RUPEE: A BIG DATA APPROACH TO INDEXING AND SEARCHING PROTEIN STRUCTURES

A Dissertation
IN
Computer Science
and
Mathematics

Presented to the Faculty of the University
of Missouri–Kansas City in partial fulfillment of
the requirements for the degree

DOCTOR OF PHILOSOPHY

by
RONALD AYOUB

MS, Johns Hopkins Whiting School of Engineering, Baltimore, Maryland, USA, 2007

Kansas City, Missouri
2021

RUPEE: A BIG DATA APPROACH TO INDEXING AND SEARCHING PROTEIN STRUCTURES

Ronald Ayoub, Candidate for the Doctor of Philosophy Degree

University of Missouri–Kansas City, 2021

ABSTRACT

Given the close relationship between protein structure and function, protein structure searches have long played an established role in bioinformatics. Despite their maturity, existing protein structure searches either compromise the quality of results to obtain faster response times or suffer from longer response times to provide better quality results. Existing protein structure searches that focus on faster response times often use sequence clustering or depend on other simplifying assumptions not based on structure alone. In the case of sequence clustering, strong structure similarities are often hidden behind cluster representatives. Existing protein structure searches that focus on better quality results often perform full pairwise protein structure alignments with the query structure against every available structure in the searched database, which can take as long as a full day to complete. The poor response times of these protein structure searches prevent the easy and efficient exploration of relationships between protein structures, which is the norm in other areas of inquiry. To address these trade-offs between faster response times and quality results, we have developed RUPEE, a fast and accurate purely geometric protein structure search combining a novel approach to encoding sequences of torsion angles with established techniques from information retrieval and big data. RUPEE can compare the query structure to every available structure in the searched database with fast response times.

To accomplish this, first, we introduce a new polar plot of torsion angles to help identify separable regions of torsion angles and derive a simple encoding of torsion angles based on the identified regions. Then, we introduce a heuristic to encode sequences of torsion angles called *Run Position Encoding* to increase the specificity of our encoding within regular secondary structures, α-helices and β-strands. Once we have a linear encoding of protein structures based on their torsion angles, we use min-hashing and locality sensitive hashing, established techniques from information retrieval and big data, to compare the query structure to every available structure in the searched database with fast response times. Moreover, because RUPEE is a purely geometric protein structure search, it does not depend on protein sequences. RUPEE also does not depend on other simplifying assumptions not based on structure alone. As such, RUPEE can be used effectively to search on protein structures with low sequence and structure similarity to known structures, such as predicted structures that results from protein structure prediction algorithms. Comparing our results to the mTM-align, SSM, CATHEDRAL, and VAST protein structure searches, RUPEE has set a new bar for protein structure searches. RUPEE produces better quality results than the best available protein structure searches and does so with the fastest response times.

APPROVAL PAGE

The faculty listed below, appointed by the Dean of the School of Graduate Studies, have examined a dissertation titled “RUPEE: A Big Data Approach to Indexing and Searching Protein Structures,” presented by Ronald Ayoub, candidate for the Doctor of Philosophy degree, and hereby certify that in their opinion it is worthy of acceptance.

Supervisory Committee

Yugyung Lee, Ph.D., Committee Chair
Department of Computer Science & Electrical Engineering

Majid Bani-Yaghoub, Ph.D., Co-Discipline Advisor
Department of Mathematics & Statistics

Xiaolan Yao, Ph.D.
Department of Molecular Biology & Biochemistry

Noah Rhee, Ph.D.
Department of Mathematics & Statistics

Praveen Rao, Ph.D.
Department of Computer Science & Electrical Engineering

CONTENTS

ILLUSTRATIONS

TABLES

ACKNOWLEDGEMENTS

Thanks to my academic advisor Dr. Yugyung Lee for providing me clear direction on how to go about conducting scientific research and what it takes to be a successful contributor. Thanks to Dr. Deendayal Dinakarpandian for introducing me to all the interesting algorithms and problems in computational biology. Thanks to my wife Krista for helping me to return to school late in my career and complete my education.

CHAPTER 1

INTRODUCTION

Proteins represent the functional end-product within the central dogma of molecular biology [22], that is, DNA codes for RNA through a cellular process called transcription and RNA codes for proteins through a cellular process called translation, as illustrated in Fig. 1. As such, the importance of understanding proteins in the biological sciences cannot be overstated. A greater understanding of proteins comes with a greater understanding of life in general.

In bioinformatics, the two most common representations of proteins are as sequences of amino acids, referred to as protein sequences, and as sets of 3-dimensional atomic coordinates that describe their native 3-dimensional structure, the structure they assume in the environment of the cell. From the advent of sequencing technology, beginning with Sanger sequencing [58] on up to the higher throughput of Next-generation sequencing [59], a large volume of protein sequences have been collected into sequence repositories and organized into sequence clusters and classification hierarchies based on

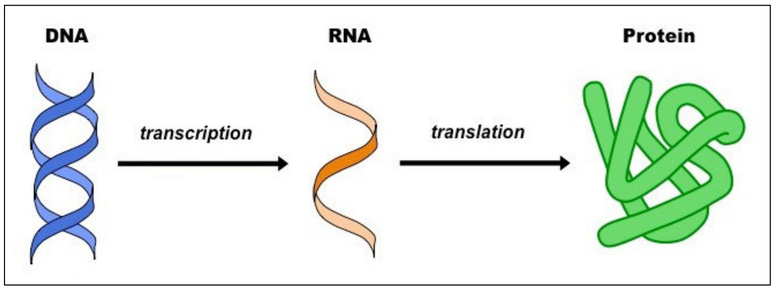


Figure 1: The central dogma of molecular biology (image from BioNinja [20])

sequence similarities. Proteins with similar sequences often have an evolutionary relationship to each other and similar functions. Thus, newly determined sequences can be compared to previously collected sequences, using pairwise sequence alignments, to provide insight into their evolutionary relationships to other proteins in addition to their function.

Similarly, the 3-dimensional structures of a large volume of proteins have been determined using X-ray crystallography and nuclear magnetic resonance (NMR), among other experimental techniques, and stored in the protein data bank [57] (PDB), the global repository for experimentally determined protein structures. Comparing the 3-dimensional structure of a protein to that of other protein structures can provide even greater insight into their evolutionary relationships and function than sequence comparisons alone. (When we use the word "structure" without qualification, we are referring to the 3-dimensional structure of a protein.)

Determining how one protein structure compares to another, using pairwise structure alignments, is mostly a solved problem in structural bioinformatics. However, searching for structurally similar proteins among hundreds of thousands of protein structures has not been adequately addressed. As will be discussed in Section 1.3.3, existing protein structure searches either compromise the quality of results to obtain faster response times or suffer from longer response times to provide better quality results. Ideally, a protein structure search should be fast and ensure quality results. A protein structure search should be fast for the serendipitous exploration of relations between protein structures. Moreover, to ensure quality results, all available structures should be considered for structure similarity. To satisfy these requirements, we developed RUPEE, a fast and accurate purely geometric protein structure search combining techniques from information retrieval and big data with a novel approach to encoding sequences of torsion angles. (RUPEE is an acronym that stands for RUn Position Encoded Encoding of residue descriptors, the meaning of which will be made clear later in Chapters 2 and 3.)

For any given protein domain identifier, whole chain identifier, or uploaded PDB file, RUPEE can search for matches among domains defined in the protein structure databases SCOPe 2.07 [26], CATH v4.2 [49], ECOD develop210 [17], or among whole chains defined in the PDB. RUPEE can search these databases using any identifier. For instance, RUPEE can search SCOPe using a CATH domain identifier. For even more flexibility, RUPEE provides search types for *Contained-In* and *Contains* searches in addition to the *Full-Length* search type. The Contained-In search type searches for structures that are contained in the query structure and the Contains search type searches for structures that the query structure contains. With these containment searches, RUPEE can be used to search for structure motifs within proteins.

For the remainder of this document, in Section 1.1, we describe the problems with existing protein structure searches that it is our objective to address in RUPEE. In Section 1.2, we cover background material on proteins required for understanding the rest of the paper. In Section 1.3. we discuss some of the most popular existing protein structure searches. In Chapters 2 to 4, we provide the methods by which we addressed the problems identified in Section 1.1. In Chapter 5, we provide the results of evaluating RUPEE against the existing protein structure searches identified in Section 1.3.3. In Chapter 6, we provide details on how to use RUPEE at our web site https: //ayoubresearch.com and how to install and use RUPEE locally. Parts of this document have been adapted from our previously published papers [5–7] and have been expanded to provide further insights and details.

1.1 Research Objectives

The two major objectives we set out to accomplish in our research are listed below.

- Develop a fast and accurate purely geometric protein structure search with no dependence on protein sequences, cluster representatives, pre-calculated results, and the exclusive use of secondary structure elements, that performs better than existing structure searches that do.
- Develop a protein structure search that is sensitive enough for searches on predicted structures with low sequence and structure similarity to known structures to assist in the development of protein structure prediction methods.

1.1.1 Purely Geometric Structure Search

To address our first objective, we initially introduced RUPEE with only two search modes of operation, fast and top-aligned [6]. In Section 5.1, we compare RUPEE fast and top-aligned search mode results against the mTM-align structure search [25], the secondary structure matching (SSM) search [44], and the CATHEDRAL structural scan [55] available at the CATH web site. Fast search mode is significantly faster than the other protein structure searches we compared to but at the expense of accuracy. Despite this, we showed that the accuracy of RUPEE in fast search mode is not far below that of some of the best available protein structure searches. On the other hand, both the accuracy and response times of RUPEE in top-aligned search mode are equal to or better than some of the best available protein structure searches.

RUPEE stands out as not just another protein structure search, of which there are many. RUPEE is the first, to our knowledge, purely geometric protein structure search to achieve comparable results to existing protein structure searches. Existing protein structure searches often depend on protein sequence alignments to reduce response times since sequence alignments are much faster than structure alignments. These existing protein structure searches either (1) perform an exhaustive search using protein sequence alignments to obtain a smaller subset of candidate matches on which to perform structure alignments; or (2) compare the query protein sequence to sequence cluster representatives to reduce the number of structures to compare against the query protein structure. Being purely geometric, RUPEE does not depend on protein sequences at all. This allows RUPEE to find structurally similar proteins that do not have similar sequences.

With respect to the use of sequences, while high sequence similarity usually indicates high structure similarity [18], high structure similarity has been observed even for structures with low sequence similarity since structure is more conserved in evolution than sequence [35]. While sequence alignments can be used to reduce the number of structures that have to be examined, RUPEE can examine every structure individually during a search and does so in a reasonable amount of time. We have found that protein structure searches that rely on sequence alignments for filtering miss structurally similar proteins because they do not consider every structure individually.

Also, existing protein structure searches often will use other techniques that can negatively impact response time and/or quality of results, such as dependence on (1) cluster representatives, (2) pre-calculated results, and (3) exclusive use of secondary structure elements. (These techniques are described in more detail in Section 1.3.3.) While using cluster representative can make the protein structure search faster, any dependence on clustering, whether clustering protein sequences or protein structures, can limit the sensitivity of a protein structure search. Also, although using pre-calculated results can reduce response times if a query protein structure has a known structure id, the query protein structure often does not have such an id, which can lengthen response times significantly. In addition, some protein structure searches depend on the exclusive use of the orientation and connectivity of secondary structure elements, which fails to capture the complexity of loops.

RUPEE does not use cluster representatives or pre-calculated results and thus avoids the problems identified above with respect to these techniques. While RUPEE uses secondary structure elements, it does not do so exclusively and thus accounts for the structure of loops. Thus, unlike existing protein structure searches, RUPEE does not compromise the quality of results to obtain faster response times or suffer from longer response times to provide better quality results.

1.1.2 Predicted Structure Search

Determining the structure of a protein is an important step toward understanding its function. There are approximately 150,000 solved protein structures currently stored in the

PDB [15], the global repository for experimentally determined protein structures. On the other hand, UniProt [19], the universal protein knowledgebase, currently provides over 60 million protein sequences. From this, it is apparent that protein structure determination is moving at a slower pace than protein sequencing and may be serving as a bottleneck in a variety of research efforts from protein design to drug discovery. Being able to predict a protein structure from its amino acid sequence would address this problem. However, protein structure prediction remains a central unsolved problem in molecular biology [1].

CASP is a biannual blind competition for protein structure prediction that began in 1994 [47]. Progress had been slow until the success of coevolutionary methods in contact prediction demonstrated in CASP11 [46]. The recent success of AlphaFold at CASP13 [1] using deep learning combined with coevolutionary methods has renewed interest in the problem. While AlphaFold's performance was remarkable, it depends on the availability of sufficiently large multiple sequence alignments (MSA) for its use of coevolutionary methods, which may not be available for all target structures. Even when a large enough MSA is available, AlphaFold as well as traditional physics-based approaches to protein structure prediction, such as Rosetta [56], have not reached the desired level of accuracy [1].

As stated above, we initially provided RUPEE with two search modes, fast and top-aligned, optimizing for response times and quality of results, respectively. Following the initial release of RUPEE [6], we observed that RUPEE had been used to upload protein structures that were the output of a protein structure prediction method to identify the most similar known structures to the predicted structures. For the most part, these uploaded protein structures had low sequence and structure similarity to known structures in the PDB. This low similarity to known structures is to be expected given the limited accuracy of current protein structure prediction methods.

When searching for structures with low sequence and structure similarity to known structures, the importance of small differences in structure similarity becomes proportionally larger since they comprise a larger percentage of the overall similarity. While the fast and top-aligned search modes have no dependence on protein sequences, they often lack sufficient sensitivity to find the most similar matches for a structure with low structure similarity to known structures. This lack of sensitivity for RUPEE fast and top-aligned search modes with respect to low similarity searches are due to the lower accuracy of their initial structure similarity estimates that are used to filter candidate matches. Recognizing the need for a fast and accurate purely geometric structure search with more sensitivity then fast and top-aligned search modes and still having no dependence on protein sequences, we added a search mode to RUPEE with increased sensitivity called all-aligned search mode.

In Section 5.2, again we compare the results of RUPEE against mTM-align [25], SSM [44] and CATHEDRAL [55], but this time we include all-aligned search mode. Additionally, this time we also compare to the VAST protein structure search [28] available at the U.S. National Center for Biotechnology Information (NCBI) web site. For these comparisons, we use a benchmark derived from protein structure predictions of free-modelling targets in CASP13 available at the CASP web site [43]. Here, we show that RUPEE, in all-aligned search mode, is better at identifying similar structures than the protein structure searches we compare to using a benchmark drawn from the output of protein structure prediction methods.

1.2 Background

In this section, we discuss the fundamentals of protein science that are necessary to understand fully the rest of this document such as what exactly are proteins, the levels of protein structure, protein folding, and some examples of commonly referenced proteins. If you are already familiar with this material, you may skip this section.

1.2.1 Proteins

Proteins are a broad class of macromolecules that are involved in numerous cellular functions. Proteins are often referred to as the machinery of the cell because their biological functions are coordinated along well-defined pathways like an assembly line within a factory. The protein machinery is instrumental in such functions as DNA replication and repair, transcription from DNA to RNA and translation from RNA to proteins, and cell division and death. In fact, the vast majority of enzymes are proteins so proteins are involved in just about every metabolic pathway.

Proteins are polymers consisting of chains of amino acid monomers. As shown in Fig. 2, all amino acids have a common structure consisting of an amino group, a carboxyl group, and a central α-carbon. The different types of amino acids are distinguished by their R-group. There are 20 types of amino acids that make up proteins, which can be divided into hydrophobic and hydrophilic amino acids along with some special cases. As a general rule, hydrophobic amino acids pack together in the core of a protein to avoid

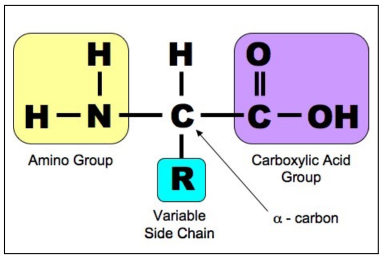


Figure 2: Amino Acid Molecule (image from BioNinja [20])

contact with the watery environment of the cell, and the hydrophilic amino acids are scattered across the surface of a protein and are often involved in interactions with other proteins and macromolecules.

1.2.2 Levels of Protein Structure

Fig. 3 illustrates the 3 levels of protein structure — primary, secondary, and tertiary. The primary structure is the linear sequence of amino acids that comprise a protein chain, that is, what we have been calling the protein sequence. When joined together into a protein chain by way of peptide bonds, the amino acids are more accurately referred to as amino acid residues, or more simply, residues. The chain of residues, excluding the R-groups, is often referred to as the backbone or main chain and the R-groups are often referred to as the side chains. The primary structure of a protein is usually represented as a sequence of letters indicating the types of amino acids that make up the backbone.

Within the environment of the cell, once the ribosomal machinery translates RNA to a protein, regular patterns of hydrogen bonds along the backbone produce two common secondary structure motifs, α-helices and β-strands. Regular hydrogen bonding patterns

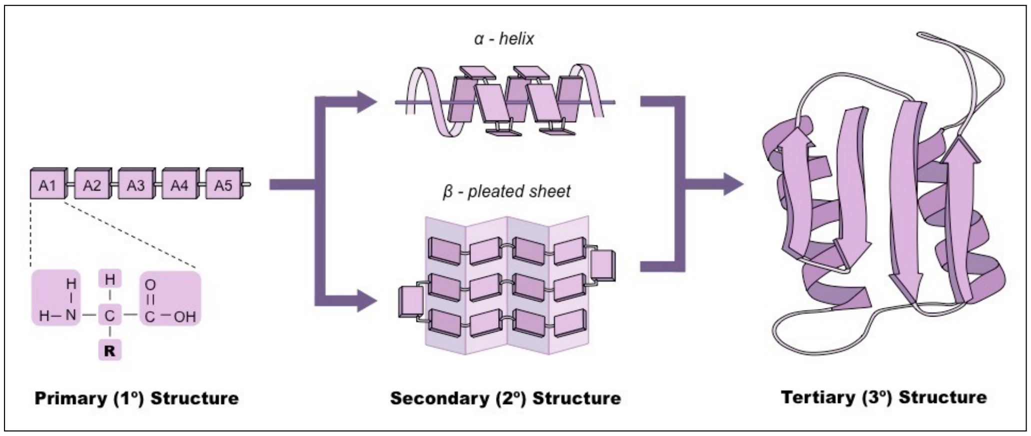


Figure 3: The 3 levels of protein structure (image from BioNinja [20])

between amino acids located 3, 4, and 5 residues apart form α-helices and regular hydrogen bonding patterns between residues farther apart form β-strands. The hydrogen bonds that are responsible for secondary structure are exclusively along the main chain of amino acids. Fig. 4 shows a more detailed example of the two secondary structure motifs, α-helices and β-strands, along with lines to indicate the hydrogen bonds.

Tertiary structure is the third level of protein structure and indicates the 3-dimensional structure of a protein. Whereas secondary structure results from main chain hydrogen bonding, tertiary structure is a result of hydrogen bonding between side chains in addition to hydrophobic packing of residues and disulfide bridges between residues containing sulfur. Furthermore, unlike secondary structure, in most cases tertiary structure does not consist of regular patterns. In fact, tertiary structure is famously complex and irregular. When John Kendrew and his coworkers, pioneers in X-ray crystallography, uncovered the structure of the myoglobin protein in 1962 [24], one is quoted as saying of the structure, “It’s a horrible object, but beautiful work.” The complexity of protein tertiary structure

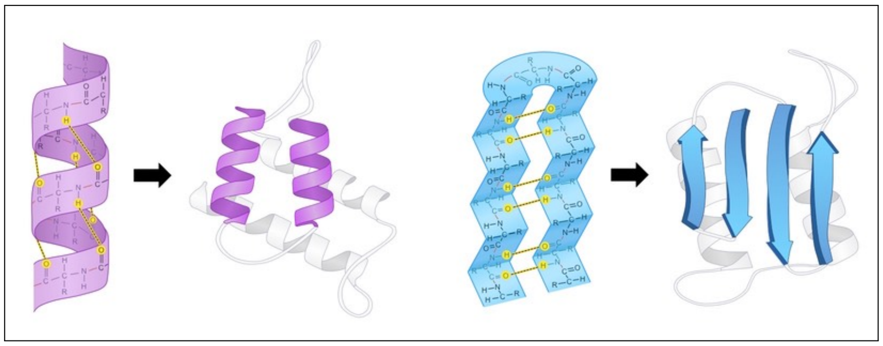


Figure 4: Protein secondary structure hydrogen bonding patterns (image from BioNinja [20])

is responsible for the many challenges encountered when indexing, searching, and predicting protein structures. At the same time, this same complexity is also responsible for attracting the scientific curiosity of researchers.

