences*. PhD thesis, University of North Carolina, Chapel Hill, NC, USA, 1996.

[Hor87a] Berthold K. P. Horn. Closed-form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America A*, 5(7):1127–1135, May 1987.

[Hor87b] Berthold K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, April 1987.

[HS93] Liisa Holm and Chris Sander. Protein structure comparison by alignment of distance matrices. *Journal of Molecular Biology*, 233(1):123–138, September 1993.

[HS94] Liisa Holm and Chris Sander. Searching protein structure databases has come of age. *Proteins: Structure, Function, and Genetics*, 19(3):165–173, July 1994.

[HS95] Liisa Holm and Chris Sander. 3-D Lookup: Fast protein structure database searches at 90% reliability. In ISMB 1995 [ISM95], pages 179–187.

[HS96] Liisa Holm and Chris Sander. Mapping the protein universe. *Science*, 273(5275):595–602, August 1996.

[HS97] Liisa Holm and Chris Sander. Dali/FSSP classification of three-dimensional protein folds. *Nucleic Acids Research*, 25(1):231–234, January 1997.

[HSV97] Rob W. W. Hooft, Chris Sander, and Gerrit Vriend. Objectively judging the quality of a protein structure from a Ramachandran plot. *Computer Applications in the Biosciences*, 13(4):425–430, August 1997.

[HT96] E. Gail Hutchinson and Janet M. Thornton. PROMOTIF – a program to identify and analyze structural motifs in proteins. *Protein Science*, 5(2):212–220, February 1996.

[Hui92] Lucas Chi Kwong Hui. Color set size problem with applications to string matching. In CPM 1992 [CPM92], pages 230–243.

[ICG98] *Proceedings of the 4th International Colloquium on Grammatical Inference (ICGI'98)*, Lecture Notes in Artificial Intelligence. Springer, July 1998.

[IDE03] *Proceedings of the 4th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL'03)*, volume 2690 of *Lecture Notes in Computer Science*. Springer, March 2003.

[IDE04] *Proceedings of the 5th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL'03)*, volume 3177 of *Lecture Notes in Computer Science*. Springer, August 2004.

[IDP$^+$02] V. A. Ivanisenko, V. A. Debelov, S. S. Pintus, S. V. Nikolaev, D. A. Grigorovich, and Nikolay A. Kolchanov. PDBSiteScan: A tool for search for the best-matching superposition in the database PDBSite. In BGRS 2002 [BGR02], pages 150–153.

[IJC03] IEEE. *Proceedings of the International Joint Conference on Neural Networks (IJCNN'03)*, volume 4, 2003.

[IMP$^+$04] Costas S. Iliopoulos, James McHugh, Pierre Peterlongo, Nadia Pisanti, Wojciech Rytter, and Marie-France Sagot. A first approach to finding common motifs with gaps. In PSC 2004 [PSC04], pages 88–97.

[ISM95] *Proceedings of the 3rd International Conference on Intelligent Systems for Molecular Biology (ISMB'95).* AAAI Press, July 1995.

[ISM97] *Proceedings of the 5th International Conference on Intelligent Systems for Molecular Biology (ISMB'97).* AAAI Press, June 1997.

[JC85] Joël Janin and Cyrus Chothia. Domains in proteins: Definitions, location, and structural principles. In Wyckoff et al. [WHT85], pages 420–430.

[JD05] Yuting Jia and T. Gregory Dewey. A random polymer model of the statistical significance of structure alignment. *Journal of Computational Biology*, 12(3):298–313, April 2005.

[JHF03] Andrew I. Jewett, Conrad C. Huang, and Thomas E. Ferrin. MINRMS: an efficient algorithm for determining protein structure similarity using root-mean-squared-distance. *Bioinformatics*, 19(5):625–634, March 2003.

[JK97] T. Alwyn Jones and Morten Kjeldgaard. Electron-density map interpretation. In Carter and Sweet [CS97], pages 173–208.

[JOE+94] Mark S. Johnson, John P. Overington, Yvonne Edwards, Alex C. W. May, and Michael A. Rodionov. The comparison of structures and sequences: alignment, searching and the detection of common folds. In HICSS 1994 [HIC94], pages 296–305.