Not shown in Fig. 3 is protein quaternary structure. Quaternary structure is the structure that results from multiple protein chains interacting to form a multi-chain protein, also called an oligomer. Proteins that consist of multiple chains are referred to as heterodimers, homodimers, trimers, etc., which indicate their oligomeric structure. For instance, a heterodimer is composed of two distinct protein chains whereas a homodimer is composed of two identical protein chains.

1.2.3 Protein Folding

Within the environment of the cell, a linear protein chain assumes its native 3-dimensional tertiary structure by reaching a thermodynamic minimum by way of hydrogen bond formation, packing of hydrophobic residues, and other side chain interactions. The dynamic process of a protein assuming its native structure is known as protein folding.

Fig. 5 is a common way to visualize the potential energy landscape of protein folding, referred to as the folding funnel. The vertical axis is the potential energy of the protein structure at a given stage of the folding process. For instance, the potential energy of the simple linear structure of protein immediately after being translated by the ribosomal machinery will be somewhere toward the top of the funnel. As main chain hydrogen bonding occurs, the potential energy will decrease and perhaps reach one of the potential energy local minima shown about halfway down the folding funnel. Then perhaps, enzymatic catalysis and the kinetics of the environment within the cell allow the protein to escape the local minimum and eventually reach its native structure at the potential energy minimum at the bottom of the folding funnel.

It is important to note that the folding funnel shown in Fig. 5 is only an aid to visualize the protein folding process. At the bottom of Fig. 5 there are only 2 variables that determine the energy minimum while in a real scenario there are thousands of variables. In Chapter 2, we will discuss the most important of these variables, torsion angles.

The question of whether a given protein can fold into more than one type of structure has been settled by Anfinsen's principle [3]. Anfinsen's principle states that a protein will always fold into its unique native structure, which is the potential energy minimum, and which is solely determined by its sequence of amino acids. On the other hand, multiple protein sequences can fold into the same structure since structure is more conserved in evolution than sequence [35]. This is one way, among many, that nature preserves the function of a protein in the case of genetic mutations over time.

Although the structure of a protein is determined by its amino acid sequence, as

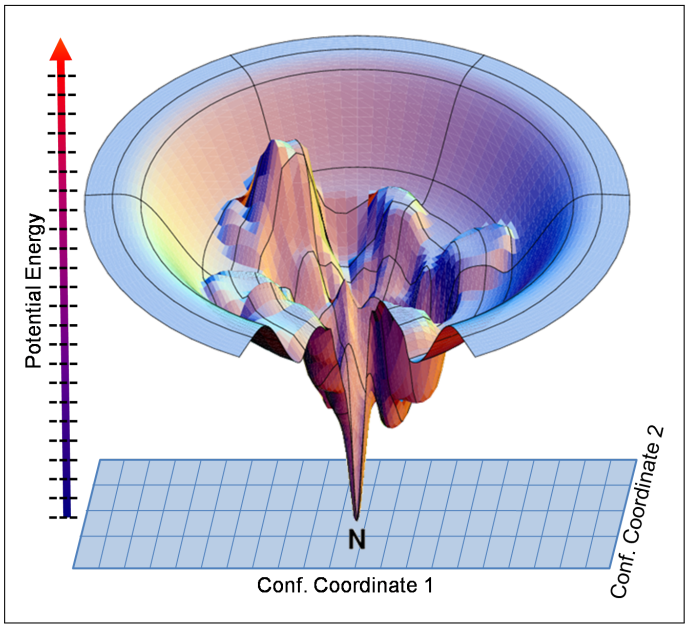
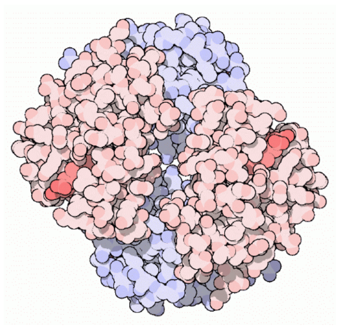
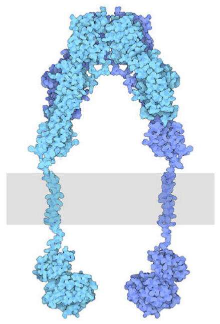


Figure 5: Protein folding funnel

of yet, there is no way to determine the structure of a protein from its sequence without determining its structure through experimental techniques such as X-ray crystallography and NMR. However, Anfinsen's principle does raise the possibility, at least in theory, of predicting the structure of a protein from its sequence of amino acids.

Now that we have discussed protein folding, we can introduce another structure definition beside the levels of protein structure. A protein domain is a independently folding unit of a protein chain that often has a well-defined function. A protein domain can either be a full protein chain or only part of a protein chain. A chain often contains multiple domains and a domain can occur in a variety of different proteins. A protein domain can be thought of as a modular structural unit that serves an identical purpose in a variety of contexts, that is, as part of different types of proteins. In this document, we


(a) Hemoglobin


(b) Insulin Receptor

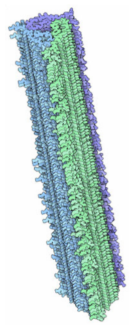

(c) Amyloid Fiber

Figure 6: Example Proteins (images from PDB-101 [51])

will often refer to protein domains since protein structures are more often classified at the domain level than at the whole chain level in protein structure databases. The exception to this is the PDB, which stores all chains for all protein structures in fulfillment of its role as the universal protein structure repository.

1.2.4 Some Example Proteins

Fig. 6 shows 3 important examples of proteins, (a) hemoglobin, (b) an insulin receptor, and (c) an amyloid fiber. Hemoglobin is a tetramer consisting of 2 pairs of identical protein chains, one pair is shown in light red and the other in light blue. The dark red portions are heme molecules bound by each protein chain for a total of 4 heme molecules per hemoglobin. The heme molecule binds iron, which in turn binds oxygen for transport throughout the body via the blood.

An insulin receptor is a multi-chain transmembrane protein. Transmembrane proteins are a large class of proteins that crosses the lipid bilayer of the cell membrane.

Insulin receptors bind the hormone insulin, which triggers a structure change in the cellular portion of the insulin receptor protein signaling the cell to uptake sugar from the bloodstream for glycolysis. Insulin is responsible for regulating the amount of sugar in the blood. Diabetes is a disease that results due to a deficiency of insulin, resulting in high blood sugar levels.

An amyloid fiber is an aggregation of misfolded amyloid-β proteins that result from incorrectly cleaved fragments of a transmembrane protein known as the amyloid precursor protein (APP). These amyloid fibers combine to form plaques, which is characteristic of Alzheimer's disease as will be discussed further in Section 6.3.1.

1.3 Related Work

In this section, we discuss some of the related work that RUPEE uses, extends, or competes with, including protein sequence alignments, protein structure alignments, and protein structure searches.

1.3.1 Sequence Alignments

Biological sequences are common in bioinformatics. Broadly, there are two types of biological sequences, sequences of nucleotides called nucleic acids such as DNA and RNA, and sequences of amino acids, often called protein sequences. For the majority of this document, we are primarily concerned with protein sequences that define proteins.

Sequence alignment algorithms are the most important algorithms in bioinformatics and among the earliest developed. For the most part, sequence alignments are used to determine homology for evolutionarily related proteins. Homologous relationships come in two varieties, paralogs and orthologs. Paralogous proteins are related proteins from the same species where one protein is likely a horizontal gene copy from the other that then evolved separately following the copy event. Orthologous proteins are related proteins existing in two distinct species where one species evolved from the other and maintained the same coding gene in evolution with some modifications.

The most common sequence alignment algorithms use the dynamic programming [8] technique from computer science, a bottom-up approach to solving optimization problems. Dynamic programming can be applied when a problem contains frequently occurring subproblems. Whereas a top-down approach may end up solving the same subproblems multiple times, the bottom-up approach solves each subproblem only once and references the solved subproblems to solve larger subproblems in a tableau fashion. One of the most frequently cited dynamic programming algorithms is the longest common subsequence [34] algorithm, which serves as the basis for the slightly more complex biological sequence alignment algorithms.

The first sequence alignment algorithm developed was the Needleman-Wunsch [48] (NW) global sequence alignment algorithm. Global sequence alignment seeks to align two sequences across their full-length. Fig. 7 shows the NW dynamic programming matrix for aligning the two nucleotide sequences GATTACA and GCATGCU. The numbers in each cell are calculated in order starting from the top left and proceeding to the bottom right going row by row. Aligning matched elements is awarded 1 point. Aligning mismatched elements is penalized 1 point. Similarly, aligning an element to a gap is penalized 1 point. The subproblems consist of aligning a prefix from GATTACA with a

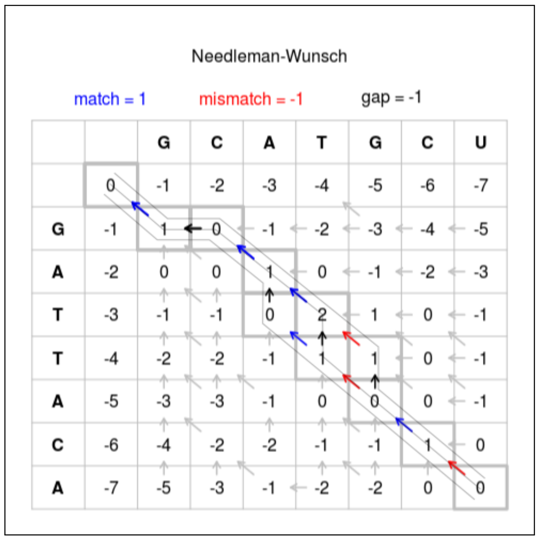


Figure 7: The Needleman-Wunsch dynamic programming matrix for aligning GATTACA with GCATGCU (image from Wikipedia [65])

prefix from GCATGCU. For instance, aligning the sequence GATTA with the sequence GCA is a subproblem.

In Fig. 7, the arrows leading from the bottom right to the top left are referred to as the backward trace. Once all the cells are filled in as described in the previous paragraph, the backward trace is used to find the optimal solution. The blue arrows indicate matches, the red arrows indicate mismatches, and the black arrows indicate the introduction of gaps into the alignment. There may be multiple optimal alignments when aligning two sequences. In this case, the alignment of GCATG-CU with G-ATTACA and the alignment of GCA-TGCU with G-ATTACA are both optimal. The dashes represent gaps in the alignment.

Following the NW global sequence alignment algorithm, the Smith-Waterman [61] (SW) local sequence alignment algorithm was introduced. The SW algorithm modifies the NW algorithm by eliminating all starting and ending gap penalties and finds the optimal score anywhere within the matrix and traces back from that point to the non-negative starting point. There are some subtleties to the SW algorithm that are out of scope for this document.

A hybrid between the NW and the SW algorithms is referred to as semi-global NW sequence alignment [13]. Whereas a local sequence alignment finds where a part of one sequence aligns well with a part of another sequence, a semi-global sequence alignment finds where an entire sequence aligns well with only a part of another sequence. In Chapter 4 we describe how we use semi-global NW sequence alignments for containment searches.

Fig. 8 shows the difference between an NW global sequence alignment and a semi-global NW sequence alignment. Here, we are aligning AGCTTC to AGTCAGT-GCGTGC, with the shorter sequence along the first column and the longer sequence along the first row. The only difference between the global alignment and the semi-global alignment is that the opening gap along the first row and the end gap along the last row are not penalized. This allows for the better alignment to be found for the semi-global sequence alignment shown on the right of Fig. 8 as opposed to the global sequence alignment shown on the left of Fig. 8.

1.3.2 Protein Structure Alignments

Protein structure alignment, the most accurate method for comparing protein structures, involves finding a set of spatial rotations and translations for two protein structures that

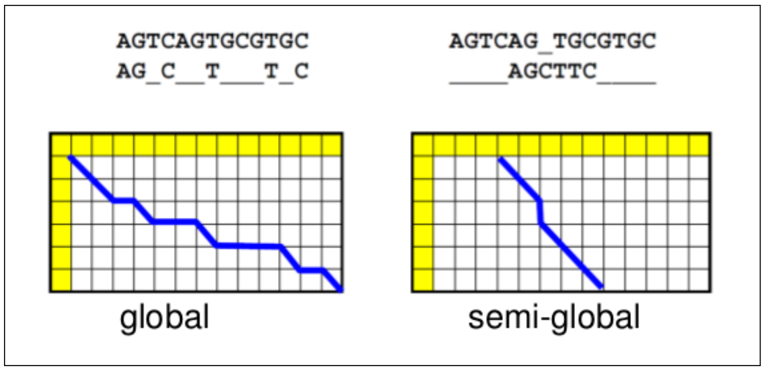


Figure 8: Global vs. semi-global sequence alignments

minimizes the sum of the distances between a set of aligned residues. Not all residues are necessarily aligned in a structure alignment. Traditionally, the root mean squared deviation (RMSD) between α-carbons of aligned residues is minimized. However, the RMSD score does not factor in inter-residue distances, alignment length, or the alignment coverage defined as the percentage of aligned residues. RMSD scores also have some dependence on the length of the aligned proteins. To address these concerns, another score, the TM-score [68], is frequently used in the scoring of protein structure alignments. The TM-score takes all residues into account and normalizes for both coverage and length of the aligned proteins.

The RMSD of two aligned structure is defined as

$$RMSD(a,b) = \sqrt{\frac{1}{n}\sum_{i}^{n} \|a_i - b_i\|^2}, \tag{1.1}$$

where a and b are vectors of aligned residues and n is the number of aligned residues. A perfect alignment will have an RMSD equal to 0.0.

The TM-score [68] is defined as

$$TMSCORE(a,b) = \frac{1}{l}\sum_{i}^{n} \frac{1}{1+\left(\frac{a_i-b_i}{d_0}\right)^2}, \tag{1.2}$$

where again a and b are vectors of aligned residues and n is the number of aligned residues. l is the normalization length, which can be the length of the query protein, the length of the target protein or the average length of both proteins being aligned. Finally, d_0 is a constant factor that normalizes the distance between aligned residues so that the average TM-score is not dependent on the size of the protein structures being aligned. In contrast to the RMSD, the TM-score is always a value between 0.0 and 1.0 and a perfect alignment will have a TM-score equal to 1.0.

Structure alignments often favor accuracy over speed because the typical use case of aligning one protein structure to another does not impose tight response time requirements. On the other hand, a protein structure search can involve thousands of comparisons and accuracy is often balanced against speed. In this case, structure alignments are still useful for evaluating the results of a search, and this is the approach we take.

For structure alignments, Combinatorial Extensions [60] (CE) and FATCAT [66] are among the most popular tools, representing rigid and flexible protein structure alignments, respectively. CE performs a rigid alignment to minimize RMSD and FATCAT allows for a constrained number of twists in the protein chain to find a more flexible alignment before minimizing RMSD.

Besides CE and FATCAT, TM-align [69] and DALI [33] are structure alignment tools also in wide use, both offering their own distinct approaches to structure alignment.

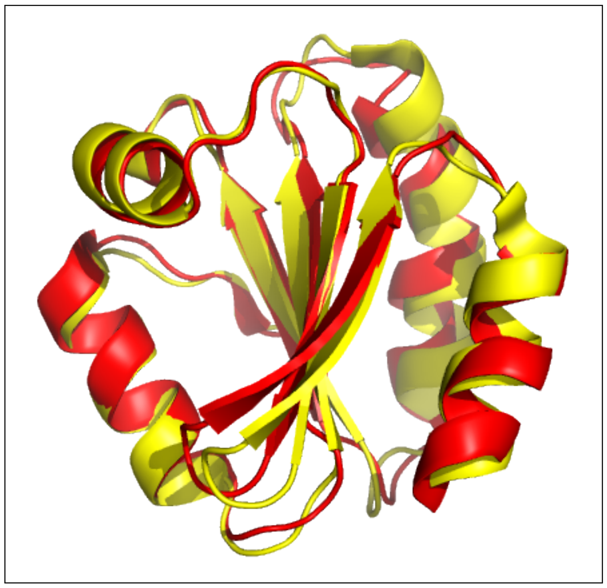


Figure 9: Structure alignment of thioredexoin proteins from the human and fly specifies. The human protein is red and the fly protein is yellow.

TM-align uses a rotation matrix designed to maximize the TM-score rather than minimizing the RMSD along with dynamic programming to find the best full-length alignment. DALI compares intra-residue distance matrices between two proteins to find a consistent set of matched submatrices that is used to align the proteins.

Of the structure alignment tools, CE, FATCAT, TM-align, and DALI, we found TM-align to be the fastest while also providing high-quality structure alignments. It is for this reason that we use TM-align for RUPEE when performing structure alignments.

As an example, Fig. 9 shows the TM-align structure alignment between homologous thioredoxin proteins from the human and fly species.

1.3.3 Protein Structure Searches

Existing protein structure searches often use a variety of techniques that can negatively impact response times and/or quality of results, such as dependence on (1) protein sequences, (2) cluster representatives, (3) pre-calculated results, and (4) exclusive use of secondary structure elements. Whereas structure alignments only depend on sequences of the 3-dimensional coordinates of the α-carbon atoms of each residue, protein structure searches often introduce a further dependence on protein sequences as mentioned in Section 1.1.1. To repeat, these existing protein structure searches that depend on protein sequences either (1) perform an exhaustive search using protein sequence alignments to obtain a smaller subset of candidate matches on which to perform structure alignments; or (2) compare the query protein sequence to sequence cluster representatives to reduce the number of structures to compare against the query protein structure.

Some existing protein structure searches also depend on cluster representatives. Any dependence on clustering, whether clustering protein sequences or protein structures, can limit the sensitivity of a protein structure search. A protein structure search that depends on clustering will compare the query structure to cluster representatives for each cluster to exclude clusters on which to compare the query structure. We have found that this approach often misses good matches hidden behind cluster representatives. In the absence of reliance on protein sequences or clustering, and without sacrificing the quality of results, response times for existing protein structure searches suffer greatly, often taking upwards of an hour for searches to complete.

Another weakness of some existing protein structure searches is that they often use pre-calculated results to reduce response times. These protein structure searches pre-calculate the results for protein structure alignments for known protein structures. In this case, if a known protein structure is used as the query structure, quality results are returned and response times are fast. However, the most common and useful scenario is to upload a protein structure as the query structure that does not have a known structure id, in which case, to obtain quality results, the query can take an hour or more.

Another weakness of some existing protein structure searches is that they depend on the exclusive use of the orientation and connectivity of secondary structure elements, which fails to capture the complexity of loops.

The protein structure searches we compare to represent a good mix of existing approaches. For structure searches that depend on protein sequence clusters, mTM-align [25] is among the best available and is capable of handling uploaded structures with the same response-times as for searching on a structure id. SSM [44] is a good example of a fast graph-theoretic structure search with no dependence on protein sequences, clustering, or pre-calculated results. However, the speed of SSM is at the expense of sensitivity since it depends on the spatial orientation and connectivity of secondary structure elements, which fails to capture the complexity of loops. The CATHEDRAL [55] structural scan, available at the CATH website [49], also uses a fast graph-theoretic approach that is more accurate than SSM, but still lacks sufficient sensitivity to identify the most similar structure matches for low similarity searches. The lack of sensitivity for CATHEDRAL, is due, at least in part, to structural clustering, because it only returns results for representatives of structural clusters at 35% similarity [49], referred to as s35 representatives. Moreover, CATHEDRAL can take upwards of 10 minutes to produce results against CATH s35 representatives. The VAST protein structure search [28] is similar to SSM in that it depends on the spatial orientation and connectivity of secondary structure elements. Although VAST is much slower than SSM for uploaded structures, because of a dependence on pre-calculated results to speed searches using a known structure id, its searched database is more recent than that of SSM. If given a known structure id, VAST can return structural neighbors in seconds using pre-calculated results. However, if uploading a PDB file where pre-calculated results are not used, response times for VAST can exceed 30 minutes.

While we would have liked to compare our results to DALI [32], another popular purely geometric protein structure search based on inter-residue distances, DALI was unable to return any results for a majority of the benchmark of predicted structures as discussed in Section 5.2. DALI also provides a heuristic structure search [32] of whole chains found in the PDB in addition to a tool for structure alignments using distance matrices as discussed above. In the case of searching, DALI first identifies matched PDB-90 cluster representatives and then walks a pre-calculated graph of structural similarities to identify further matches in the PDB to gradually build up the set of structures similar to the query structure. DALI is slow in comparison to SSM, and mTM-align has shown better quality results than DALI [25].