[JSRS05] Kyle L. Jensen, Mark P. Styczynski, Isidore Rigoutsos, and Gregory N. Stephanopoulos. A generic motif discovery algorithm for sequential data. *Bioinformatics*, 2005.

[JT86] T. Alwyn Jones and Sören Thirup. Using known substructures in protein model building and crystallography. *The EMBO Journal*, 5(4):819–822, April 1986.

[Kab76] Wolfgang Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica*, A32(5):922–923, September 1976.

[Kab78] Wolfgang Kabsch. A discussion of the solution for the best rotation to relate two sets of vectors. *Acta Crystallographica*, A34(5):827–828, September 1978.

[Kaw03] Takeshi Kawabata. MATRAS: a program for protein 3D structure comparison. *Nucleic Acids Research*, 31(13):3367–3369, July 2003.

[KBD+58] John C. Kendrew, G. Bodo, H. M. Dintzis, R. G. Parrish, H. Wyckoff, and D. C. Phillips. A three-dimensional model of the myoglobin molecule obtained by x-ray analysis. *Nature*, 181(4610):662–666, March 1958.

[KdHN89] Mary E. Karpen, Pieter L. de Haseth, and Kenneth E. Neet. Comparing short protein substructures by a method based on backbone torsion angles. *Proteins: Structure, Function, and Genetics*, 6(2):155–167, 1989.

[Kea89] Simon K. Kearsley. On the orthogonal transformation used for structural comparisons. *Acta Crystallographica*, A45(2):208–210, February 1989.

[KGLK05] Rachel Kolodny, Leonidas Guibas, Michael Levitt, and Patrice Koehl. Inverse kinematics in biology: The protein loop closure problem. *The International Journal of Robotics Research*, 24(2-3):151–163, February/March 2005.

[KH04] Eugene Krissinel and Kim Henrick. Secondary-structure matching (SSM), a new tool for fast protein structure alignment in three dimensions. *Acta Crystallographica*, D60(12-1):2256–2268, December 2004.

[KJ97] Gerard J. Kleywegt and T. Alwyn Jones. Detecting folding motifs and similarities in protein structures. In Carter and Sweet [CS97], pages 525–545.

[KL97] Ina Koch and Thomas Lengauer. Detection of distant structural similarities in a set of proteins using a fast graph-based method. In ISMB 1997 [ISM97], pages 167–178.

[KL04] Rachel Kolodny and Nathan Linial. Approximate protein structural alignment in polynomial time. *Proceedings of the National Academy of Sciences of the USA*, 101(33):12201–12206, August 2004.

[Kle97] Gerard J. Kleywegt. Validation of protein models from C^α coordinates alone. *Journal of Molecular Biology*, 273(2):371–376, October 1997.

[Kle99] Gerard J. Kleywegt. Recognition of spatial motifs in protein structures. *Journal of Molecular Biology*, 285(4):1887–1897, January 1999.

[KLW96] Ina Koch, Thomas Lengauer, and Egon Wanke. An algorithm for finding maximal common subtopologies in a set of protein structures. *Journal of Computational Biology*, 3(2):289–306, 1996.

[KMR72] Richard M. Karp, Raymond E. Miller, and Arnold L. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. In STOC 1972 [STO72], pages 125–136.

[KN00] Takeshi Kawabata and Ken Nishikawa. Protein structure comparison using the Markov transition model of evolution. *Proteins: Structure, Function, and Genetics*, 41(1):108–122, October 2000.

[Knu98] Donald E. Knuth. *The Art of Computer Programming – Sorting and Searching*, volume 3. Addison-Wesley, second edition, 1998.

[KP60] William Klyne and Vladimir Prelog. Description of steric relationships across single bonds. *Experientia*, 16(12):521–523, December 1960.

[KP04] Natalio Krasnogor and David A. Pelta. Measuring the similarity of protein structures by means of the universal similarity metric. *Bioinformatics*, 20(7):1015–1021, May 2004.

[Kur99] Stefan Kurtz. Reducing the space requirement of suffix trees. *Software: Practice and Experience*, 29(13):1149–1171, 1999.

[LC05] Sangyoon Lee and Gregory S. Chirikjian. Pose analysis of alpha-carbons in proteins. *The International Journal of Robotics Research*, 24(2-3):183–210, February/March 2005.

[LDA$^+$03] Simon C. Lovell, Ian W. Davis, W. Bryan Arendall III, Paul I. W. de Bakker, J. Michael Word, Michael G. Prisant, Jane S. Richardson, and David C. Richardson. Structure validation by C_α geometry: ϕ,ψ and C_β deviation. *Proteins: Structure, Function, and Genetics*, 50(3):437–450, February 2003.

[Les79] Arthur M. Lesk. Detection of three-dimensional patterns of atoms in chemical structures. *Communications of the ACM*, 22(4):219–224, April 1979.

[Lev65] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Doklady Akademii Nauk SSSR*, 163(4):845–848, August 1965.

[Lev76] Michael Levitt. A simplified representation of protein conformations for rapid simulation of protein folding. *Journal of Molecular Biology*, 104(1):59–107, June 1976.

[LG98] Michael Levitt and Mark Gerstein. A unified statistical framework for sequence comparison and structure comparison. *Proceedings of the National Academy of Sciences of the USA*, 95(11):5913–5920, May 1998.

[LKSD00] Peter Lackner, Walter A. Koppensteiner, Manfred J. Sippl, and Francisco S. Domingues. ProSup: a refined tool for protein structure alignment. *Protein Engineering*, 13(11):745–752, November 2000.

[LNW01] Nathaniel Leibowitz, Ruth Nussinov, and Haim J. Wolfson. MUSTA - a general, efficient, automated method for multiple structure alignment and detection of common motifs: Application to proteins. *Journal of Computational Biology*, 8(2):93–121, April 2001.

[Lot04] Itay Lotan. *Algorithms Exploiting the Chain Structure of Proteins.* PhD thesis, Department of Computer Science, Stanford University, Stanford, CA, USA, 2004.

[LP85] David J. Lipman and William R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, March 1985.

[LP01] Hongyuan Li and Srinivasan Parthasarathy. Deriving multi-level protein structures through data mining. In HiPC 2001 [HiP01].

[LS04] Itay Lotan and Fabian Schwarzer. Approximation of protein structure for fast similarity measures. *Journal of Computational Biology*, 11(2-3):299–317, March 2004.

[LSW84] M. Levine, D. Stuart, and J. Williams. A method for the systematic comparison of the three-dimensional structures of proteins and some results. *Acta Crystallographica*, A40(5):600–610, September 1984.

[Lu00] Guoguang Lu. TOP: a new method for protein structure comparisons and similarity searches. *Journal of Applied Crystallography*, 33(1):176–183, February 2000.

[Maa04] Moritz Maaß. Average-case analysis of approximate trie search. In CPM 2004 [CPM04], pages 472–483.

[Mac84] Alan L. Mackay. Quaternion transformation of molecular orientation. *Acta Crystallographica*, A40(2):165–166, March 1984.

[Mar00] Andrew C.R. Martin. The ups and downs of protein topology; rapid comparison of protein structure. *Protein Engineering*, 13(12):829–837, December 2000.

[MBHC95] Alexey G. Murzin, Steven E. Brenner, Tim J. P. Hubbard, and Cyrus Chothia. SCOP: A structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247(4):536–540, April 1995.

[McC76] Edward M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, April 1976.

[McL72] A. D. McLachlan. A mathematical procedure for superimposing atomic coordinates of proteins. *Acta Crystallographica*, A28(6):656–657, November 1972.

[MK02] Dennis Madsen and Gerard J. Kleywegt. Interactive motiv and fold recognition in protein structures. *Journal of Applied Crystallography*, 35(1):137–139, February 2002.

[Mor68] Donald R. Morrison. PATRICIA – practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM*, 15(4):514–534, October 1968.

[MR85] Brian W. Matthews and Michael G. Rossmann. Comparison of protein structures. In Wyckoff et al. [WHT85], pages 397–420.

[MS00] Laurent Marsan and Marie-France Sagot. Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification. *Journal of Computational Biology*, 7(3):345–362, August 2000.

[MTGW04] Ioannis Michalopoulos, Gilleain M. Torrance, David R. Gilbert, and David R. Westhead. TOPS: An enhanced database of protein structural topology. *Nucleic Acids Research*, 32(Database Issue):D251–D254, January 2004.