Given the above, there remains a need for a purely geometric protein structure search that is fast, scalable, and sensitive enough for searches on structures with low sequence and structure similarity to known structures. Being purely geometric, RUPEE does not depend on protein sequences, which allows RUPEE to find structurally similar proteins that do not have similar sequences. RUPEE also does not use clustering or pre-calculated results, so it avoids the problems described above associated with the use of those techniques. While RUPEE uses secondary structure elements, it does not do so exclusively and thus accounts for the structure of loops. RUPEE also is fast, which will allow for the serendipitous exploration of relations between protein structures performed in the trenches. RUPEE also is scalable, which is important given a 10% yearly growth rate of solved structures deposited in the PDB [57]. As such, unlike existing protein structure searches, RUPEE does not compromise the quality of results to obtain faster response times or suffer from longer response times to provide better quality results.

CHAPTER 2

ENCODING TORSION ANGLES

Within a protein structure, all bond angles and bond lengths are roughly constant. Therefore, the only degrees of freedom defining the backbone geometry of a protein structure are the torsion angles joining the amino acids together. Hence, our first step toward a linear encoding of protein structures is to identify separable regions of permissible torsion angles to assign labels to each position in a protein chain.

2.1 Torsion Angles

First, from geometry, a dihedral angle is defined as the angle between two planes within a third plane that cuts the intersection of the two planes by right angles. Fig. 10 shows a dihedral angle between two planes α and β cut by a third plane.

In chemistry, dihedral angles created by planes defined by the positions of atoms are more often called torsion angles. From geometry, only 3 atoms are required to define a

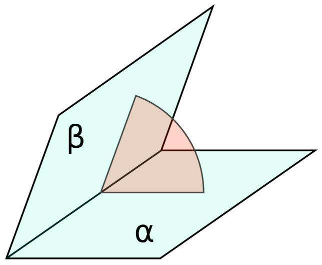


Figure 10: A dihedral angle between two planes α and β cut by a third plane colored orange (image from Wikipedia [64])

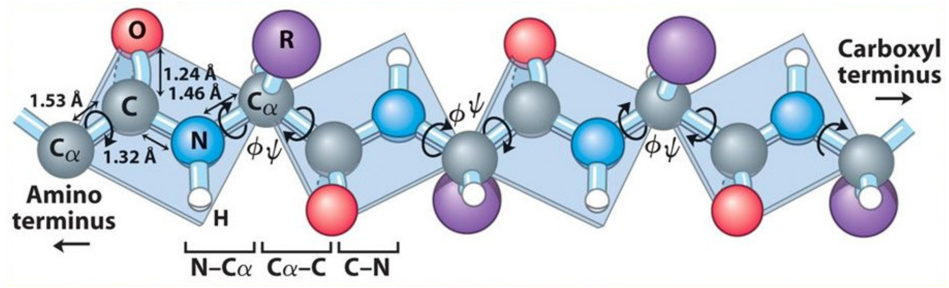


Figure 11: ϕ and ψ torsion angles along a protein backbone (image from *Lehninger principles of biochemistry* [21])

plane. The ϕ torsion angle is defined as the dihedral angle between the plane defined by C, N, and C_α atoms in succession and the plane defined by N, C_α, and C atoms in succession with the N and α-carbon atoms shared by both planes. The ψ torsion angle is defined as the dihedral angle between the plane defined by N, C_α, and C atoms in succession and the plane defined by C_α, C, and N atoms in succession with the C_α and C atoms shared by both planes.

Fig. 11 shows the ϕ and ψ torsion angles along a protein chain. As previously mentioned, since all other bond lengths and bond angles are roughly constant, the ϕ and ψ angles are the only degrees of freedom defining the backbone geometry of a protein.

2.2 Plotting Torsion Angles

To identify regions of torsion angles, we randomly sampled 10,000 residues from high-resolution CATH s35 representatives to account for precision and redundancy, respectively. A Ramachandran [54] plot of the sampled torsion angles is shown in Fig. 12. The Ramachandran has been the default plot to represent torsion angles along a protein backbone since its introduction.

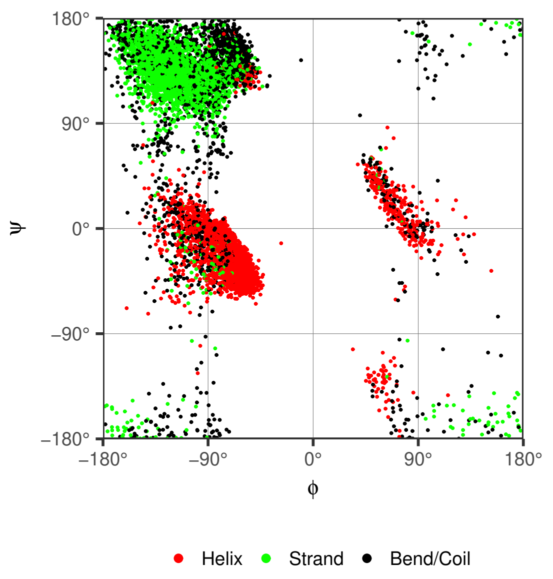


Figure 12: Ramachandran plot of randomly sampled torsion angles

Despite their utility and familiarity, Ramachandran plots represent angular data using a square plot better suited for scalar data. This leads to the unwieldy arrangement where the top part of the plot is continuous with the bottom and the left is continuous with the right. As can be seen in Fig. 12, a single cluster of residues, consisting primarily of β-strands, appears at all 4 corners of the Ramachandran plot.

The continuity problem with Ramachandran plots was partially addressed in [31] using *wrapped* and *mirrored* plots. Both wrapped and mirrored plots take advantage of two sparsely populated areas of the Ramachandran plot at $\phi = 0°$ and $\psi = -120°$. However, with larger samples of torsion angles, the area at $\psi = -120°$ becomes less sparse.

Fig. 13 shows two wrapped Ramachandran plots, where the torsion angles have been translated to place the sparsely populated regions at the boundaries of the plot to avoid cutting off continuous regions of torsion angles. $\phi < 0°$ is translated to $\phi + 360°$ and $\psi < -120°$ is translated to $\psi + 360°$. While the break at $\phi = 0°$ is indeed sparse, the break at $\psi = -120°$ does appear to create a break in a continuous cloud of torsion angles. The break at $\psi = -120°$ is even more questionable for the glycine (GLY) residues shown in Fig. 13(B).

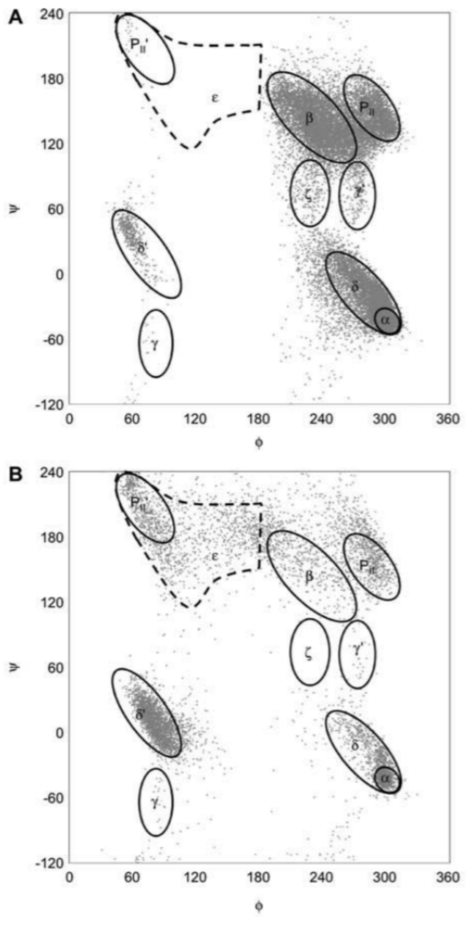


Figure 13: Wrapped Ramachandran plots [31] with $\phi < 0°$ translated to $\phi + 360°$ and $\psi < -120°$ translated to $\psi + 360°$. A) All residues except GLY and PRO residues. B) GLY residues only.

Fig. 14 shows a mirrored Ramachandran plot, where torsion angles with $\phi <$

$0°$ have been translated to $\phi + 360°$. To take advantage of the sparse region at $\psi = -120°$, instead of translating points as with the wrapped plots, values along the ψ-axis are repeated to highlight mirrored torsion angle conformations. To avoid plotting the same torsion angles more than once, shaded regions are introduced in Fig. 14.

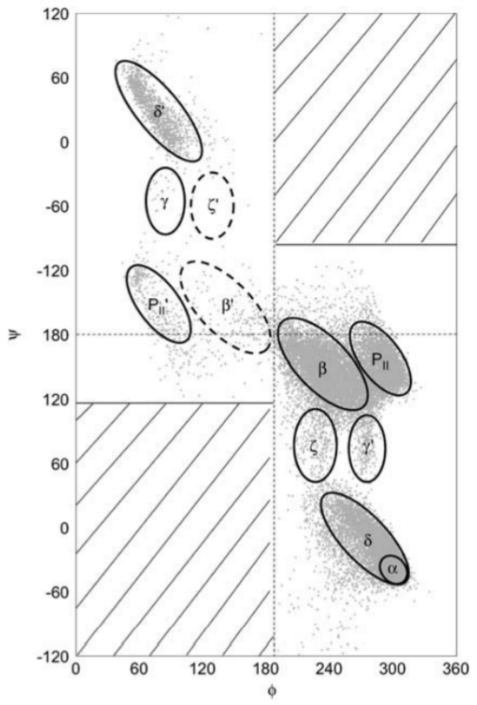


Figure 14: Mirrored Ramachandran plot [31] with $\phi < 0°$ translated to $\phi + 360°$. The axis at $\psi = 180°$ is better understood as $\psi = \pm 180°$.

The labeled regions in both Fig. 13 and Fig. 14 were used to identify identical regions when comparing regions between the two plots. Given that the $\psi = -120°$ regions of both types of plots are not entirely empty, using $\psi = -120°$ as a break is not as effective as may have been hoped for. Moreover, translating points outside of the traditional range of $(-180°, 180°)$ may lead to confusion. On the other hand, while the mirrored plot may be of use for highlighting mirrored conformations, it is unwieldy and

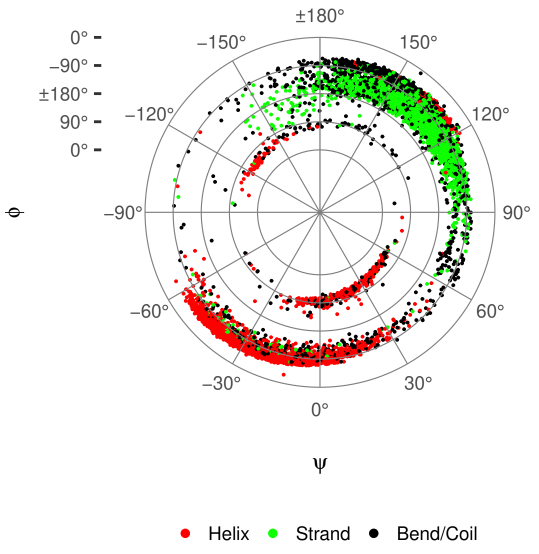


Figure 15: Polar plot of randomly sampled torsion angles

not generally useful.

The use of a polar plot resolves the continuity problem with the traditional Ramachandran plots more elegantly than wrapped and mirrored plots by only requiring one break in continuity at $\phi = 0°$. In Fig. 15, we show the same torsion angles appearing in the Ramachandran plot of Fig. 12 using a polar plot. In this plot, ϕ corresponds to the radius r and ψ corresponds to the angle θ in traditional polar plots. Notice the residues appearing at the 4 corners of the Ramachandran plot now appear in one continuous region of the polar plot centered at $\phi = \pm 180°$ and $\psi = \pm 180°$.

2.3 Torsion Angle Regions

While the plot of Fig. 15 does solve the continuity problem with the Ramachandran plot, the torsion angles themselves are not clearly separable just yet. By plotting the torsion angles for helices, strands, and coil individually, separable regions become clear. Therefore, we divide six of the eight DSSP secondary structure assignment codes defined in [40] into three groups: helices (‘G’,‘H’,‘I’), strands (‘E’), and bends and coil (‘S’,‘C’). For each of the three groups, we plotted the torsion angles and identified regions into which they clustered as shown in Fig. 16. The regions for helices are assigned descriptors 1 to 4, for strands 5 to 7, and for bends and coil 8 to 10. The other two DSSP secondary structure assignments codes for turns (‘T’) and bridges (‘B’), are assigned descriptors 11 and 12, respectively. For each polar plot, there are well-defined continuous regions of torsion angles that remain continuous in the plots. The only exception is found in the bends and coil plot at $\psi = 60°$ between $\phi = -180°$ and $\phi = 0°$.

The descriptors shown in Fig. 16 are the basis for our linear encoding of protein structures as will be described in Chapter 3.

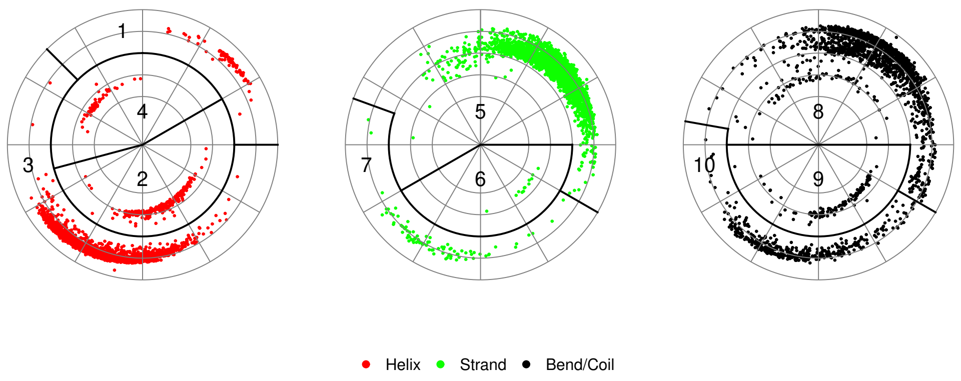


Figure 16: Polar plots of randomly sampled torsion angles with designated descriptors for region and DSSP code combinations

CHAPTER 3

INDEXING PROTEIN STRUCTURES

Now that we have an encoding for torsion angles as shown in Chapter 2, we first demonstrate a simple linear encoding of a protein structure. Then, we transform the linear encoding, through several intermediate steps, into a bag of integers and then into a fixed-length signature of min-hashes [11]. For indexing, in this chapter we work forward, ending with what is stored in the index. In the next chapter, for searching, we work backward through the index from the less accurate filtering methods of min-hashing and LSH to the more accurate final protein structure alignments.

3.1 Linear Encoding of Protein Structures

As an example of our linear encoding, Fig. 17 shows a typical β-turn-β motif annotated with the residue descriptors for the sequence shown below. The underlined elements in Equation (3.1) correspond to the underlined elements in Equation (3.2), Equation (3.4), and Equation (3.7) below to help illustrate the subsequent transformations from descriptors to shingles and finally to min-hashes.

$$[\,5,5,5,5,5,5,\underline{7,5,11},11,5,5,5,5,5,5\,] \qquad (3.1)$$

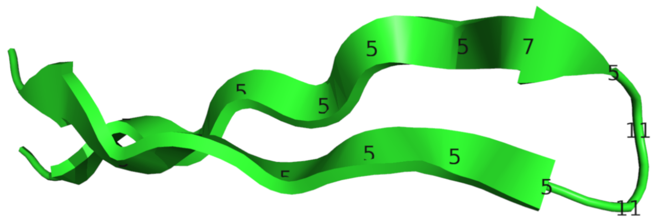


Figure 17: β-turn-β motif from CATH domain 1nycA00

3.2 Bag Representation of Protein Structures

Once a linear encoding for a protein structure is obtained, it needs to be further transformed into a representation suitable for fast and scalable similarity comparisons to other structures. The processing of text documents within Information Retrieval (IR) has long been used to satisfy these requirements using bag representations. There are two broad categories of representations for documents, syntactic and semantic, and much of the research applying IR to protein structure search has focused on the latter [4, 14, 67].

The combination of term frequency and inverse document frequency (TF/IDF) statistics is the most notable example of the semantic approach in IR to document similarity. The TF/IDF statistics are based on the intuition that if a term frequently occurs in a document but not frequently in all other documents under consideration, then it is likely a meaningful term useful for describing the contents of a document that it frequently occurs in. Commonly, in the TF/IDF approach, a vector of TF/IDF weights is used to represent a document and cosine similarity is used for comparing documents.

The problem with applying the semantic TF/IDF approach to protein structures is that the highly variable loop regions of protein structures may receive too much weight and the regular secondary structures such as α-helices and β-strands may receive too little weight. While loop regions are important to the overall structure of a protein, the secondary structure regions are at least as important. In contrast to the semantic approach, the syntactic approach in IR considers all symbols within a document as equally important, with the possible exception of white space.

Given that the semantic IR approaches are not well-suited to protein structure comparisons, we adapted the syntactic approach to document similarity, originally referred to as shingling [10], to our linear encoding of protein structure. In the field of protein structure bioinformatics, this strategy is more often referred to as a sliding window strategy and the shingles are more often referred to as k-mers. However, we continue to use the shingle-related terms to remain consistent with the original paper from which we took the inspiration for our approach.

We transform a linear sequence of descriptors into a multiset of shingles consisting of 3 consecutive descriptors. The overlap between shingles ensures some of the order information within the original sequence of descriptors is preserved in the bag. By shingling, we obtain a multiset of sequences from a sequence of residue descriptors. As an example, the sequence of descriptors in Equation (3.1) becomes the following multiset of shingles.

$$\begin{aligned} \{\ &[5,5,5],[5,5,5],[5,5,5],[5,5,5],[5,5,7] \\ &[5,7,5],\underline{[7,5,11]},[5,11,11],[11,11,5],[11,5,5] \\ &[5,5,5],[5,5,5],[5,5,5],[5,5,5]\ \} \end{aligned} \tag{3.2}$$

Next, each shingle s is hashed to an integer s_{hash} as shown in Equation (3.3), where s_i is the i^{th} descriptor of the shingle s. The hash function used is a simplification of the hash function used in the Rabin-Karp algorithm [41]. The prime number 13 is used as the base since it is large enough to spread the descriptor values out in hash space without collisions.

$$s_{hash} = s_1 \times 13^2 + s_2 \times 13 + s_3 \tag{3.3}$$

After hashing, the multiset in Equation (3.2) becomes the following multiset of integers.

$$\begin{gathered} \{\, 915, 915, 915, 915, 917, 941, \underline{1259} \\ 999, 2007, 1929, 915, 915, 915, 915 \,\} \end{gathered} \tag{3.4}$$

This step completes the transformation of a sequence of residue descriptors to a multiset of integers that still retains some of the order information present in the original sequence.

Notice in Equation (3.4) the value 915, corresponding to the shingle $[5, 5, 5]$, occurs frequently, indicating the presence of β-strands. Since most proteins are dominated by regular secondary structures, the abundance of shingles for β-strands as well as the three types of helices, end up dominating comparisons. Moreover, since shingles are limited in length, this situation allows for structures with many short β-strands to match structures with fewer long β-strands. The same situation applies to helices.

To address this lack of specificity, we introduced a heuristic we called *run position encoding* (RPE), where a run is a consecutive sequence of identical descriptors. To distinguish between short and long runs, thereby increasing the specificity of the shingles, we add a factor of 10^5 to each shingle hash as a function of the first residue's position in a run i.

$$runfactor(i) = \begin{cases} i & \text{if } i < \lfloor l/2 \rfloor \\ l - i - 1 & \text{otherwise} \end{cases} \tag{3.5}$$

where i is zero-based and l is the length of the run. Multiplying the run factors by 10^5 and adding them to the shingle hashes places the run factors as the left-most digits in the hashes to avoid interference with the digits provided by the hash in Equation (3.3). This placement is also convenient for visual inspection since the run factors are isolated as the left-most digits.

The run factors for the sequence in Equation (3.1) are

$$[\,0, 1, 2, 2, 1, 0, 0, 0, 0, 0, 0, 1, 2, 2, 1, 0\,]. \tag{3.6}$$

Applying RPE to the multiset of integers in Equation (3.4) gives

$$\begin{gathered}\{\,00915, 10915, 20915, 20915, 10917, 00941, \underline{01259} \\ 00999, 02007, 01929, 00915, 10915, 20915, 20915\,\}\end{gathered} \tag{3.7}$$

where the leading zero run factors are shown for clarity.

The pyramidal approach to the run factors used in RPE preserves matches at the boundaries between secondary structure runs and loops that would not otherwise be preserved in the presence of differences in run lengths of one or more.

Once we have a representation of a protein structure as a bag of integers, similarity for a candidate pair of structures a and b can be defined as the Jaccard similarity [45] for multisets,

$$J(a, b) = \frac{\sum_i min(a_i, b_i)}{\sum_i max(a_i, b_i)}, \tag{3.8}$$

where i ranges over all possible shingle hashes s_i and a_i and b_i give the counts of shingle hash s_i in structures a and b, respectively.