[MTT04] Kevin B. Murray, William R. Taylor, and Janet M. Thornton. Toward the detection and validation of repeats in protein structure. *Proteins: Structure, Function, and Bioinformatics*, 57(2):365–380, November 2004.

[Mul04] Lisa Mullan. Domains and motifs — proteins in bite-sized chunks. *Briefings in Bioinformatics*, 5(1):71–74, March 2004.

[NBG$^+$02] Giri Narasimhan, Changsong Bu, Yuan Gao, Xuning Wang, Ning Xu, and Kalai Mathee. Mining protein sequences for motifs. *Journal of Computational Biology*, 9(5):707–720, October 2002.

[NBYST01] Gonzalo Navarro, Ricardo Baeza-Yates, Erkki Sutinen, and Jorma Tarhio. Indexing methods for approximate string matching. *Bulletin of the Technical Committee on Data Engineering*, 24(4):19–27, December 2001.

[NK05] Marian Novotny and Gerard J. Kleywegt. A survey of left-handed helices in protein structures. *Journal of Molecular Biology*, 347(2):231–241, March 2005.

[NMK04] Marian Novotny, Dennis Madsen, and Gerard J. Kleywegt. Evaluation of protein fold comparison servers. *Proteins: Structure, Function, and Bioinformatics*, 54(2):260–270, February 2004.

[NO74] Ken Nishikawa and Tatsuo Ooi. Comparison of homologous tertiary structures of proteins. *Journal of Theoretical Biology*, 43(2):351–374, February 1974.

[NSB03] Stephen Neidle, Bohdan Schneider, and Helen M. Berman. Fundamentals of DNA and RNA structure. In Bourne and Weissig [BW03], pages 41–73.

[NW70] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, March 1970.

[OH94] T. J. Oldfield and R. E. Hubbard. Analysis of C_α geometry in protein structures. *Proteins: Structure, Function, and Genetics*, 18(4):324–337, April 1994.

[OKA03] Stephen D. O'Hearn, Anthony J. Kusalik, and Joseph F. Angel. MolCom: a method to compare protein molecules based on 3-d structural and chemical similarity. *Protein Engineering*, 16(2):169–178, February 2003.

[OMJ$^+$97] Christine A. Orengo, Alex D. Michie, Susan Jones, David T. Jones, Mark B. Swindells, and Janet M. Thornton. CATH – a hierarchic classification of protein domain structures. *Structure*, 5(8):1093–1108, August 1997.

[OPB$^+$99] Christine A. Orengo, Frances M. G. Pearl, James E. Bray, Annabel E. Todd, Andrew C.R. Martin, Loredana Lo Conte, and Janet M. Thornton. The CATH database provides insights into protein structure/function relationships. *Nucleic Acids Research*, 27(1):275–279, January 1999.

[OPT03] Christine A. Orengo, Frances M. G. Pearl, and Janet M. Thornton. The CATH domain structure database. In Bourne and Weissig [BW03], pages 249–271.

[PA98] Xavier Pennec and Nicholas Ayache. A geometric algorithm to find small but highly similar 3D substructures in proteins. *Bioinformatics*, 14(6):516–522, July 1998.

[PC50] Linus Pauling and Robert B. Corey. Two hydrogen-bonded spiral configurations of the polypeptide chain. *Journal of the American Chemical Society*, 72(11):5349, November 1950.

[PC51] Linus Pauling and Robert B. Corey. Configurations of polypeptide chains with favored orientations around single bonds: Two new pleated sheets. *Proceedings of the National Academy of Sciences of the USA*, 37(11):205–211, November 1951.

[PCB51] Linus Pauling, Robert B. Corey, and Herman R. Branson. The structure of proteins: Two hydrogen-bonded helical configurations of the polypeptide chain. *Proceedings of the National Academy of Sciences of the USA*, 37(4):205–211, April 1951.

[PCCS03] Alberto Paccanaro, Chakra Chennubhotla, James A. Casbon, and Mansoor A. S. Saqi. Spectral clustering of protein sequences. In IJCNN 2003 [IJC03], pages 3083–3088.

[PDB03] PDB Team. The protein data bank. In Bourne and Weissig [BW03], pages 181–198.