In applying the Jaccard similarity to sets of RPE shingle hashes, we chose 3 as the length of the shingles to balance false positives, in the case of shorter shingles, against false negatives, in the case of longer shingles.

3.3 A Run-Factor Nuance

To see why RPE run factors are calculated at the descriptor level and applied at the shingle level, consider shingling a list of RPE run factors themselves, which mirrors applying them at the descriptor level.

The sequence of RPE factors

$[\,0, 1, 2, 3, 2, 1, 0\,]$ becomes

$\{\,[0, 1, 2, 3], [1, 2, 3, 2], [2, 3, 2, 1], [3, 2, 1, 0]\,\}$

and with one less element

$[\,0, 1, 2, 2, 1, 0\,]$ becomes

$\{\,[0, 1, 2, 2], [1, 2, 2, 1], [2, 2, 1, 0]\,\}$

Notice above, there is not a single shingle match for this one-off difference in run length. Now consider shingling a list of RPE factors, but this time all elements in the shingle are equal to the first run factor for the shingle, which mirrors applying run factors at the shingle level.

The sequence of RPE factors

$[\,0, 1, 2, 3, 2, 1, 0\,]$ becomes

$\{\,[0, 0, 0, 0], [1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3]\,\}$

and with one less element

$[\,0, 1, 2, 2, 1, 0\,]$ becomes

$\{\,[0, 0, 0, 0], [1, 1, 1, 1], [2, 2, 2, 2]\,\}$

In the latter case, a one-off difference in run length results in one less shingle match while still serving to increase the specificity of the shingles.

3.4 Min-Hashing and LSH

In IR, the syntactic bag of shingles representation of documents is used in the near dupe clustering of documents [12]. One application of near dupe clustering is in the review stage of electronic-discovery [39], which is the most expensive stage in a discovery process. Often millions of documents must be examined by a staff of attorneys to make a reasonable effort at providing all documents relevant to the discovery request. Grouping documents into near dupe clusters and assigning all documents within a cluster to a single reviewer reduces duplication of effort. In e-discovery, the syntactic approach to defining near dupe documents is necessary since an algorithmic definition of meaning is legally questionable.

In the case of near dupe clustering, each document must be compared to every other document in the collection, taking quadratic time. For this task, min-hashing [11] and locality sensitive hashing [36] (LSH), described below, can be combined to reduce this to subquadratic time. Although we do not near dupe cluster protein structures, we can still leverage the techniques of min-hashing and LSH to speed up protein structure searches by a large constant factor.

In addition to the near dupe clustering of documents, LSH has been used to group similar items together in an index for fast query processing. For example, RDF Indexing on Quads [42] (RIQ) uses LSH to group similar RDF graphs. First, RIQ represents RDF graphs as vectors. Second, RIQ uses LSH to determine pairs of graphs that exceed a similarity threshold. Third, using the pairs of similar graphs, connected components of similar graphs are used to determine the grouping of RDF graphs. By using LSH effectively, RIQ quickly filters out a large number of graphs that do not match the query.

3.4.1 Min-Hashing

At this stage, we have reduced protein structures to multisets of RPE shingles hashes similar to what is shown in Equation (3.7). Theoretically, we could stop now and simply compare protein structures by finding the Jaccard similarity between corresponding multisets of RPE shingles hashes. However, this would involve comparing variable-size multisets containing potentially hundreds of elements. It would be better to first convert these multisets to fixed-length signatures that can be easily compared and instead of obtaining the exact Jaccard similarities, obtain good-enough Jaccard similarity estimates from faster comparisons.

Min-hashing is used to create a fixed-length signature for a set of items by repeatedly randomly hashing the items, sorting the hashes into a list, and then selecting the minimum hash in each permuted list. If the same process is performed repeatedly on a pair of sets, the key result is that the probability of matching min-hashes for any given random permutation is equal to the Jaccard similarity of the two sets [11]. To approximate the Jaccard similarity for a given pair of sets, a sufficient number of min-hashes must be obtained.

The following example and proof are adapted and expanded to our specific use-case from the book *Mining of Massive Datasets* [53]. To illustrate min-hashing, we use the simpler definition of the Jaccard similarity for sets as opposed to the Jaccard similarity for multisets as shown in Equation (3.8). The Jaccard similarity for two sets a and b is defined as

$$J(a,b) = \frac{|a \cap b|}{|a \cup b|}. \tag{3.9}$$

First, Table 1 shows the characteristic matrix for the four sets S_1, S_2, S_3, and S_4. The first column contains the possible elements, the second column contains the hash of each element, and each subsequent column contains a 1 if the set for that column contains the element for that row or a 0 if not. For example, the set $S_1 = \{\, a, c, g \,\}$ and the set $S_3 = \{\, e, g \,\}$.

Next, Table 2 shows a random permutation of the rows of the characteristic matrix shown in Table 1. For each set, the hash for the first row for which the set contains the corresponding element is defined as the min-hash for that set. For example, $MINHASH(S_1) = h_a$ and $MINHASH(S_3) = h_e$.

Table 1: Characteristic matrix for sets

Element	**Hash**	S_1	S_2	S_3	S_4
a	h_a	1	0	0	0
b	h_b	0	0	0	1
c	h_c	1	1	0	0
d	h_d	0	1	0	0
e	h_e	0	0	1	1
f	h_f	0	0	0	1
g	h_g	1	1	1	0

Table 2: Permuted characteristic matrix for sets

Element	**Hash**	S_1	S_2	S_3	S_4
b	h_b	0	0	0	1
e	h_e	0	0	1	1
a	h_a	1	0	0	0
g	h_g	1	1	1	0
f	h_f	0	0	0	1
d	h_d	0	1	0	0
c	h_c	1	1	0	0

Table 2 shows only one random permutation of the characteristic matrix. If the matrix is repeatedly randomly permuted n times, the key result is that the probability of matching min-hashes between any two sets approaches the Jaccard similarity with increasing n, as stated above. To see this, consider the comparison of the sets S_1 and S_2 from Table 2 isolated in Table 3 with a new **Type** column. There are three types of rows; rows with both 1s are of type X, rows with both 0s are of type Z, and all other rows are of type Y. In a real example, the rows of type Z would be far more numerous, i.e., the matrix would be sparse.

From Table 3, let x equal the count of rows of type X, let y equal the count of rows

Table 3: Comparing permuted rows of sets S_1 and S_2 from Table 2

Element	**Hash**	S_1	S_2	**Type**
b	h_b	0	0	Z
e	h_e	0	0	Z
a	h_a	1	0	Y
g	h_g	1	1	X
f	h_f	0	0	Z
d	h_d	0	1	Y
c	h_c	1	1	X

of type Y, and disregard the rows of type Z. Then it is not hard to see that $x = |S_1 \cap S_2|$ and $x + y = |S_1 \cup S_2|$ since $S_1 = \{\, a, g, c \,\}$ and $S_2 = \{\, g, d, c \,\}$. Hence, from the Jaccard similarity for sets defined in Equation (3.9), we have

$$J(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = \frac{x}{x+y} = \frac{2}{2+2} = \frac{1}{2}, \tag{3.10}$$

as expected.

Now, consider the probability of equal min-hashes for any two sets a and b, that is, $Pr(MINHASH(a) = MINHASH(b))$. As we go down the rows of a permuted characteristic matrix for sets a and b, the probability of encountering a row of type X before a row of type Y is simply $\frac{x}{x+y}$ since there are x rows of type X and y rows of type Y and the total of both types is $x + y$. From Equation (3.10) above, this is equal to the Jaccard similarity. This proves the key result that the probability of matching min-hashes is equal to the Jaccard similarity. To estimate this probability, we need only repeat the permutations of the characteristic matrix a sufficient number of times to obtain a good-enough estimate of the Jaccard similarity when comparing min-hash signatures of any two sets.

In actual practice, to implement min-hashing, instead of repeatedly permuting a large data structure such as a characteristic matrix, it is far more practical to define n number of random hashes and for each hash, in a specific order, store the minimum hash value as part of the signature for each set. Hence, each set will have a signature consisting of n min-hashes and the proportion of matching min-hashes across all positions will estimate the Jaccard similarity.

In the above example, we only considered estimating the Jaccard similarity for sets. However, the same methodology can be easily adapted to multisets by converting multisets into sets. For instance, if some element a belongs to a multiset k times then we add indexed elements $a_1, a_2, a_3, ..., a_k$ to the corresponding set.

In our case, the sets contain indexed RPE shingle hashes, not to be confused with min-hashes. Following our example from Section 3.2, from indexing the multiset in Equation (3.7), we obtain the following set

$$\begin{array}{c} \{\, 100915, 110915, 120915, 220915, 110917, 100941, \underline{101259} \\ 100999, 102007, 101929, 200915, 210915, 320915, 420915\,\} \end{array}, \tag{3.11}$$

where for the i^{th} occurrence of an RPE shingle hash we prepend the index i to the integer.

For min-hashing the sets of indexed RPE shingle hashes, the hash function we use for calculating each min-hash is

$$h = (s_{hash} * RANDOM\_PRIMES(i)) \bmod RANDOM\_NUMBERS(i), \tag{3.12}$$

where s_{hash} is an RPE shingle hash, i indexes the i^{th} hash function, and $RANDOM\_PRIMES$ and $RANDOM\_NUMBERS$ are arrays that implicitly define the n hash functions. For each set and for each i^{th} hash function, all the hashes are calculated as in Equation (3.12) and the minimum hash is obtained and serves as the i^{th} min-hash for the set.

Finally, for each set of indexed RPE shingle hashes similar to Equation (3.11) we obtain $n = 99$ min-hashes from 99 hash functions implicitly defined by the $RANDOM\_PRIMES$ and $RANDOM\_NUMBERS$ arrays. At this stage, we have reduced our representation of protein structures from variable-sized sets of indexed RPE shingle hashes to fixed-length signatures. Given the key result above, the Jaccard similarity for any pair of protein structures can now be estimated by the proportion of matching min-hashes.

3.4.2 Locality Sensitive Hashing

Locality sensitive hashing (LSH) is a general technique of hashing, where items are hashed from one hash space to another such that items considered close in the former space are considered close in the latter space. Instead of discussing the general theory of LSH [36], we describe concretely how it is applied to min-hashing. Again, we follow the general outline of what is described in *Mininig of Massive Datasets* [53] adapted and expanded to our specific use-case.

Following the min-hashing steps described above, we are left with fixed-length signatures, which, although better than variable-sized sets, still have to be fully compared to estimate the Jaccard similarity of two proteins. If we planned on clustering proteins based on the Jaccard similarity using the min-hash signatures, we would still have a quadratic problem because the min-hash signature of every protein would have to be compared to that of every other protein. The following LSH technique can be used to reduce the clustering problem to subquadratic time. Although we are not clustering protein structures, we can still reduce our linear search through every available protein structure by a large constant factor by using LSH to reduce our search for candidate matches to having to match on only a single value as will be described below. In addition, the technique of LSH presents new opportunities for the parallelization and distribution of the search.

The idea for applying LSH to min-hashing is to hash disjoint sets of min-hashes to buckets such that for similar proteins, at least one pair of sets of min-hashes, one from each protein, will hash to an identical bucket. One way to approach this, as described in *Mining of Massive Datasets* [53], is known as the banding technique. For the banding technique, the min-hash signatures are divided into b bands of r rows and all rows of each band are hashed to a single hash for that band, which we refer to as a band-hash. Hence, hashing to an identical bucket is equivalent to matching on a band-hash.

Ideally, for the banding technique, we should use a hash function similar to how we hashed shingles to shingle hashes in Equation (3.3). However, for data type considerations, we simply sum the min-hashes within a band to calculate the band-hash. While this compromise may increase the probability of hash collisions, we only use the band-hashes as the first step to identify candidate matches and so invalid hash matches will be accounted for downstream in the search process.

For our implementation of the banding technique, we divide the 99 min-hashes in the signatures, taken in order, into bands of 3 min-hashes each, giving a total of 33 bands. For each band, the min-hashes it contains are summed to form the band-hash. Table 4 illustrates the banding technique applied to the CATH domain 1nycA00 shown in Fig. 17.

In Table 4, every 3 rows of min-hashes, taken in order, corresponds to 1 band-hash.

Table 4: Banding technique applied to the CATH domain 1nycA00 shown in Fig. 17

Min-Hashes	**Band-Hashes**
17874515	
5627160	24770792
1269117	
2978350	
9679892	23598363
10940121	
9467906	
2941545	13247476
838025	
…	…
49344	
6020442	11609537
762882	

To see why the banding technique works, consider the general case of dividing n min-hashes into b bands of r rows each. Recalling the key result from Section 3.4.1 above, for any two protein structures, the probability of matching min-hashes at any specific position is a 'good-enough' estimate for the Jaccard similarity of the two structures. Assume the Jaccard similarity between two protein structures is s, then the following derivation holds.

1. The probability of matching min-hashes for all rows of a specific band is s^r
2. implies the probability of not matching on all the rows of a specific band is $1 - s^r$
3. implies the probability of not matching on all the rows of all the bands is $(1 - s^r)^b$
4. implies the probability of matching on all the rows of at least one band is $1-(1-s^r)^b$

Applying this result to our specific case of $b = 33$ bands and $r = 3$ rows, for two protein structures having a Jaccard similarity of s, the probability of a single band match is $1 - (1 - s^3)^{33}$. A graph of this function is shown in Fig. 18. As indicated in the graph, for a Jaccard similarity of 0.50, the probability of matching on at least one band-hash is 0.9878, which is very high. Furthermore, even for a Jaccard similarity of just 0.275, there is a 50% probability of matching on at least one band-hash.

By requiring only a single band match to qualify as a candidate protein structure match, banding allows the problem of finding similar protein structures to be parallelized across bands since all that is needed for a candidate match is a single band match. This can be easily scaled up by having separate tables for each band indexed on the band hashes.

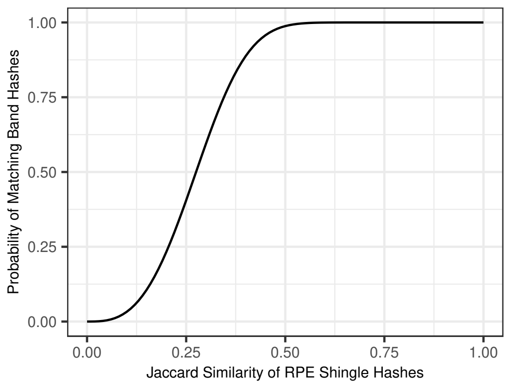


Figure 18: Graph of $1 - (1 - s^3)^{33}$

3.5 Index Implementation

Before RUPEE can service a search request, an offline process has to be executed to index the available protein structures. This index consist of residue descriptor sequences, RPE shingle hashes, min-hashes and band-hashes stored in a PostgreSQL database. If a user searches on a structure id, its representation will already be stored in the index. On the other hand, if a user uploads a protein structure, it will be parsed into a residue descriptor sequence, RPE shingle hashes, min-hashes, and band-hashes. Aside from the initial parsing, searching on an uploaded structure is identical to searching by structure id.

Here, to make the preceding discussion more concrete, we describe some of the table structures and provide some example data. For each of the protein structure databases RUPEE supports, that is, PDB whole chains, SCOPe, CATH, and ECOD, we maintain separate sets of tables. While it may be natural to store identically structured data from the different structure databases in the same tables and include a type column to distinguish between the different structure databases, we store the data from each structure database separately because RUPEE structure searches require full-table scans and minimizing the size of the tables optimizes for response times.

We chose to use PostgreSQL because it is the fastest freely available relational database that we can find. Moreover, rather than using a NoSQL [38] database for the storage of arrays of residue descriptors, RPE shingle hashes, min-hashes, and band-hashes, PostgresSQL provides array data types that allow us to minimize the row-overhead of having to store each element of these items in its own row, that is, fully normalized. Again, for the same reason, we maintain separate tables for each supported database, by using array data types we minimize the size of the tables that have to be scanned during searches which in turn minimizes the RUPEE search response times.

Fig. 19 shows the table definitions for the RUPEE CATH structure index. The tables containing metadata information for classification and identification are not shown. For the table shown in Fig. 19, the corresponding tables for the PDB whole chains, SCOPe, and ECOD structure databases are identical except for the names. The cath_descrs table stores the residue descriptors, the cath_grams table stores the RPE shingle hashes, and the cath_hashes table stores the min-hashes and band-hashes.

```
CREATE TABLE cath_descrs
(
    cath_id VARCHAR NOT NULL,
    descrs INTEGER ARRAY NOT NULL,
    coords REAL ARRAY NOT NULL
);
CREATE UNIQUE INDEX
    idx_cath_descrs_unique ON cath_descrs (cath_id);

CREATE TABLE cath_grams
(
    cath_id VARCHAR NOT NULL,
    grams INTEGER ARRAY NOT NULL
);
CREATE UNIQUE INDEX
    idx_cath_grams_unique ON cath_grams (cath_id);

CREATE TABLE cath_hashes
(
    cath_id VARCHAR NOT NULL,
    min_hashes INTEGER ARRAY NOT NULL,
    band_hashes INTEGER ARRAY NOT NULL
);
CREATE UNIQUE INDEX
    idx_cath_hashes_unique ON cath_hashes (cath_id);
```

Figure 19: PostgreSQL table definitions for CATH structure index

Table 5 shows the cath_descrs row for the 1nycA00 domain shown in Fig. 17. We only show the portion of the **descrs** field corresponding to the list of descriptors in Equation (3.1) since the entire field value would be quite large.

Table 5: cath_descrs row for CATH domain 1nycA00 shown in Fig. 17

Field	**Values**
cath_id	1nycA00
descrs	…5, 5, 5, 5, 5, 5, 7, 5, 11, 11, 5, 5, 5, 5, 5, 5…
coords	…12.217, 5.346, 10.729, 12.47, 8.503, 8.658, 15.469, 8.91, 6.274, 14.394, 11.695, 3.857, 11.368, 12.833, 1.876, 11.803, 9.784, -0.418, 12.93, 7.037, 1.936, 11.631, 5.45, 5.163, 12.679, 2.255, 6.852, 10.535, -0.429, 8.456, 11.455, -2.698, 11.326, 9.406, -5.224, 13.197, 9.543, -5.782, 16.97, 7.092, -7.052, 19.538, 5.401, -6.447, 22.804, 6.815, -9.511, 24.588…

Table 6 shows the cath_grams row for the 1nycA00 domain shown in Fig. 17. We only show the portion of the **grams** field corresponding to the multiset of RPE shingle hashes in Equation (3.7) since the entire field value would be quite large.

Table 6: cath_grams row for CATH domain 1nycA00 shown in Fig. 17

Field	**Values**
cath_id	1nycA00
grams	…915, 10915, 20915, 20915, 10917, 941, 1259, 999, 2007, 1929, 915, 10915, 20915, 20915…

Table 7 shows the cath_hashes row for the 1nycA00 domain shown in Fig. 17. Here, we show all 99 min-hashes and 33 band-hashes for the 1nycA00 domain.

Table 7: cath_hashes row for CATH domain 1nycA00 shown in Fig. 17

Field	Values
cath_id	1nycA00
min_hashes	17874515, 5627160, 1269117, 2978350, 9679892, 10940121, 9467906, 2941545, 838025, 6981121, 7102413, 2643484, 16775743, 1736842, 15713988, 19302649, 3717602, 887203, 7181034, 5644479, 19111656, 1509644, 3821572, 377663, 4014489, 4642950, 1253473, 21260794, 1508421, 5249121, 2759082, 158864, 8801740, 12428835, 16175007, 2088513, 5954522, 409643, 871425, 3416470, 4164386, 7635960, 8558087, 2484202, 12129673, 2390206, 2637883, 5157695, 452539, 4102894, 14017870, 580803, 1049450, 15260004, 4067591, 2091500, 19846484, 2289531, 550418, 3926685, 36435757, 3064676, 665873, 11076678, 527745, 8576108, 4161181, 456387, 2074542, 14305038, 5926958, 1003885, 62451, 1839644, 650257, 7035403, 1152809, 18828272, 4959709, 4331085, 2109837, 5763713, 11331092, 6250302, 1198427, 4868383, 296414, 1584731, 2607598, 11950087, 2931089, 6965705, 439284, 15223139, 7161321, 49344, 6020442, 762882
band_hashes	24770792, 23598363, 13247476, 16727018, 34226573, 23907454, 31937169, 5708879, 9910912, 28018336, 11719686, 30692355, 7235590, 15216816, 23171962, 10185784, 18573303, 16890257, 26005575, 6766634, 40166306, 20180531, 6692110, 21235881, 2552352, 27016484, 11400631, 23345107, 6363224, 16142416, 10336078, 22433804, 11609537

CHAPTER 4

SEARCHING PROTEIN STRUCTURES

Once protein structures have been indexed, as described in Chapter 3, they can be searched in a variety of ways with trade-offs for speed and quality of results. RUPEE provides three search modes; fast, top-aligned, and all-aligned. We first discuss the algorithms that are common to all search modes followed by a detailed discussion of the search modes themselves. Fig. 20 shows a flowchart for the RUPEE fast, top-aligned, and all-aligned search modes to help guide the discussion.

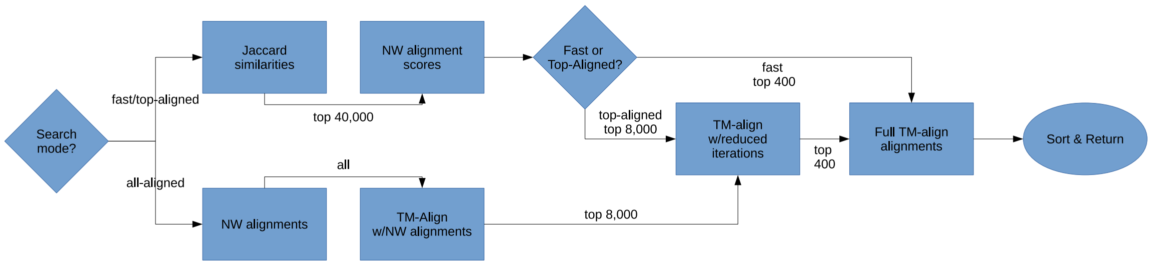


Figure 20: Flowchart for RUPEE search modes. It is assumed that descriptor sequences, shingles and min-hashes for all structures other than the query structure have been stored via an offline indexing process and are accessible throughout the flowchart.

4.1 Needleman-Wunsch Sequence Alignments

As mentioned in Section 1.3.1, the Needleman-Wunsch algorithm was initially introduced for biological sequence alignments, that is, nucleotide sequences and amino acid sequences. However, the NW algorithm can be applied to sequences in general and a variety of scoring methods can be used to satisfy specific objectives. In our case, we use NW global sequence alignments of residue descriptors for full-length protein structure searches and semi-global NW sequence alignments for containment searches. Here we describe each kind of alignment in detail.

4.1.1 NW Global Sequence Alignments

NW global sequence alignment is a dynamic programming algorithm [8] similar to the well-known longest common subsequence (LCS) algorithm. Whereas the LCS [30] algorithm only aligns exact matches, NW global sequence alignment allows for aligning mismatched symbols. Additionally, whereas the LCS algorithm simply optimizes for the raw count of exact matches, NW global sequence alignment can use different scores for aligning matches, mismatches, and gaps, besides simply counting exact matches.

For aligning biological sequences, the NW algorithm normally uses a substitution matrix for scoring along with a fixed penalty for aligning to gaps. For instance, for aligning amino acid sequences, aligning a mismatch between almost identical amino acids may receive a smaller penalty than would be received for aligning widely different amino acids. As an example of a substitution matrix, Fig. 21 shows the PAM250 substitution matrix, famously introduced by Margaret Dayhoff in the 1970s.

C	12																						
S	0	2																					
T	-2	1	3																				
P	-3	1	0	6																			
A	-2	1	1	1	2																		
G	-3	1	0	-1	1	5																	
N	-4	1	0	-1	0	0	2																
D	-5	0	0	-1	1	2	2	4															
E	-5	0	0	-1	0	0	1	3	4														
Q	-5	-1	-1	0	0	-1	1	2	2	4													
H	-3	-1	-1	0	-1	-2	2	1	1	3	6												
R	-4	0	-1	0	-2	-3	0	-1	-1	1	2	6											
K	-5	0	0	-1	-1	-2	1	0	0	1	0	3	5										
M	-5	-2	-1	-2	-1	-3	-2	-3	-2	-1	-2	0	0	6									
I	-2	-1	0	-2	-1	-3	-2	-2	-2	-2	-2	-2	-2	2	5								
L	-6	-3	-2	-3	-2	-4	-3	-4	-3	-2	-2	-3	-3	4	2	6							
V	-2	-1	0	-1	0	-1	-2	-2	-2	-2	-2	-2	-2	2	4	2	4						
F	-4	-3	-3	-5	-4	-5	-4	-6	-5	-5	-2	-4	-5	0	1	2	-1	9					
Y	0	-3	-3	-5	-3	-5	-2	-4	-4	-4	0	-4	-4	-2	-1	-1	-2	7	10				
W	-8	-2	-5	-6	-6	-7	-4	-7	-7	-5	-3	2	-3	-4	-5	-2	-6	0	0	17			
B	-4	0	0	-1	0	0	2	3	2	1	1	-1	1	-2	-2	-3	-2	-5	-3	-5	2		
Z	-5	0	-1	0	0	-1	1	3	3	3	2	0	0	-2	-2	-3	-2	-5	-4	-6	2	3	
	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	B	Z	

Figure 21: PAM250 substitution matrix

Rather than using a substitution matrix for aligning sequences of residue descriptors, we use a simple cost function c, where matches are awarded 1 point and mismatches are penalized 1 point. The cost function c is defined below for aligning the i^{th} element of sequence a to the j^{th} element of sequence b. Our fixed gap penalty is 1 point, the same as for aligning mismatched residue descriptors.

$$c(a_i, b_j) = \begin{cases} 1 & \text{if} a_i = b_j \\ -1 & \text{if} a_i \neq b_j \end{cases} \tag{4.1}$$

If we are aligning a sequence of residue descriptors a of length k with a sequence of residue descriptors b of length l, we initialize the first row and first column of the 0-indexed dynamic programming scoring matrix as shown below. Here, i indexes the rows of the scoring matrix and j indexes the columns of the scoring matrix. The first row and first column scores correspond to the scoring penalty accrued for opening gaps in the alignment.

$$\begin{aligned} &S(0,0) = 0 && \\ &S(i,0) = -i && \text{for } 0 < i \leq k \\ &S(0,j) = -j && \text{for } 0 < j \leq l \end{aligned} \tag{4.2}$$

The rest of the scoring matrix is filled in row by row from the top left to the bottom right. The dynamic programming recurrence is defined below, where the function c corresponds to the cost function defined in Equation (4.1) above.

$$S(i,j) = max \begin{cases} S(i-1,j-1) + c(a_i, b_j) \\ S(i,j-1) - 1 \\ S(i-1,j) - 1 \end{cases} \quad \text{for } 1 < i \leq k \text{ and } 1 < j \leq l \tag{4.3}$$

As an example, Fig. 22 shows the filled-in scoring matrix for the NW global sequence alignment of the residue descriptor sequences 958778 and 9578957585758. The first row and first column are separated from the rest of the rows and columns with a dashed line to highlight the scoring for opening gaps. The arrows in each cell indicate the cells from which the maximum scores for each cell came from. An arrow pointing up indicates aligning a gap to the residue descriptor for the current row, an arrow pointing left indicates aligning a gap to the residue descriptor for the current column, and an arrow pointing diagonally up and to the left indicates aligning the residue descriptors for the current row and column with each other.

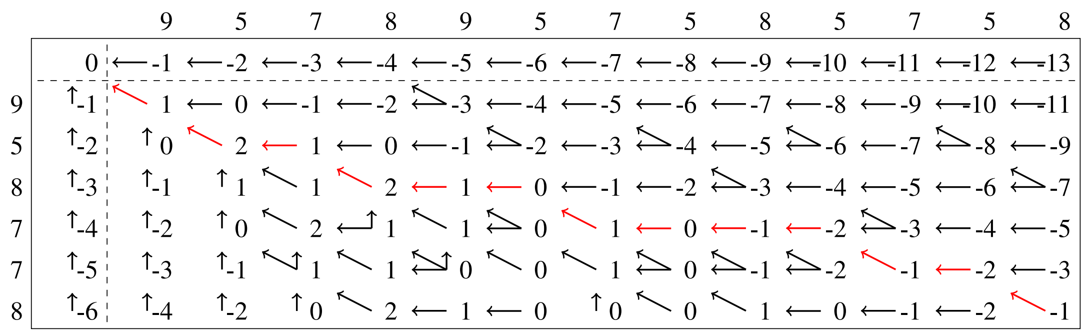


Figure 22: Needleman-Wunsch global sequence alignment for the residue descriptor sequences 958778 and 9578957585758

In Fig. 22, we find an optimal global sequence alignment, of which there may be several, by following the red arrows back up from the bottom right corner of the scoring matrix to the top left corner of the scoring matrix. In dynamic programming, this is referred to as the backward trace. By following the backward trace, we can construct an optimal global sequence alignment for the two sequences. For the NW global sequence alignment of 958778 with 9578957585758 we have the following optimal alignment indicated by the red arrows.

```
95_8__7___7_8

9578957585758
```

While this alignment would not be considered a good alignment, it is an optimal global sequence alignment for the two sequences. In the next section, we will find a better alignment for these two sequences by relaxing the global requirement.

4.1.2 Semi-Global NW Sequence Alignments

Semi-global Needleman-Wunsch sequence alignment [13] is a modification to the original NW global sequence alignment algorithm that does not penalize opening and ending gaps in the alignment for one or both of the sequences. We use semi-global sequence alignment that does not penalize the opening and ending gaps for only one of the sequences being aligned as part of our implementation of containment searches. The sequence of residue descriptors for which we do penalize opening and ending gaps is aligned *within* the sequence of residue descriptors for which do not penalize opening and ending gaps.

For our implementation of semi-global NW sequence alignment, we use the same cost function c as in Equation (4.1). Again, if we are aligning a sequence of residue descriptors a of length k with a sequence of residue descriptors b of length l, we initialize the first row and first column of the 0-indexed dynamic programming scoring matrix as shown below. We also assume the longer sequence is indexed by j. The only difference from Equation (4.2) above is that the first row is initialized with all 0s, corresponding to not penalizing the opening gaps for the longer sequence.

$$\begin{aligned} S(0,0) &= 0 \\ S(i,0) &= -i && \text{for } 0 < i \le k \\ S(0,j) &= 0 && \text{for } 0 < j \le l \end{aligned} \tag{4.4}$$

The recurrence for filling in the dynamic programming scoring matrix is almost identical to Equation (4.3) above except for the k^{th} row indexed by i needs a small modification to not penalize the ending gaps for the longer sequence. Here is the recurrence for filling in the scoring matrix up until the k^{th} row.

$$S(i,j) = max \begin{cases} S(i-1,j-1) + c(a_i, b_j) \\ S(i,j-1) - 1 & \text{for } 1 < i < k \text{ and } 1 < j \leq l \\ S(i-1,j) - 1 \end{cases} \tag{4.5}$$

Here is the recurrence for filling in the scoring matrix for the k^{th} row. The key difference is that we do not subtract 1 from $S(k-1,j)$ for the final term in the max expression, corresponding to ending gaps for the longer sequence.

$$S(k,j) = max \begin{cases} S(k-1,j-1) + c(a_k, b_j) \\ S(k,j-1) - 1 & \text{for } 1 < j \leq l \\ S(k-1,j) \end{cases} \tag{4.6}$$

As an example, Fig. 23 shows the filled-in scoring matrix for the semi-global NW sequence alignment of the residue descriptor sequences 958778 and 9578957585758. As in Fig. 22 above, the first row and first column are separated from the rest of the rows and columns with a dashed line to highlight the scoring for opening gaps. The arrows in each cell indicate the cells from which the maximum scores for each cell came from. An arrow pointing up indicates aligning a gap to the residue descriptor for the current row, an arrow pointing left indicates aligning a gap to the residue descriptor for the current column, and an arrow pointing diagonally up and to the left indicates aligning the residue descriptors for the current row and column with each other.

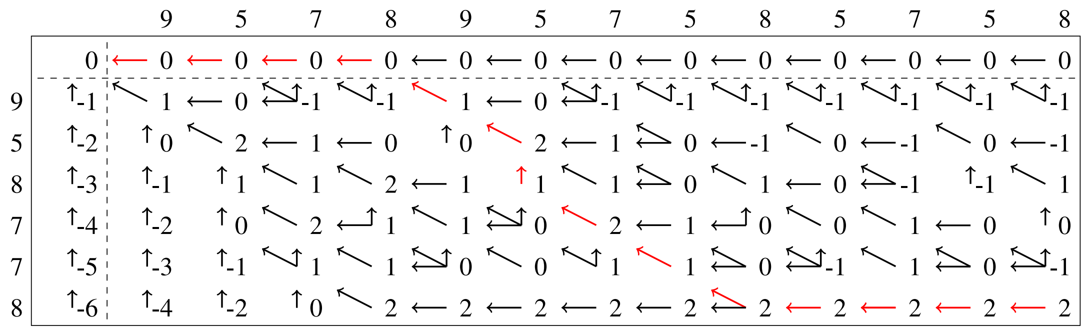


Figure 23: Semi-global Needleman-Wunsch sequence alignment for the residue descriptor sequences 958778 and 9578957585758

In Fig. 23, we find an optimal semi-global sequence alignment by following the backward trace as we did for Fig. 22. For the semi-global NW sequence alignment of 958778 with 9578957585758 we have the following optimal alignment indicated by the red arrows.

```
______958778______

957895_7585758
```

This alignment is considerably better than the previous alignment we found for the same two sequences using NW global sequence alignment.

4.2 TM-align Algorithm

As mentioned in Section 1.3.2, TM-align [69] is a protein structure alignment tool that optimizes for the TM-score [68]. Describing the full details of the TM-align algorithm is out-of-scope for this document. However, we will describe enough of the algorithm necessary to understand how we use TM-align.

First, it is important to highlight the difference between a sequence alignment and a structure alignment. Whereas a sequence alignment attempts to align symbols between two sequences of symbols, a structure alignment attempts to superimpose the 3-dimensional atomic coordinates of two protein structures as close as possible in space. However, it should be noted that sequence alignments can be used to give a rough approximation of structure similarity. For example, RUPEE uses sequence alignments of residue descriptors to identify candidate structures for closer examination.

Second, structure alignment algorithms often determine an alignment of residues corresponding to the pairs of 3-dimensional atomic coordinates to be superimposed. One potential misunderstanding is that a structure alignment algorithm does not attempt to align all the residues of one structure with all the residues of the other. This is easy to see in the case of different size proteins but it is even the case for equal size proteins. However, most structure alignment algorithms do attempt to align as many residues as possible. The *alignment length* is defined as the number of pairs of aligned residues and the *coverage* is defined as the percentage of aligned residues. In addition to inter-residue distances, most protein structure alignment algorithms optimize for alignment length or coverage.

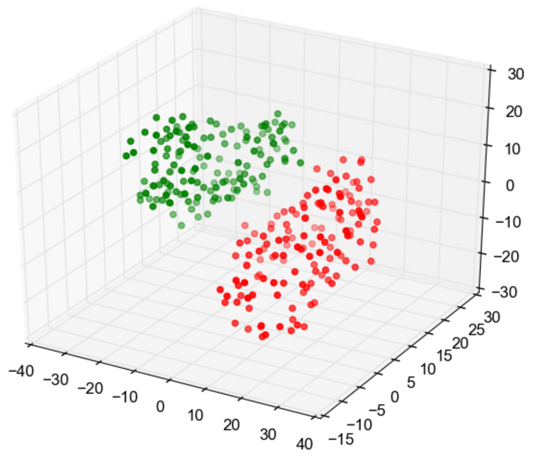
In the case of the TM-align protein structure alignment algorithm, the structure alignment optimizes for the TM-score, which factors in both inter-residue distances and alignment length. The TM-score is defined as

$$TMSCORE(a,b) = \frac{1}{l}\sum_{i}^{n}\frac{1}{1+\left(\frac{a_i-b_i}{d_0}\right)^2}, \tag{4.7}$$

where a and b are vectors of aligned residues and n is the number of aligned residues. l is the normalization length, which can be the length of the query protein, the length of the target protein or the average length of both proteins being aligned. Finally, d_0 is a constant factor that normalizes the distance between aligned residues so that the average TM-score is not dependent on the size of the protein structures being aligned. The TM-score is always a value between 0.0 and 1.0 and a perfect alignment will have a TM-score equal to 1.0.

To approximate the structure alignment with the optimal TM-score, first TM-align uses a variety of techniques to determine the *initial alignment*, that is, an alignment of residues to be structurally aligned in 3-dimensional space. Then, the initial alignment is improved using an iterative dynamic programming technique similar to that used by sequence alignment algorithms. However, instead of storing sequence alignment scores in each cell of the cost matrix, TM-align stores the best TM-score that can be obtained by superimposing in space the 3-dimensional coordinates of the aligned residues for that cell.

When filling in the cost matrix, for each cell representing a candidate residue alignment, the TM-score is calculated with the assistance of the Kabsch [40] algorithm.


(a) Before translation and rotation

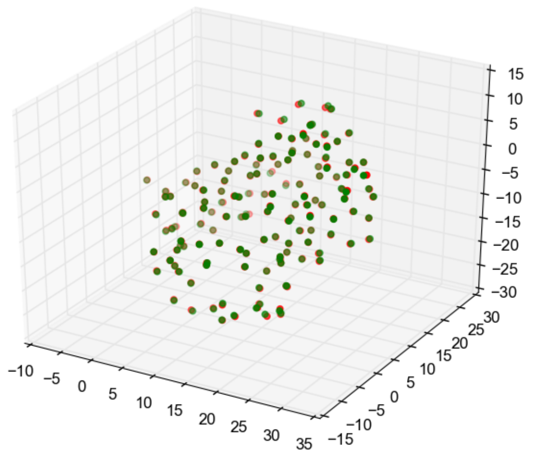

(b) After translation and rotation

Figure 24: Kabsch algorithm determines translation and rotation matrices to minimize RMSD between corresponding points

In general terms, given two sets of 3-dimensional points a and b and a one-to-one mapping of points from a to b, not necessarily onto, the Kabsch algorithm determines the rotation and translation matrices that minimize the RMSD between corresponding points in the map and returns the RMSD and the transformation matrices as output. Using the transformation matrices output from the Kabsch algorithm, TM-align superimposes the structures and calculates the TM-score as shown in Equation (4.7) above. This is done for every cell and guides the determination of the best residue alignment, transformation matrices, and TM-score, which are the final output of the TM-align structure alignment tool.

Fig. 24 demonstrates how the Kabsch algorithm works. In the left plot, the darker green and red points represent the mapped points. In the right plot, the green points are transformed by the rotation and translation matrices on top of the red points.

TM-align provides parameters that can speed up the determination of the initial alignment of residues, either by inputting a predefined initial alignment or running a reduced number of iterations in the determination of the initial alignment. If inputting a predefined initial alignment of residues, TM-align can also be instructed to *stick* to that initial alignment and not attempt to iteratively improve it at all.

4.3 Fast and Top-Aligned Search Modes

Originally, RUPEE just provided two search modes, fast and top-aligned. Both of these search modes use the results provided by the min-hashing [11] and LSH [36] initial filters described in Chapter 3 to quickly estimate the Jaccard similarity of a query protein against all available structures in the searched database, which in the case of ECOD is greater than 600,000 protein structures.

When a RUPEE structure search is executed in either fast or top-aligned search mode, several parallel tasks are executed corresponding to the 33 bands used for LSH as described in Chapter 3. These parallel tasks identify candidate matches based on a single band match. Then, for each candidate match, we estimate the Jaccard similarity using the min-hashes. In this way, RUPEE quickly compares the query structure to every structure in the database and returns the top-scoring 40000 protein structures based on the Jaccard similarity estimates.

Next, regarding the multiset of RPE shingle hashes in Equation (3.7) as an ordered sequence of integers, we obtain more accurate similarity scores for the top-scoring 40000 matches by performing NW sequence alignments as described in Section 4.1 using the hash sequences, where matches are awarded 1 point and mismatches and gaps are penalized 1 point.

For fast search mode, after the NW sequence alignments are performed for the top-scoring 40000 protein structures, full structure alignments using TM-align are performed on the top-scoring 400 structures and the results are immediately returned sorted by TM-score.

For top-aligned search mode, after the NW sequence alignments are performed on the top-scoring 40000 protein structures, we execute TM-align on the top-scoring 8000 matches using a reduced number of dynamic programming iterations for improving the initial alignments in the TM-align algorithm as described in Section 4.2 and obtain the top 400 matches. Finally, we execute TM-align using the default number of dynamic programming iterations on the top 400 matches and return the results sorted by TM-score.