[Pea97] William R. Pearson. Identifying distantly related protein sequences. *Computer Applications in the Biosciences*, 13(4):325–332, August 1997.

[Per51] Max F. Perutz. New x-ray evidence on the configuration of polypeptide chains. *Nature*, 167(4261):1053–1054, June 1951.

[PES91] Robert Preißner, Ursula Egner, and Wolfram Saenger. Occurrence of bifurcated three-center hydrogen bonds in proteins. *FEBS Letters*, 288(1-2):192–196, August 1991.

[Pet91] Gregory A. Petsko. Déjà vu all over again. *Nature*, 352(6331):104–105, July 1991.

[PGR$^+$01] Robert Preißner, Andrean Goede, Kristian Rother, Frank Osterkamp, Ulrich Koert, and Cornelius Frömmel. Matching organic libraries with protein-substructures. *Journal of Computer-Aided Molecular Design*, 15(9):811–817, September 2001.

[Phi70] D. C. Phillips. The development of crystallographic enzymology. In *Biochem. Soc. Symp.*, volume 30, pages 11–28, 1970.

[PL88] William R. Pearson and David J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences of the USA*, 85(8):2444–2448, April 1988.

[PPBS05] Pierre Peterlongo, Nadia Pisanti, Frederic Boyer, and Marie-France Sagot. Lossless filter for finding long multiple approximate repetitions using a new data structure, the bi-factor array. In SPIRE 2005 [SPI05], pages 179–190.

[PPvGR02] Dariusz Plewczynski, Jakub Pas, Marcin von Grotthuss, and Leszek Rychlewski. 3D-Hit: fast structural comparison of proteins. *Applied Bioinformatics*, 1(4):223–225, 2002.

[PR04a] Sung-Hee Park and Keun Ho Ryu. Effective filtering for structural similarity search in protein 3d structure databases. In DEXA 2004 [DEX04], pages 761–770.

[PR04b] Sung-Hee Park and Keun Ho Ryu. Fast filtering of structural similarity search using discovery of topological patterns. In IDEAL 2004 [IDE04], pages 396–401.

[PR04c] Sung-Hee Park and Keun Ho Ryu. Fast similarity search for protein 3d structure databases using spatial topological patterns. In DEXA 2004 [DEX04], pages 771–780.

[PR04d] Gregory A. Petsko and Dagmar Ringe. *Protein Structure and Function.* New Science Press, London, UK, 2004.

[PRS03a] Sung-Hee Park, Keun Ho Ryu, and Hyeon S. Son. Protein structural information management based on spatial concepts and active trigger rules. In DEXA 2003 [DEX03], pages 413–422.

[PRS03b] Sung-Hee Park, Keun Ho Ryu, and Hyeon S. Son. Protein structure modeling using a spatial model for structure comparison. In IDEAL 2003 [IDE03], pages 490–497.

[PSB99] *Proceedings of the 4th Pacific Symposium on Biocomputing (PSB'99).* World Scientific Press, January 1999.

[PSB03] *Proceedings of the 8th Pacific Symposium on Biocomputing (PSB'03).* World Scientific Press, January 2003.

[PSB04] *Proceedings of the 9th Pacific Symposium on Biocomputing (PSB'04).* World Scientific Press, January 2004.

[PSB05] *Proceedings of the 10th Pacific Symposium on Biocomputing (PSB'05).* World Scientific Press, January 2005.

[PSC04] *Proceedings of the 9th Prague Stringology Conference (PSC'04)*, August/September 2004.

[RA76] Michael G. Rossmann and Patrick Argos. Exploring structural homology of proteins. *Journal of Molecular Biology*, 105(1):75–95, July 1976.

[RB03] Boojala V. B. Reddy and Philip E. Bourne. Protein structure evolution and the SCOP database. In Bourne and Weissig [BW03], pages 239–248.

[RDR$^+$02] S. Robin, J.-J. Daudin, H. Richard, Marie-France Sagot, and S. Schbath. Occurence probability of structured motifs in random sequences. *Journal of Computational Biology*, 9(6):761–773, December 2002.

[REC99] *Proceedings of the 3rd Annual International Conference on Research in Computational Molecular Biology (RECOMB'99).* ACM Press, April 1999.