The filter sizes of 40000, 8000, and 400 have been chosen based on the quality of results and speed. We found that increasing the size of either of these filters results in only marginal improvements in the quality of results. Given that performing the TM-align structure alignments is the most time-consuming aspect of the RUPEE structure search, the marginal improvements gained from larger filter sizes have to be balanced against the number of TM-align structure alignments performed.

4.4 All-Aligned Search Modes

While fast and top-aligned search modes may be sufficient for searching for known protein structures [6], the need for greater sensitivity arises when searching with predicted structures that may only have a maximum TM-score of less than 0.50 when compared against all available structures. Furthermore, for fast and top-aligned search modes, the effectiveness of the containment searches discussed below is limited by the initial filtering using Jaccard similarity estimates, which biases the initial filtering toward full-length matches. We addressed both of these concerns with the addition of all-aligned search mode.

In contrast to fast and top-aligned, all-aligned search mode skips the initial step of using min-hashing and LSH filtering. Instead, all-aligned runs the NW sequence alignments on all available structures using the residue descriptor sequences to obtain residue descriptor sequence alignments rather than just scores as in fast and top-aligned search modes. The residue descriptor sequence alignments are then passed into TM-align as the initial alignments and TM-align is set to stick to those initial alignments as described in Section 4.2. Along with the residue descriptor initial alignments, we pass in the corresponding atomic coordinates for the aligned residues, that are stored in the descriptor tables as described in Section 3.5. Skipping the min-hashing and LSH filtering combined with initializing TM-align with the NW sequence alignments increases the sensitivity of RUPEE in all-aligned search mode and can support containment searches at all stages of processing.

Once the TM-align structure alignments are run on all available structures using the NW residue descriptor sequence alignments as initial alignments, the top-scoring 8000 are obtained. As is done in top-aligned search mode, we run TM-align with a reduced number of iterations on these top-scoring 8000 structures to obtain the top-scoring 400 and finally run TM-align with the default number of iterations on these to obtain the final results sorted by TM-score.

While running NW sequence alignments on all available structures reduces the scalability of all-aligned search mode, it is still reasonably fast as will be shown in Section 5.2.

4.5 Containment Searches

By default, TM-align normalizes by the length of the query structure, that is, the first structure passed to it. This normalization is asymmetric since higher TM-scores result when the query structure is smaller than the target structure and conversely, lower TM-scores result when the query structure is larger than the target structure. In extreme cases, high TM-scores can be achieved even when the target structure is 10 times larger than the query structure so long as the query structure can be aligned somewhere within the target structure.

When we first introduced RUPEE with fast and top-aligned search modes, the search was implicitly a full-length search. This fits naturally with the Jaccard similarity estimates returned from the min-hashing and LSH steps since the Jaccard similarity measures full set similarity, which in this case, translates directly to full-length protein structure similarity. To match the kinds of measures across all steps, instead of using the default setting for performing the TM-align structure alignments, we explicitly used the TM-align option of normalizing by the average length of compared structures.

With the addition of all-aligned search mode, we were no longer bound by the full-length min-hashing and LSH steps. We recognized that by using alignment normalization effectively at each stage, RUPEE can also execute containment searches in addition to full-length searches. We introduced two kinds of containment searches, *Contained-In* and *Contains*. For both kinds of containment searches, we use semi-global NW sequence alignments as described in Section 4.1.2. For Contained-In searches, the query structure is set as the shorter structure indexed by i in Fig. 23 and for Contains searches, the query structure is set as the longer structure indexed by j in Fig. 23.

Similar to how we use global and semi-global NW sequence alignments for full-length and containment search types respectively, we apply the same logic to how we use normalization in the TM-align algorithm to be compatible with the NW sequence alignments, normalizing by the query structure for Contained-In searches, by the target structure for Contains searches, and by the average length of structures for Full-Length searches.

Despite that fast and top-aligned search modes still rely on min-hashing and LSH for the initial filtering, we have extended the option of containment searches to both modes. However, for fast and top-aligned search modes, the containment criterion does not factor into the search until after the min-hashing and LSH initial filtering. This does limit the efficacy of fast and top-aligned search modes for containment searches; however, they are still quite effective as will be shown in Section 5.2.

With support for containment searches, RUPEE can be used to search for structural motifs within proteins or look for matches of smaller structures within larger structures.

CHAPTER 5

RESULTS AND EVALUATION

5.1 Results for Purely Geometric Structure Search

For our first objective, we published the paper ”RUPEE: A fast and accurate purely geometric protein structure search” [6], from which the results in this section are taken. At that time, we had not yet introduced all-aligned search mode or containments searches, which were introduced in our later work [7]. Therefore, when we compare RUPEE to mTM-align normalized by the query structure in Fig. 26 below, RUPEE is at a disadvantage because we did not yet have a way for RUPEE to normalize by the query structure internally as described in Chapter 4. We addressed this disadvantage in our later work on RUPEE [7] as described in the next half of this chapter when we use containment searches in our comparisons to mTM-align to match the normalization method used by mTM-align.

Protein structure searches can be evaluated using structure alignment scores like RMSD and TM-score as described in Section 1.3.2 or by comparison of results against the hierarchy of a protein structure classification database, which usually consists of structural domains. Among protein structure classification databases for which corresponding structure searches exist, SCOPe [26] and CATH [49] are the most popular.

For our results, we derived three benchmarks of structural domains, scop_d360, scop_d62, and cath_d99, for pairwise evaluations against the mTM-align structure search [25], the secondary structure matching (SSM) search [44], and the CATHEDRAL structural scan [55] available at the CATH web site, respectively. To avoid introducing our own bias, we were careful to derive each benchmark from an existing benchmark used in a previously published work or an existing list of protein domains provided by an independent third party. scop_d360 was derived from the d500 benchmark used in [25] filtered for domains in SCOPe 2.07 for which mTM-align returned 100 or more results. scoo_d360 contains domains from 262 distinct folds. Similarly, scop_d62 was derived from the d500 benchmark filtered for domains defined in SCOP 1.73 for which SSM returned 50 or more results. scop_d62 contains domains from 53 distinct folds. For all domains contained in the d500 benchmark, RUPEE returned 100 or more results. The cath_d99 benchmark contains 99 superfamily representatives from the top 100 most diverse superfamilies defined in CATH v4.2 for which CATHEDRAL returned results in less than 12 hours.

We performed pairwise evaluations to ensure the fairness of our comparisons. First, for domain searches, SSM was working with the SCOP 1.73 database, so accordingly we had RUPEE search on SCOP 1.73 domains to ensure RUPEE did not have more domains to work with for scoring and precision evaluations. Second, mTM-align was updated to work with SCOPe 2.07 domain definitions but still retained domains from 2.06 that had since been redefined either through mergers or splits in 2.07. On the other hand, CATHEDRAL presented no such challenges but still required a separate benchmark since it was working with a distinct set of domains, CATH v4.2.

Benchmark definitions for scop_d360, scop_d62, and cath_d99 can be found in appendix A.

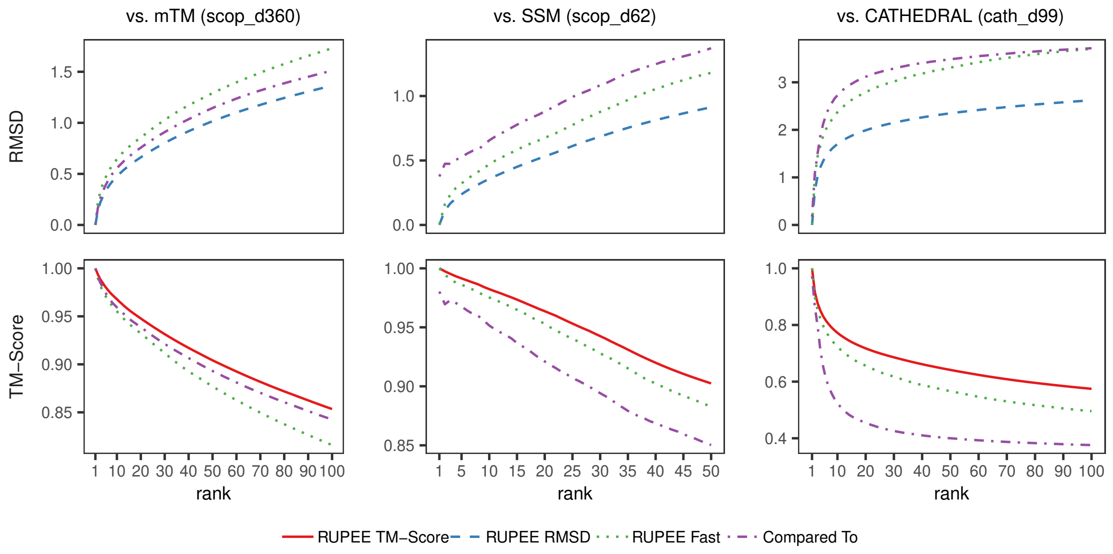


Figure 25: Scoring from TM-align structure alignments normalized by the average length of aligned structures for RUPEE fast and top-aligned search modes sorted by TM-score, and RUPEE top-aligned search mode sorted by RMSD

5.1.1 Scoring

Fig. 25 shows average cumulative values for each ranked result averaged over all searches. Both RMSD and TM-score values are shown, provided as outputs from TM-align structure alignments normalized by the average length of aligned structures. A TM-score above 0.5 is a good predictor for whether or not two domains are in the same fold [37]. TM-scores greater than 0.17 are considered potentially meaningful whereas TM-scores less than 0.17 are considered to be due to random alignments [68].

RUPEE fast and top-aligned sorted by RMSD, and top-aligned sorted by TM-score, performed better than SSM and CATHEDRAL. The scoring in the cath_d99 benchmark comparisons were notably lower than for the other two benchmarks. This was expected since CATHEDRAL only returned CATH s35 representatives. Likewise, for this comparison RUPEE filtered for s35 representatives to match. Given that the cath_d99 benchmark was evaluated against representatives, there were fewer highly similar structures returned in the results.

In our evaluation, mTM-align faired better than SSM and CATHEDRAL. mTM-align also performed better than RUPEE fast, although RUPEE fast was still within 0.08 TM-score points of mTM-align at the 100th result, which is notable considering its speed.

For both TM-score and RMSD, RUPEE top-aligned performed better than mTM-align. For RMSD, RUPEE top-aligned performed better than mTM-align but this can most likely be attributed to the fact that mTM-align is only sorted by TM-score. If mTM-align sorted by RMSD, their results likely would be improved. Nevertheless, it is worth noting that the initial min-hashing and LSH technique used by RUPEE does not explicitly bias results towards one particular full-length measure.

Fig. 26 again shows average cumulative values for each ranked result averaged over all searches but only for RUPEE and mTM-align. The difference here is that instead of normalizing by the average length of the protein structures we show results for TM-align normalized by the length of the query structure. This time, mTM-align performed better than RUPEE top-aligned sorted by TM-score.

Taken together, Figs. 25 and 26 show that the results of RUPEE and mTM-align are roughly equal and where they did differ is a result of the differences in normalization.

As mentioned in Section 1.3.2, Combinatorial Extensions [60] (CE) and FATCAT [66] are among the most popular structure alignment tools, representing rigid and

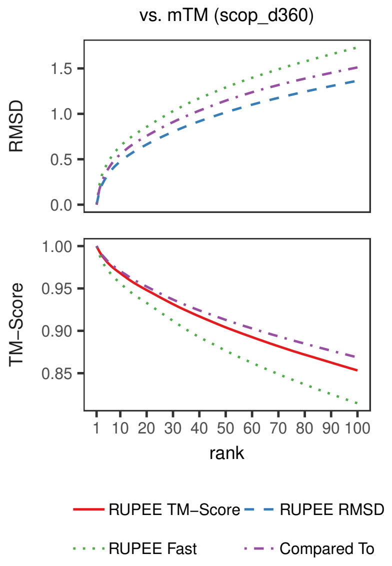


Figure 26: Scoring from TM-align structure alignments normalized by the length of the query structure for RUPEE fast and top-aligned search modes sorted by TM-score, and RUPEE top-aligned search mode sorted by RMSD

flexible protein structure alignments, respectively. CE performs a rigid alignment to minimize RMSD and FATCAT allows for a constrained number of twists in the protein chain to find a more flexible alignment before minimizing RMSD.

In addition to reporting the RMSD for their determined alignments, CE and FATCAT also return the TM-score. In Fig. 27, we aligned the top-100 scoring results from RUPEE, mTM-align, SSM, and CATHEDRAL using CE and provide the plots of RMSD and TM-score across all ranks. Fig. 28 is the same as Fig. 27 with FATCAT used as the aligner instead of CE.

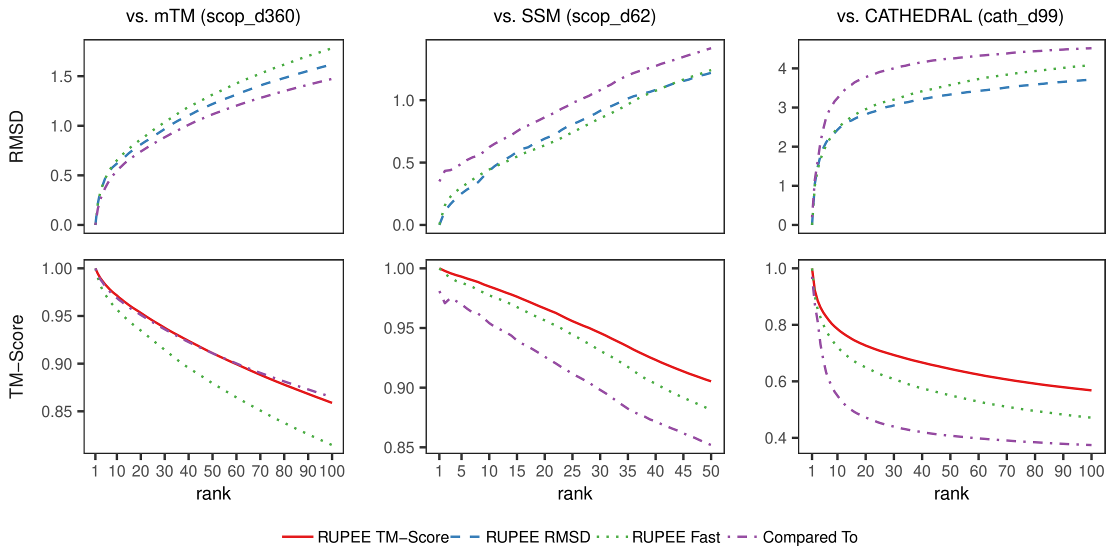


Figure 27: Scoring from CE structure alignments for RUPEE fast, RUPEE top-aligned sorted by TM-Score, and RUPEE top-aligned sorted by RMSD

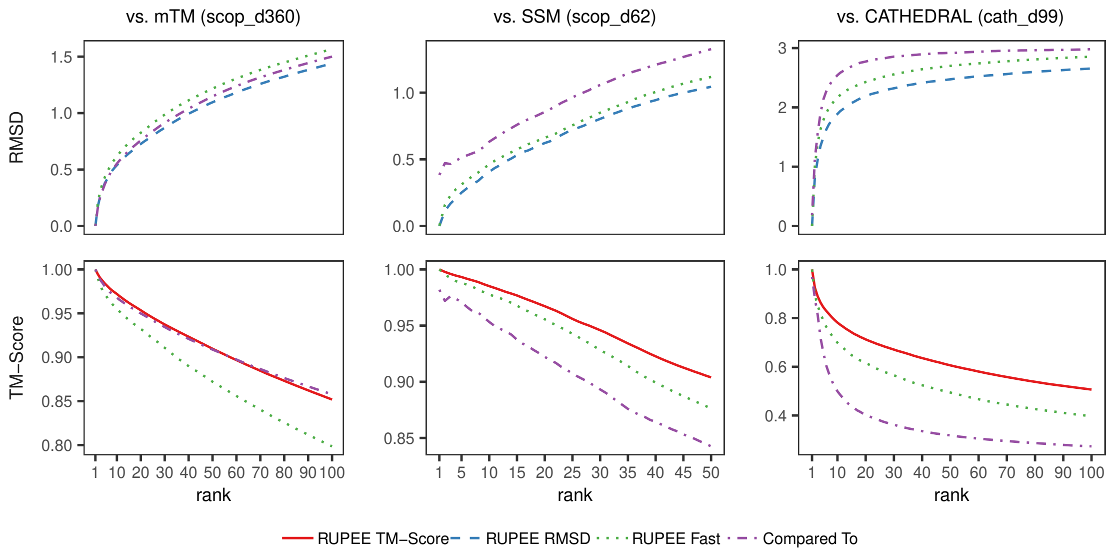


Figure 28: Scoring from FATCAT structure alignments for RUPEE fast, RUPEE top-aligned sorted by TM-Score, and RUPEE top-aligned sorted by RMSD

Once again, Figs. 27 and 28 show that RUPEE performs significantly better than SSM and CATHEDRAL for both RMSD and TM-score. Likewise, the performance of RUPEE is almost identical to the performance of mTM-align.

5.1.2 Precision

Fig. 29 shows precision (i.e., positive predictive value or PPV) averaged over all searches, where positive results are defined as domains with the same classification for the indicated hierarchy level as the query domain. A plot of recall is unnecessary since Fig. 29 provides precision at specific ranks for identical sets of searches. Hence, recall curves have the same relative relationships as those shown for precision.

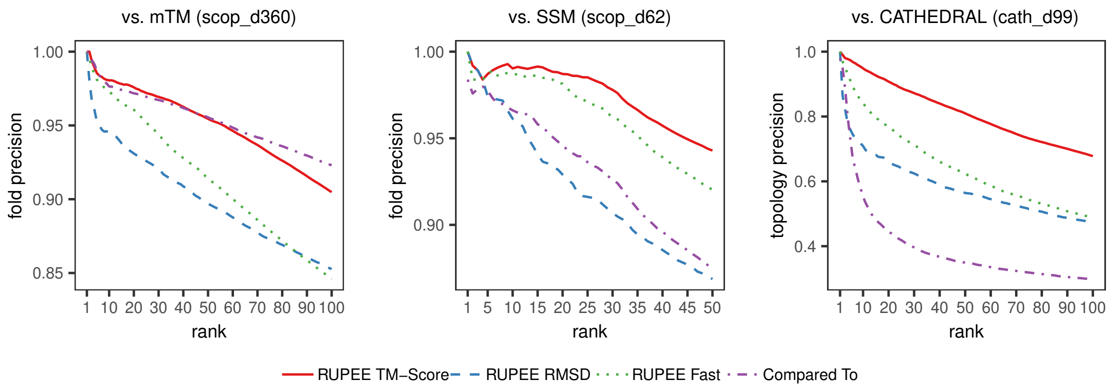


Figure 29: Precision for RUPEE fast and top-aligned search modes sorted by TM-score, and RUPEE top-aligned search mode sorted by RMSD

We would expect a structure search to have reasonable precision with respect to the hierarchy levels of the structure classification it is searching. However, it is not clear how to define reasonable. On the other hand, if precision is too high, the search provides little value beyond that provided by the structure classification hierarchy it is searching.

Towards the extreme end of high precision, it would be sufficient for a search to return the best match and from there refer to the hierarchy for additional results.

RUPEE fast and top-aligned sorted by TM-score showed higher precision than SSM and CATHEDRAL. mTM-align showed higher precision than RUPEE fast. However, RUPEE top-aligned sorted by TM-score showed equal or higher precision than mTM-align up to the 50th result and then dropped below that of mTM-align. The lower precision after the 50th result can be attributed, in large part, to the differences in normalization. A good example of this is for the search on domain d1j6va_ from scop_d360 having 148 residues. After the initial block of closely matched structures, mTM-align began returning structures within the same fold with more than 200 residues like d1q2la2 and d1q2la3. On the other hand, RUPEE began to look outside the fold to structures like d2fyxb2 and d2fyxa1 that provided better full-length alignments with respect to both the query and the target protein.

RUPEE top-aligned sorted by RMSD showed higher precision than for CATHEDRAL but lower precision than mTM-align and SSM. Also, RUPEE top-aligned sorted by RMSD showed lower precision than both RUPEE fast and RUPEE top-aligned sorted by TM-score. The lower precision for RUPEE top-aligned sorted by RMSD is a direct consequence of RMSD not being suitable for full-length alignments as discussed in Section 1.3.2. For this reason, it is advised to sort RUPEE results by TM-score. RUPEE top-align provides a sort by RMSD only for the sake of completeness.

5.1.3 Response Times

Response times to a large degree are a measure of the amount of resources available to an application. Response times for RUPEE were gathered from the RUPEE web site running on a single Amazon Web Services (AWS) c5.2xlarge elastic compute (EC2) unit. With more resources, RUPEE response times can be further improved since TM-align structure alignments can be run in parallel. For mTM-align, SSM, and CATHEDRAL, we gathered response times by automating their respective web sites using the Selenium WebDriver API.

Given that our response time comparisons were made against the respective tools running in different environments, we did not derive solid conclusions about the efficiency of the methods themselves. Nevertheless, these response times do fairly compare the user experience of the respective tools. Moreover, in some cases, the response times differed dramatically, by an order of magnitude in the case of RUPEE fast search mode.

Fig. 30 shows response times in seconds for the scop_d62 and cath_d99 benchmarks. Here, we were able to show RUPEE fast and top-aligned search modes, mTM-align and SSM on the same plot because scop_d62 is a subset of the scop_d360 benchmark. Both plots are shown with a logarithmic scale to include all outliers while still being able to view the overall trends in response times. We also provided Loess regression curves to further highlight the overall trends.

In all cases, RUPEE fast search mode was considerably faster than all other searches. It is also clear that fast search mode was not as sensitive to increasing residue counts in contrast to RUPEE top-aligned search mode, mTM-align, and CATHEDRAL. Response

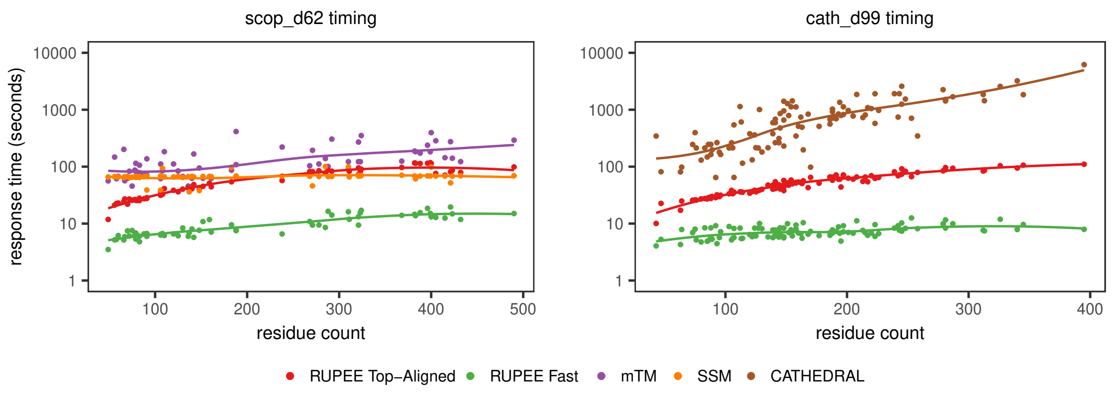


Figure 30: Response times for RUPEE fast and top-aligned search modes. The response times for RUPEE top-aligned search mode are dominated by TM-align structure alignments and do not depend on the sort order.

times for SSM were not affected by residue counts at all and always returned results in less than 100 seconds but this was at the expense of performance as shown in Fig. 25.

The left plot of Fig. 30 shows that RUPEE top-aligned search mode was faster than mTM-align for all residue counts. Likewise, the right plot of Fig. 30 shows that RUPEE top-aligned search mode was significantly faster than CATHEDRAL for all residue counts.

The trend of increasing response times for RUPEE top-aligned search mode is a direct result of the TM-align structure alignments that are performed on the top-scoring 8000 results provided by the NW sequence alignments as described in Chapter 4 and shown in Fig. 20.

5.2 Results for Predicted Structure Search

For our second objective, we published the paper ”Protein structure search to support the development of protein structure prediction methods” [7], from which the results in this section are taken. Like our earlier work on RUPEE [6] as described in Section 5.1, we compared response times and the average scores of ranked results for each rank across a set of benchmark structures to those of mTM-align [25], SSM [44], CATHEDRAL [55], and VAST [28]. However, this time we included results for RUPEE all-aligned search mode, and instead of using a benchmark of known protein structures, we used a benchmark derived from structure predictions submitted to CASP13.

Again, like our earlier work on RUPEE [6] as described in Section 5.1, we compared to each structure search individually to reduce sources of systemic error in our evaluation. Each comparison to mTM-align, SSM, CATHEDRAL, and VAST is discussed in its own section below. We also compared all three RUPEE search modes to an exhaustive search using full TM-align structure alignments on every available structure. For comparing RUPEE to the exhaustive search, we used both a benchmark of protein structure predictions and a benchmark of known protein structures to illustrate differences in performance based on benchmark difficulty.

5.2.1 Benchmarks

To evaluate the results of RUPEE against mTM-align [25], SSM [44], CATHEDRAL [55] and VAST [28] for the case of providing support for the development of protein structure prediction methods, we derived our initial benchmark from structure predictions

Table 8: Top 10 performing CASP13 prediction groups ranked by the Assessors formula (GDT_TS + QCS) applied to free-modeling targets

Number	**Name**
043	A7D
322	Zhang
145	QUARK
089	MULTICOM
261	Zhang-Server
224	Destini
498	RaptorX-Contact
197	MESHI
354	wfAll-Cheng
196	Grudinin

submitted to CASP13. To ensure the benchmark was challenging, we only considered predictions submitted for all 25 single-segment free-modeling (FM) target domains in CASP13 [43]. To ensure the benchmark was not too challenging, we only considered the first designated predictions of the top 10 performing groups ranked by the Assessors' formula (GDT_TS + QCS) applied to free-modeling targets. We called this benchmark casp_d250 since it consists of 250 structures, corresponding to 25 target domains for each of the top 10 performing groups. The top 10 performing CASP13 prediction groups are shown in Table 8.

While the casp_d250 benchmark does have some redundancy given that every 10 structures are structure predictions for the same target domain, Fig. 31 shows there is some variability within each group of 10 structure predictions for each target. In order to measure variability, we determined the SCOPe superfamily corresponding to the best search result for each of the 250 benchmark structures and then counted the number of

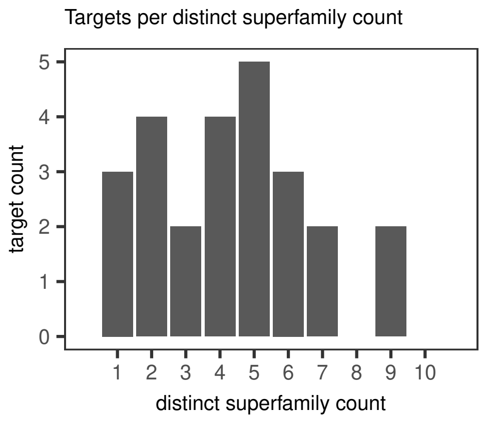


Figure 31: The target count for each count of distinct SCOPe superfamilies within each group of 10 predictions for the same target

distinct superfamilies in each group of 10 structures for each target structure. In Fig. 31, we show the target count for each count of distinct SCOPe superfamilies within a group of 10 predictions for the same target. The average number of distinct SCOPe superfamilies within a group of 10 predictions for the same target is 4.32.

The difficulty of the casp_d250 benchmark presented challenges for some protein structure searches. While we would have liked to compare our results to DALI [32], another popular purely geometric protein structure search based on inter-residue distances, DALI was unable to return any results for a majority of the benchmark structures. On the other hand, mTM-align, SSM, CATHEDRAL, and VAST returned results for all but a few missed structures. To avoid comparing against potential bugs, when there were missed structures for a specific comparison, we created a new benchmark for that comparison that excludes the missed structures from our initial casp_d250 benchmark. For all benchmarks structures, we required that at least 100 results were returned, except for mTM-align, which only returned 10 results for a majority of structures. The benchmark names indicate the number of structures included in the benchmark and appear in the title of each plot below.

All benchmark definitions for this section can be found in Appendix B. We uniquely identify each benchmark structure using the format $\langle\text{target}\rangle\text{TS}\langle\text{group}\rangle\text{-}\langle\text{domain}\rangle$. For instance, the prediction submitted by the AlphaFold team named A7D and numbered 043 for the second domain in the target T0960 is referred to as T0960TS043-D2.

5.2.2 Scoring

To evaluate scoring fairly, in addition to the search types *Contained-In*, *Contains* and *Full-Length* that all depend on the TM-score as described above, we added the additional search types *Q-score* [44] and *SSAP-score* [50] to RUPEE to perform comparisons to SSM and CATHEDRAL using their native scores, respectively. The additional search types demonstrate the pluggable nature of RUPEE. Although we still use TM-align for all internal structure alignments after the initial filtering and NW sequence alignments, we can easily apply different scores to the alignments provided by TM-align besides the TM-score.

For the TM-score comparisons, we normalized by either the length of the query structure or the average length of the structures being compared. For the TM-score plots, in the vertical axis, we use (q) to indicate normalization by the length of the query structure or (avg) to indicate normalization by the average length of the two structures.

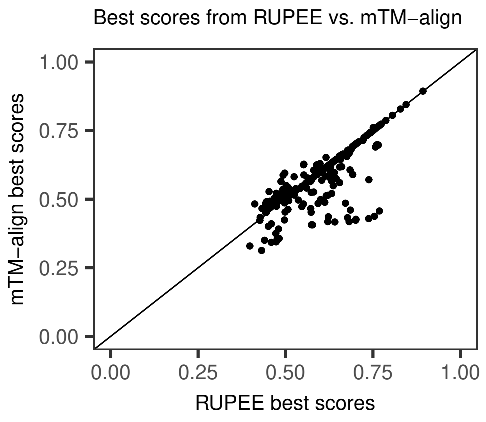


Figure 32: Plot of the best scores returned by RUPEE all-aligned and mTM-align for structure searches for each casp_d250 benchmark structure

5.2.2.1 Scoring vs. mTM-align

In Fig. 32, we compare the average TM-scores [68] of the top 100 results for each rank across the casp_d250 benchmark for RUPEE and mTM-align [25], searching whole PDB chains deposited in the PDB as of 2020-08-28. We used the RUPEE Contained-In search type to search by TM-score normalized by the query structure, identical to the scoring used by mTM-align. All TM-scores were calculated using TM-align [69].

mTM-align only returned 10 results for 96 benchmark structures and returned 100 or more results for only 85 benchmark structures. For the first 10 results, mTM-align did better than RUPEE fast and top-aligned and stayed within 0.02 TM-score points of RUPEE all-aligned but then dropped off precipitously. In Fig. 32 we did not cut off the plot after 10 results because that would have been unfair to RUPEE to not highlight the problem that mTM-align had with respect to the number of results it was able to return for difficult searches. Since protein structure search results usually contain blocks of highly similar structures within the same fold, it is important to return a sufficient number of results to show structures from multiple folds, especially when attempting to validate that a predicted structure is within the neighborhood of the expected fold.

Since the performance of mTM-align was almost as good as RUPEE all-aligned for the first 10 results, we examined the relationship between the scores more closely. In Fig. 33 we show a point for each casp_d250 benchmark structure corresponding to the best score for that structure from both RUPEE all-aligned and mTM-align. In Fig. 33, the points below the diagonal indicate a higher score for RUPEE and the points on the diagonal indicate equal scores. From Fig. 33, it is clear that mTM-align performs well for searching on structures that have higher structure similarity to known structures. On the other hand, the performance of mTM-align quickly declined when the TM-scores dropped below 0.77. This further highlights that RUPEE has more sensitivity in comparison to other protein structure searches for identifying similar structures to predicted structures that have low sequence and structure similarity to known structures.

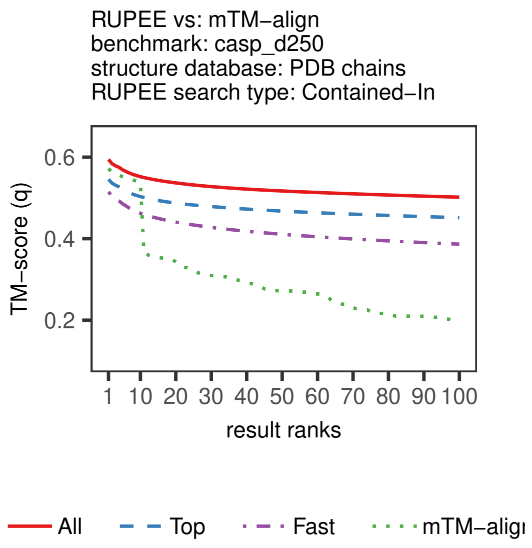


Figure 33: RUPEE vs mTM-Align. Comparison of average TM-scores for result ranks from 1 to 100 across the casp_d250 benchmark

5.2.2.2 Scoring vs. SSM

In Fig. 34, we compare the average Q-scores [44], a measure of full-length structure similarity, of the top 100 results for each rank across the casp_ssm_d248 benchmark for RUPEE and SSM [44], searching the SCOP v1.73 domains. We calculated our own Q-scores for SSM because we observed the scores that SSM provided were wildly incorrect in many cases. For instance, a large set of results all started with a block of perfect matches with a Q-score of 1.0 and RMSD of 0.0, which is clearly impossible given that we were searching with predicted structures. We did not observe this problem with SSM when searching on known protein structures. To be fair, we sorted the top 100 results from SSM by the calculated Q-score.

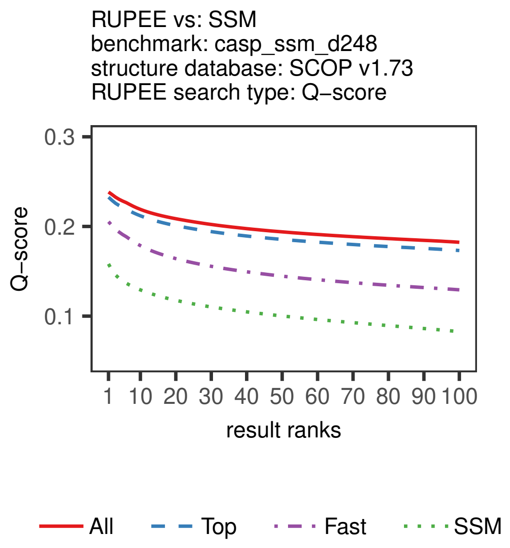


Figure 34: RUPEE vs SSM. Comparison of average Q-scores for result ranks from 1 to 100 across the casp_ssm_d248 benchmark

As shown in Fig. 34, all RUPEE search modes performed better than SSM at all ranks.

5.2.2.3 Scoring vs. CATHEDRAL

In Fig. 35, we compare the average SSAP-scores [50] and TM-scores [68] of the top 100 results for each rank across the casp_cathedral_d247 benchmark for RUPEE and CATHEDRAL [55], searching the CATH v4.2 domains. For comparing to CATHEDRAL, we filtered by CATH s35 cluster representatives since that is all that CATHEDRAL returns. We included the TM-score in our comparisons because the SSAP-score [50] is not well-documented and supported as a measure of full-length structure similarity. We used the cath-ssap tool provided in the cath-tools suite [23] to calculate the SSAP-scores and we

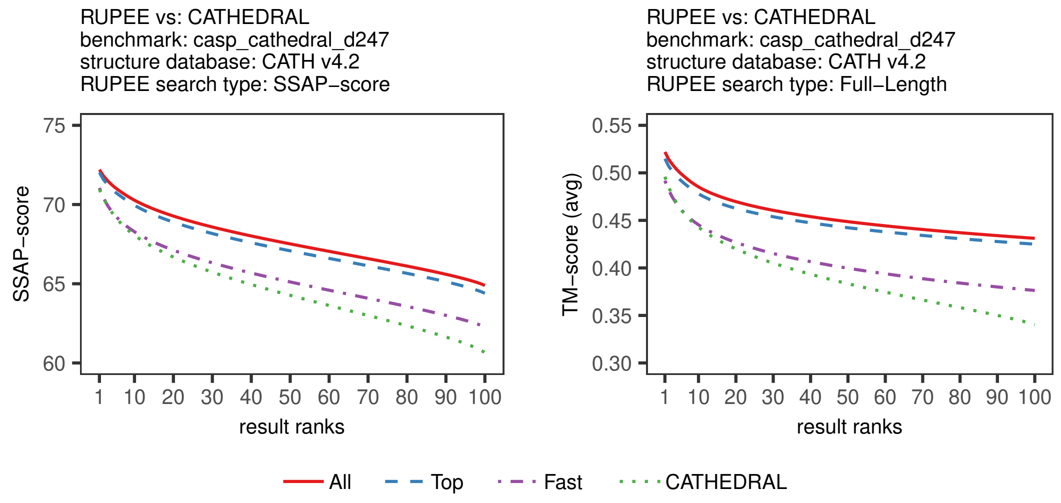


Figure 35: RUPEE vs CATHEDRAL. Comparison of average SSAP-scores (left) and TM-scores (right) for result ranks from 1 to 100 across the casp_cathedral_d247 benchmark

used TM-align to calculate the TM-scores, normalized by the average length of the compared structures. To be fair, we sorted the top 100 results by the compared score for both comparisons.

As shown in Fig. 35, both RUPEE top-aligned and all-aligned performed better than CATHEDRAL at all ranks for SSAP-scores and TM-scores. Additionally, RUPEE fast search mode performed better than CATHEDRAL for both scores at all ranks except the top 10 results, where RUPEE fast performed roughly equal to CATHEDRAL. It is remarkable that RUPEE top-aligned and all-aligned performed better than CATHEDRAL using the SSAP-score native to CATHEDRAL.

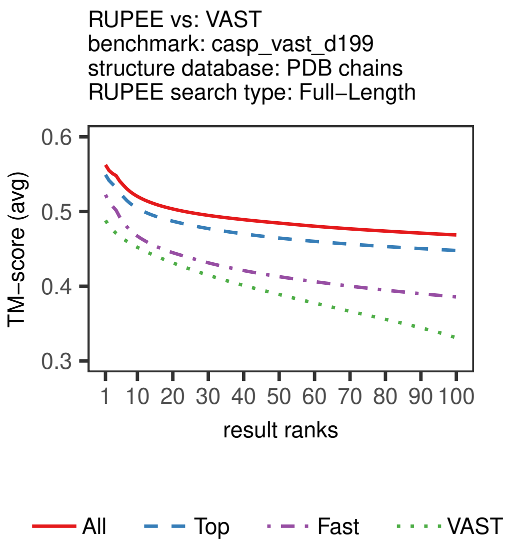


Figure 36: RUPEE vs VAST. Comparison of average TM-scores for result ranks from 1 to 100 across the casp_vast_d199 benchmark

5.2.2.4 Scoring vs. VAST

In Fig. 36, we compare the average TM-scores of the top 100 results for each rank across the casp_vast_d199 benchmark for RUPEE and VAST. We queried VAST by the VAST-score, a measure of full-length structure similarity [62] like the TM-score. However, we were not able to duplicate the VAST-score ourselves for our internal alignment scoring and so did not provide an additional search type as we did for the Q-score and the SSAP-score. Instead, we used TM-align to calculate the TM-scores, normalized by the average length of the compared structures. Therefore, in our comparison to VAST, we can only conclude that RUPEE is better than VAST with respect to TM-score and in so far as TM-score is a good measure of structure similarity, this comparison is useful. To be fair, we sorted the top 100 results from VAST by the calculated TM-score.

For RUPEE, we searched whole PDB chains deposited in the PDB as of 2020-08-28 whereas VAST searched the molecular modeling database (MMDB), which augments familiar PDB structures with knowledge annotations. It was not clear what version of the PDB VAST searched. Despite possible differences in the structural databases searched, it was the case that all the VAST results for our benchmark structures did match structures contained in the PDB as of 2020-08-28.

As shown in Fig. 36, all RUPEE search modes performed better than VAST at all ranks for TM-scores.

5.2.2.5 Scoring vs. Exhaustive

In Fig. 37, we compare the average TM-scores of the top 100 results for each rank across two different benchmarks for RUPEE against the results of an exhaustive search. For both benchmarks, RUPEE searched the same structure database, the SCOP v2.07 domains, consisting of more than 250,000 structures. The easier benchmark, scop_d360, is the benchmark of known proteins structures that we used in our previous work on RUPEE [6]. The harder benchmark, casp_d250, is the benchmark of protein structure predictions from above. For scop_d360, the difference between RUPEE all-aligned and top-aligned is less than 0.01 TM-score points, whereas for casp_d250, all-aligned is 0.01 to 0.02 TM-score points better than top-aligned across all ranks except for the first 10.