[REC02] *Proceedings of the 6th Annual International Conference on Research in Computational Molecular Biology (RECOMB'02).* ACM Press, April 2002.

[Ric85] Jane S. Richardson. Describing patterns of protein tertiary structure. In Wyckoff et al. [WHT85], pages 341–358.

[RM80] S. James Remington and Brian W. Matthews. A systematic approach to the comparison of protein structures. *Journal of Molecular Biology*, 140(1):77–99, June 1980.

[Röm04] Uwe Römers. Motiv-Suche in Proteinstrukturdatenbanken unter Berücksichtigung von Deletionen und Insertionen. Diploma thesis, Technische Universität München, Garching, Germany, September 2004. (In German).

[Ros85] George D. Rose. Automatic recognition of domains in globular proteins. In Wyckoff et al. [WHT85], pages 430–440.

[Ros99] Burkhard Rost. Twilight zone of protein sequence alignment. *Protein Engineering*, 12(2):85–94, February 1999.

[RR85] Jane S. Richardson and David C. Richardson. Interpretation of electron density maps. In Wyckoff et al. [WHT85], pages 189–206.

[RRS63] Gopalasamudram N. Ramachandran, Chandrashekharan Ramakrishnan, and Visvanathan Sasisekharan. Stereochemistry of polypeptide chain configurations. *Journal of Molecular Biology*, 7:95–99, July 1963.

[RS68] Gopalasamudram N. Ramachandran and Visvanathan Sasisekharan. Conformation of polypeptides and proteins. *Advances in Protein Chemistry*, 23:283–438, 1968.

[SB98] Ilya N. Shindyalov and Philip E. Bourne. Protein structure alignment by incremental combinatorial extension (CE) of the optimal path. *Protein Engineering*, 11(9):739–747, September 1998.

[SB04a] Jessica Shapiro and Douglas Brutlag. FoldMiner and LOCK 2: protein structure comparison and motif discovery on the web. *Nucleic Acids Research*, 32(Web Server Issue):W536–W541, July 2004.

[SB04b] Jessica Shapiro and Douglas Brutlag. FoldMiner: Structural motif discovery using an improved superposition algorithm. *Protein Science*, 13(1):278–294, January 2004.

[SCH99] Shaobing Su, Diane J. Cook, and Lawrence B. Holder. Applications of knowledge discovery to molecular biology: Identifying structural regularities in proteins. In PSB 1999 [PSB99], pages 190–201.

[SCH$^+$02] Christian J. Sigrist, Lorenzo Cerutti, Nicolas Hulo, Alexandre Gattiker, Laurent Falquet, Marco Pagni, Amos Bairoch, and Philipp Bucher. PROSITE: A documented database using patterns and profiles as motif descriptors. *Briefings in Bioinformatics*, 3(3):265–274, September 2002.

[SDSD$^+$04] Maxim Shatsky, Oranit Dror, Dina Schneidman-Duhovny, Ruth Nussinov, and Haim J. Wolfson. BioInfo3D: a suite of tools for structural bioinformatics. *Nucleic Acids Research*, 32(Web Server Issue):W503–W507, July 2004.

[SF03] Eric D. Scheeff and J. Lynn Fink. Fundamentals of protein structure. In Bourne and Weissig [BW03], pages 15–39.

[SGJT04] Hugh P. Shanahan, Mario A. Garcia, Susan Jones, and Janet M. Thornton. Identifying DNA-binding proteins using structural motifs and the electrostatic potential. *Nucleic Acids Research*, 32(16):4732–4741, September 2004.

[SH03] Edward S. C. Shih and Ming-Jing Hwang. Protein structure comparison by probability-based matching of secondary structure elements. *Bioinformatics*, 19(6):735–741, April 2003.

[Sip82] Manfred J. Sippl. On the problem of comparing protein structures: Development and applications of a new method for the assessment of structural similarities of polypeptide conformations. *Journal of Molecular Biology*, 156(2):359–388, April 1982.

[SK04] Michael L. Sierk and Gerard J. Kleywegt. Déjà vu all over again: Finding and analyzing protein structure similarities. *Structure*, 12(12):2103–2111, December 2004.

[SMB04] Bohdan Schneider, Zdeněk Morávek, and Helen M. Berman. RNA conformational classes. *Nucleic Acids Research*, 32(5):1666–1677, March 2004.

[Smi04] Scott F. Smith. Protein family classification using structural and sequence information. In CIBCB 2004 [CIB04], pages 168–174.

[SPI99] *Proceedings of the 6th Symposium on String Processing and Information Retrieval (SPIRE'99)*, September 1999.

[SPI05] *Proceedings of the 12th Symposium on String Processing and Information Retrieval (SPIRE'05)*, volume 3772 of *Lecture Notes in Computer Science*. Springer, November 2005.

[SR03] Alexander Stark and Robert B. Russell. Annotation in three dimensions. PINTS: Patterns in non-homologous tertiary structures. *Nucleic Acids Research*, 31(13):3341–3344, July 2003.

[SS03] Rohit Singh and Mitul Saha. Identifying structural motifs in proteins. In PSB 2003 [PSB03], pages 228–239.

[STO72] *Proceedings of the 4th Annual ACM Symposium on Theory of Computing (STOC'72)*, May 1972.

[SVPS95] Marie-France Sagot, Alain Viari, Joël Pothier, and Henry Soldano. Finding flexible patterns in a text – an application to 3d molecular matching. *Computer Applications in the Biosciences*, 11(1):59–70, February 1995.

[SVS97] Marie-France Sagot, Alain Viari, and Henry Soldano. Multiple sequence comparison – a peptide matching approach. *Theoretical Computer Science*, 180(1-2):115–137, June 1997.

[SW81] Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, March 1981.

[Tan04] Thomas Tang. *Discovering Protein Sequence-Structure Motifs and Two Applications to Structural Prediction.* PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 2004.

[TBG04] Hanjo Täubig, Arno Buchner, and Jan Griebsch. A method for fast approximate searching of polypeptide structures in the PDB. In GCB 2004 [GCB04], pages 65–74.

[TG89] Janet M. Thornton and Stephen P. Gardner. Protein motifs and data-base searching. *Trends in Biochemical Sciences*, 14(7):300–304, July 1989.

[The05] Douglas L. Theobald. Rapid calculation of RMSDs using a quaternion-based characteristic polynomial. *Acta Crystallographica*, A61(4):478–480, July 2005.

[Toh97] Hiroyuki Toh. Introduction of a distance cut-off into structural alignment by the double dynamic programming algorithm. *Computer Applications in the Biosciences*, 13(4):387–396, August 1997.

[TXL05] Thomas Tang, Jinbo Xu, and Ming Li. Discovering sequence-structure motifs from protein segments and two applications. In PSB 2005 [PSB05], pages 370–381.

[Ukk93] Esko Ukkonen. Approximate string-matching over suffix trees. In CPM 1993 [CPM93], pages 228–242.

[Ukk95] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, September 1995.

[Ume91] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):376–, April 1991.

[VBK02] Saraswathi Vishveshwara, K. V. Brinda, and N. Kannan. Protein structure: Insights from graph theory. *Journal of Theoretical and Computational Chemistry*, 1(1):187–211, July 2002.

[VG01] Juris Vīksna and David R. Gilbert. Pattern matching and pattern discovery algorithms for protein topologies. In WABI 2001 [WAB01], pages 98–111.

[vHJH98] Kensal E. van Holde, W. Curtis Johnson, and P. Shing Ho. *Principles of Physical Biochemistry*. Prentice-Hall, 1998.

[Vil02] Jaak Vilo. *Pattern Discovery from Biosequences*. PhD thesis, University of Helsinki, Helsinki, Finland, 2002.

[WAB01] *Proceedings of the 1st Workshop on Algorithms in BioInformatics (WABI'01)*, volume 2149 of *Lecture Notes in Computer Science*, August 2001.

[WAD99] *Proceedings of the 6th International Workshop on Algorithms and Data Structures (WADS'99)*, volume 1663 of *Lecture Notes in Computer Science*, August 1999.

[WB03] Helge Weissig and Philip E. Bourne. Other structure-based databases. In Bourne and Weissig [BW03], pages 217–236.

[WBT97] Andrew C. Wallace, Neera Borkakoti, and Janet M. Thornton. TESS: A geometric hashing algorithm for deriving 3d coordinate templates for searching structural databases. application to enzyme active sites. *Protein Science*, 6(11):2308–2323, November 1997.

[WC53] James D. Watson and Francis H. Crick. Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature*, 171(4356):737–738, April 1953.

[Wei73] Peter Weiner. Linear pattern matching algorithms. In *Proceedings of the 14th IEEE Annual Symposium on Switching and Automata Theory*, pages 1–11. IEEE, 1973.

[WHT85] Harold W. Wyckoff, C. H. W. Hirs, and Serge N. Timasheff, editors. *Diffraction Methods for Biological Macromolecules*, volume 115 B of *Methods in Enzymology*. Academic Press, 1985.

[WKHK03] Nils Weskamp, Daniel Kuhn, Eyke Hüllermeier, and Gerhard Klebe. Efficient similarity search in protein structure databases: Improving clique-detection through clique hashing. In GCB 2003 [GCB03], pages 179–184.

[WKHK04] Nils Weskamp, Daniel Kuhn, Eyke Hüllermeier, and Gerhard Klebe. Efficient similarity search in protein structure databases by k-clique hashing. *Bioinformatics*, 20(10):1522–1526, July 2004.

[WLT96] Andrew C. Wallace, Roman A. Laskowski, and Janet M. Thornton. Derivation of 3d coordinate templates for searching structural databases: Application to ser-his-asp catalytic triads in the serine proteinases and lipases. *Protein Science*, 5(6):1001–1013, June 1996.

[WW03] Lorenz Wernisch and Shoshana J. Wodak. Identifying structural domains in proteins. In Bourne and Weissig [BW03], pages 365–385.

[YG03] Yuzhen Ye and Adam Godzik. Flexible structure alignment by chaining aligned fragment pairs allowing twists. *Bioinformatics*, 19(Suppl. 2):ii246–ii255, September 2003.

[YG04a] Yuzhen Ye and Adam Godzik. Database searching by flexible protein structure alignment. *Protein Science*, 13(7):1841–1850, July 2004.

[YG04b] Yuzhen Ye and Adam Godzik. FATCAT: a web server for flexible structure comparison and structure similarity searching. *Nucleic Acids Research*, 32(Web Server Issue):W582–W585, July 2004.

[YG05] Yuzhen Ye and Adam Godzik. Multiple flexible structure alignment using partial order graphs. *Bioinformatics*, 21(10):2362–2369, May 2005.

[YJL04] Jieping Ye, Ravi Janardan, and Songtao Liu. Pairwise protein structure alignment based on an orientation-independent backbone representation. *Journal of Bioinformatics and Computational Biology*, 2(4):699–717, December 2004.

[Zem03] Adam Zemla. LGA: a method for finding 3D similarities in protein structures. *Nucleic Acids Research*, 31(13):3370–3374, July 2003.

[ZKS96] Feng Zu-Kang and Manfred J. Sippl. Optimum superimposition of protein structures: ambiguities and implications. *Folding and Design*, 1(2):123–132, April 1996.

[ZS89] Michael Zuker and Ray L. Somorjai. The alignment of protein structures in three dimensions. *Bulletin of Mathematical Biology*, 51(1):55–78, January 1989.

[ZW05] Jianhua Zhu and Zhiping Weng. FAST: A novel protein structure alignment algorithm. *Proteins: Structure, Function, and Bioinformatics*, 58(3):618–627, February 2005.

Glossary

Convergent evolution	Evolutionary process that gradually alters unrelated structures into similar structures due to selective pressures of similar functional requirements. , 113
Divergent evolution	Evolutionary process that gradually alters similar related sequences (of a common ancestor) into different sequences due to adaption of different new selective pressures. , 113
E-value	Expectation value, e.g. in BLAST, used to describe the significance of an alignment by the number of equivalent or better alignments that are expected to occur in a database search by chance. A lower E-value indicates a higher significance of the score. , 100
P-value	Probability of an alignment to occur with at least the given score. The value is calculated by relating the observed alignment score to the expected distribution of high-scoring pairs (HSP) scores that are derived from random sequences of the same length and composition as the query. A lower P-value indicates a higher significance of the score. This measure is used, for instance, in the context of BLAST searches. , 100

Index