Notably, for the scop_d360 benchmark, RUPEE top-aligned and all-aligned performed roughly equal to the exhaustive search. However, for the more difficult casp_d250 benchmark, the exhaustive search performed better than RUPEE all-aligned by roughly

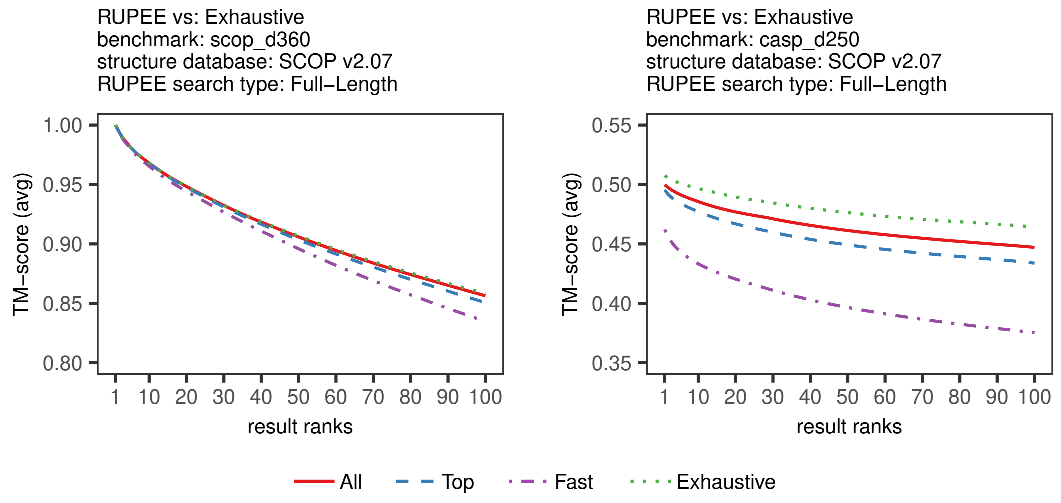


Figure 37: RUPEE vs Exhaustive. Comparison of average TM-scores for result ranks from 1 to 100 across the scop_d360 (left) and casp_d250 (right) benchmarks

the same amount all-aligned is an improvement over top-aligned.

The right of Fig. 37 suggest that RUPEE all-aligned is more suitable than RUPEE top-aligned for searching on protein structure predictions. However, for known protein structures, the performance of top-aligned is almost identical to all-aligned.

5.2.3 Response times

In Fig. 38, we compare the user response times of all the RUPEE search modes against mTM-align and SSM. We gathered these response times by automating the structure search web sites using the Selenium WebDriver API. Hence, our comparison of response times only considers the user experience rather than the computational cost of the respective structure searches. When automating web sites, we never issued more than one request at a time. The RUPEE web site runs on a single Amazon Web Services (AWS) c5.2xlarge elastic compute (EC2) unit. In Fig. 39, we compare the response times of all

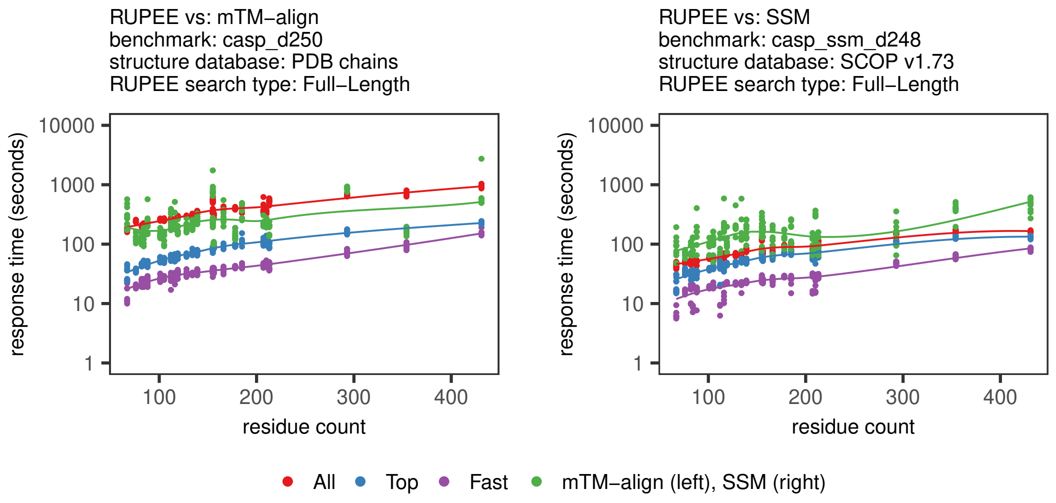


Figure 38: RUPEE vs mTM-align and SSM. Comparison of response times, in seconds, across the casp_d250 (left) and casp_ssm_d248 (right) benchmarks, respectively

the RUPEE search modes against the exhaustive search using both the scop_d360 and the casp_d250 benchmarks. We gathered these response times from RUPEE running on a desktop computer equipped with an AMD RYZEN 5 3600X 6-Core 3.8 GHz CPU with 12 hyperthreads of execution. We included all data points in the plots for completeness and we use a logarithmic scale and Loess regression curves to highlight the overall trends.

Even considering the difference in the number of structures between the PDB chains database and the SCOP v2.07 database, the difference in user response times for all-aligned in the left plot of Fig. 38 and all-aligned in the right plot of Fig. 39 are disproportionate. This difference in response times suggests the RUPEE web site response times can be significantly improved with a modest increase in computing resources.

On the left side of Fig. 38, we see that response times for mTM-align are a little

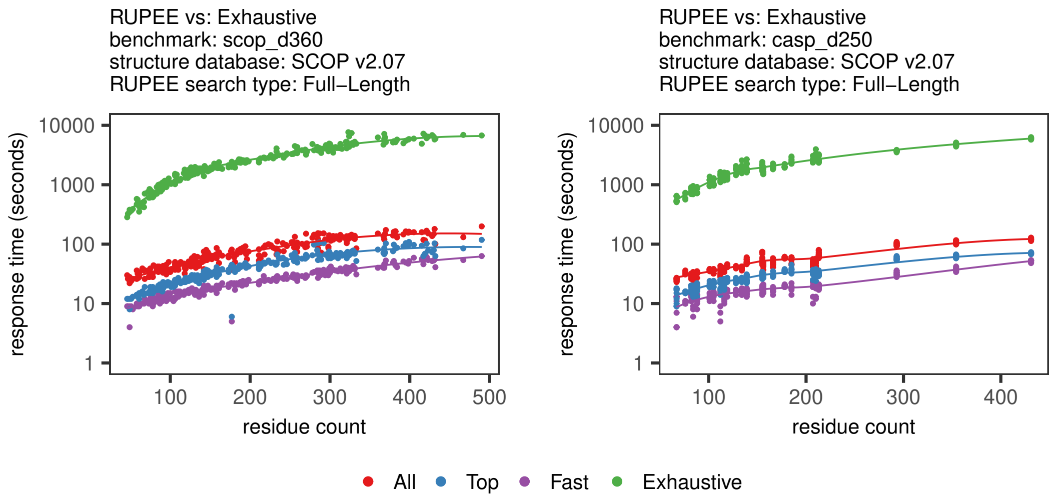


Figure 39: RUPEE vs Exhaustive. Comparison of response times, in seconds, across the scop_d360 (left) and casp_d250 (right) benchmarks

faster than response times for RUPEE all-aligned. However, as expected from our previous work on RUPEE [6], response times for fast and top-aligned are better than those for mTM-align. Surprisingly, response times for all the RUPEE search modes are faster than response times for SSM. Response times for RUPEE searching against the SCOP v1.73 domains are faster than for searching against PDB chains because there are fewer than 100,000 structures in SCOP v1.73.

Both plots in Fig. 39 are roughly the same, indicating that the benchmark difficulty does not affect response times for RUPEE. Not surprisingly, response times for the exhaustive search are roughly one order of magnitude greater for residue counts less than 300 and approaching two orders of magnitude greater for residue counts above 300.

Unfortunately, we were not able to collect response times for the CATHEDRAL and VAST structure searches. In the case of CATHEDRAL, we decided to collect response times after collecting the search results. When we returned to the CATHEDRAL search to collect response times, it was no longer processing requests and has remained in this state for several weeks. Nevertheless, response time comparisons with CATHEDRAL were completed in our previous work on RUPEE [6], and it was found that CATHEDRAL was the slowest among the structure searches we compared to. In the case of VAST, the search routinely returned with an error. However, the VAST request ids remained valid, so we ended up using automation to periodically check on the status of the request ids but were not able to collect exact times of completion. Additionally, we were not able to serialize our request to VAST because of this issue.

5.2.4 Sample Structure Alignments

Fig. 40 shows the structure alignments of the top-scoring full-length structures matches that RUPEE identified that none of the other protein structure searches we compared to identified. While we recognize that mTM-align, SSM, CATHEDRAL, and VAST may have performed equal to or better than RUPEE on some benchmark structures, RUPEE performs better on average as was shown in Figs. 33, 34, 35, and 36.

Fig. 40 illustrates qualitatively the types of difficult matches RUPEE has no problem with finding despite the complexity of the loops. We believe it is these types of difficult matches that would be of interest to researchers investigating protein structure prediction.

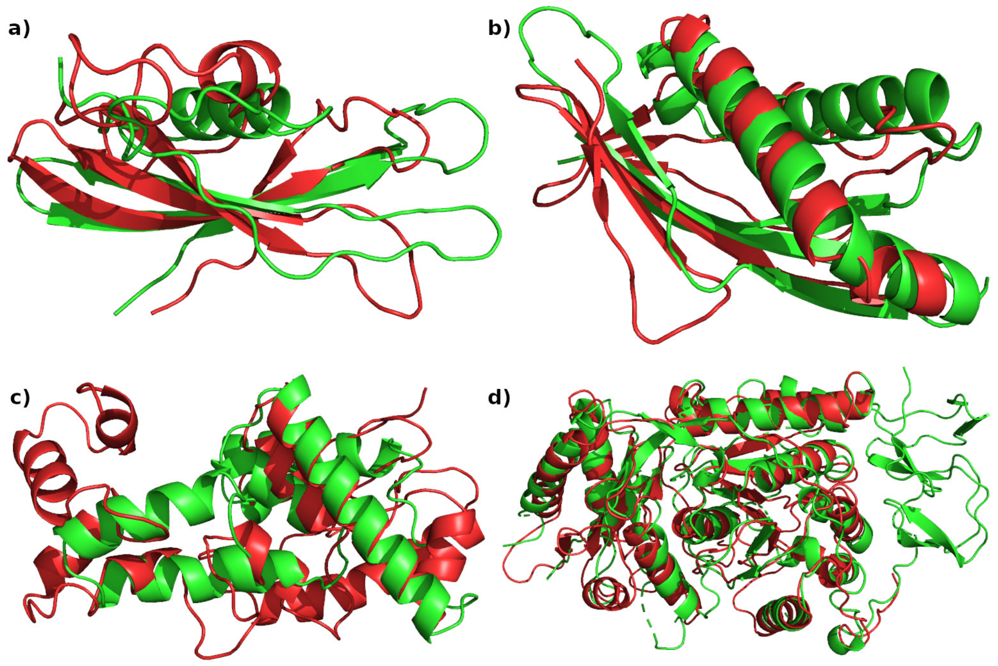


Figure 40: Red structures are CASP predictions. (a) T0960TS354-D2 aligned with 2lyxA00 found in CATH v4.2. (b) T0980s1TS043-D1 aligned with 6ahqC found among whole PDB chains. (c) T0990TS089-D3 aligned with 1w07A01 found in CATH v4.2. (d) T1000TS043-D2 aligned with 6u7lD found among whole PDB chains.

CHAPTER 6

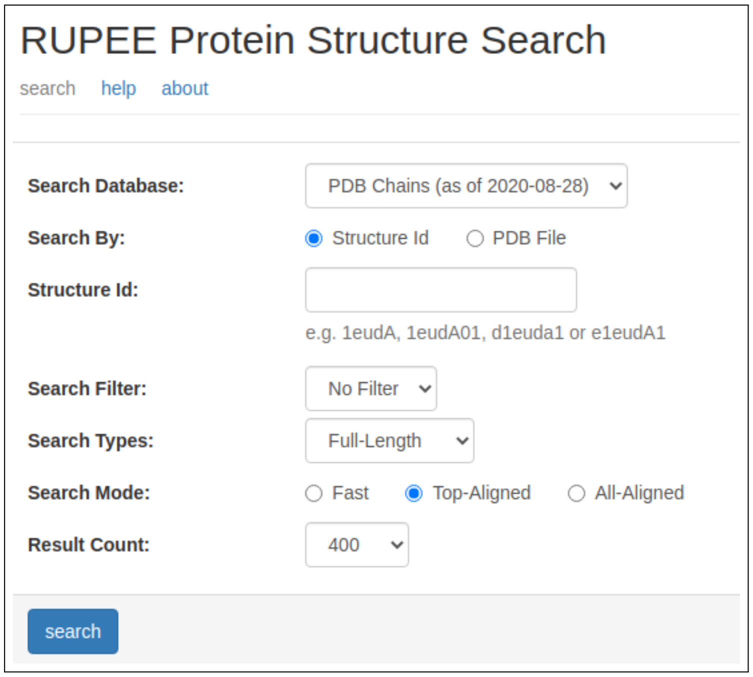
USING RUPEE

6.1 Using the RUPEE Web Site

Here we describe how to use the RUPEE protein structure search available at our web site `https://ayoubresearch.com`. First, we describe the search criteria. Then, we provide a walkthrough of an example search. In the next section, we will describe how to install and use RUPEE locally.

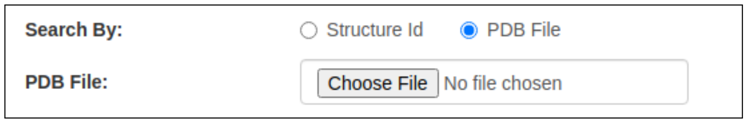
6.1.1 Search Criteria

Fig. 41 is a screenshot of the available search criteria. Fig. 42 shows the file upload element that appears in place of the *Structure Id* criterion when the *PDB File* option is selected.


RUPEE Protein Structure Search

Search Database:	PDB Chains (as of 2020-08-28)
Search By:	[✓ Structure Id] PDB File
Structure Id:	e.g. 1eudA, 1eudA01, d1euda1 or e1eudA1
Search Filter:	No Filter
Search Types:	Full-Length
Search Mode:	Fast [✓ Top-Aligned] All-Aligned
Result Count:	400

Figure 41: RUPEE search criteria



Search By:	Structure Id [✓ PDB File]
PDB File:	Choose File No file chosen

Figure 42: *PDB File* upload option

6.1.2 Search Database

The databases available for searching, along with their corresponding versions, are shown in Table 9.

Table 9: Databases available for searching with RUPEE

Database	**Version**
PDB Chain	as of 2020-08-28
SCOPe	2.07
CATH	v4.2
ECOD	develop240

6.1.3 Search By

You can search by a structure id or an uploaded PDB file.

Search By Structure Id

The structure id can be any SCOPe, CATH, or ECOD domain identifier or a PDB id and PDB chain id concatenated together for identifying whole chains within the PDB. The structure id does not have to match the database type you are searching — this is a feature of RUPEE that arose naturally when implementing the upload PBD file feature.

Search By PDB File

When uploading a PDB file, only the first chain of the first model is considered. Additionally, all backbone atoms (i.e. N, CA, C, and O) should be present for the search to be effective. If you want to find structures similar to a given domain, then upload the domain. If you want to find structures similar to a protein chain, then upload the chain and search the PDB Chain database.

6.1.4 Search Filter

The available search filters change dynamically based on the selected search database and search by criteria. SCOPe, CATH, and ECOD are hierarchical classifications. CATH designates a representative domain for each grouping at each level of the CATH hierarchy whereas SCOPe and ECOD do not (or at least we are not aware of this being the case). On the other hand, whole PDB chains are not classified into a hierarchy at all. Given the above, the SCOPe, CATH, and ECOD databases allow you to filter the search results by differences from the query structure for different hierarchy level classifications. In addition to filtering by different classifications, the CATH database allows you to filter by hierarchy level representatives. Currently, search filters are not provided for searching the PDB Chain database. Search filters allow for the discovery of structural similarities between differentially classified domains while preventing the results from being buried by known similarities.

6.1.5 Search Type

6 search types are available for searching. The first 3 are the recommended search types. Q-Score and SSAP-Score were provided for comparisons to SSM and CATHEDRAL, respectively, in our most recent paper [7] as described in Section 5.2. The RMSD score is provided for its ubiquity.

Full-Length

Return structures similar to the full-length of the query.

Contained In

Return structures that contain the query structure.

Contains

Return structures similar to a fragment of the query structure.

RMSD

Search on RMSD. Ubiquitous and generally useful.

Q-Score

Search on an SSM-specific score for RUPEE comparisons to SSM.

SSAP-Score

Search on a CATHEDRAL-specific score for RUPEE comparisons to CATHEDRAL.

To avoid too many trivial matches, the *Contains* search type is limited to structures no less than one third the size of the query structure. Containment searches will also return many structures similar to the full-length of the query structure since that is trivial containment.

6.1.6 Search Mode

Fast

For *Fast* mode, the top-scoring 40000 structures are obtained from the initial filtering with min-hashing and LSH. Then, a simplified Needleman-Wunsch residue descriptor sequence alignment between the query structure and the filtered structures is performed. Matches are awarded 1 point and mismatches and gaps are penalized 1 point. Finally, full TM-align structure alignments are performed on the top-scoring 400 structures and the results are returned sorted based on the *Search Type* criteria.

Top-Aligned

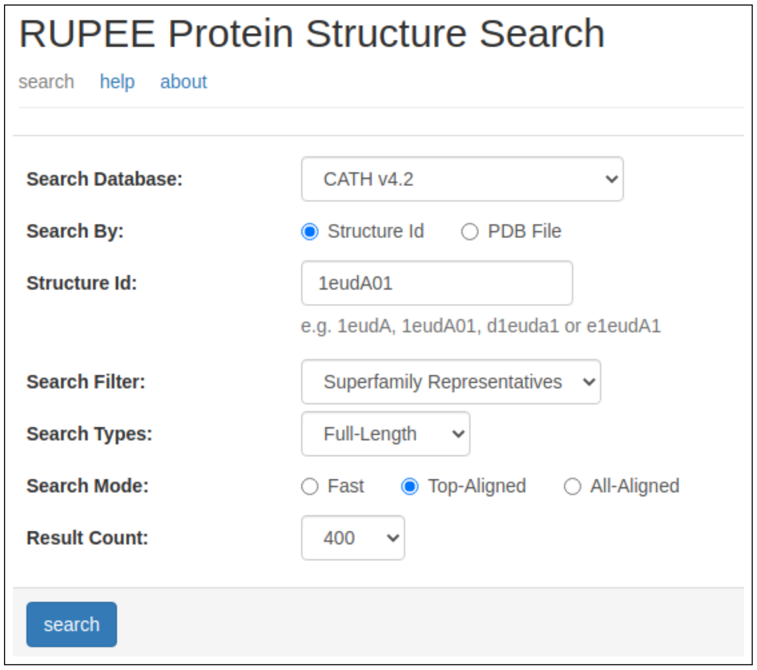
For *Top-Aligned* mode, the top-scoring 40000 structures are obtained from the initial filtering with min-hashing and LSH. Then, a simplified Needleman-Wunsch residue descriptor sequence alignment between the query structure and the filtered structures is performed. Matches are awarded 1 point and mismatches and gaps are penalized 1 point. Following the NW sequence alignments, the top-scoring 8000 structures are obtained and TM-align structure alignments are performed using a reduced number of iterations. Finally, full TM-align structure alignments are performed on the top-scoring 400 structures and the results are returned sorted based on the *Search Type* criteria.

All-Aligned

All-Aligned mode skips the min-hashing and LSH steps and instead, for each structure in the searched database, performs the simplified Needleman-Wunsch algorithm to obtain a residue alignment that is used as the initial alignment in a modified TM-align algorithm that does not attempt to find initial alignments by other means. This allows RUPEE to apply the modified TM-align to all available structures in a reasonable amount of time, typically between 5 and 10 minutes. Following the modified TM-align structure alignments, the top-scoring 8000 structures are obtained and normal TM-align structure alignments are performed using a reduced number of iterations. Finally, full TM-align structure alignments are performed on the top-scoring 400 structures and the results are returned sorted based on the *Search Type* criteria.

6.1.7 Example Search

Fig. 43 shows the criteria for an example full-length top-aligned search on the CATH domain 1eudA01 against the CATH v4.2 database filtered by superfamily representatives. Fig. 44 shows the top 10 scoring search results for the example search from Fig. 43. The left columns C, A, T, and H correspond to the CATH hierarchy designations Class, Architecture, Topology, and Homologous superfamily, respectively, for the result domains. The far right column has links where the user can view the text alignments or the 3d alignments for the results. In addition, the user can download PDB files for the alignments for viewing in an external viewer.


RUPEE Protein Structure Search

search help about

Search Database: CATH v4.2

Search By: [✓ Structure Id] PDB File

Structure Id: 1eudA01

e.g. 1eudA, 1eudA01, d1euda1 or e1eudA1

Search Filter: Superfamily Representatives

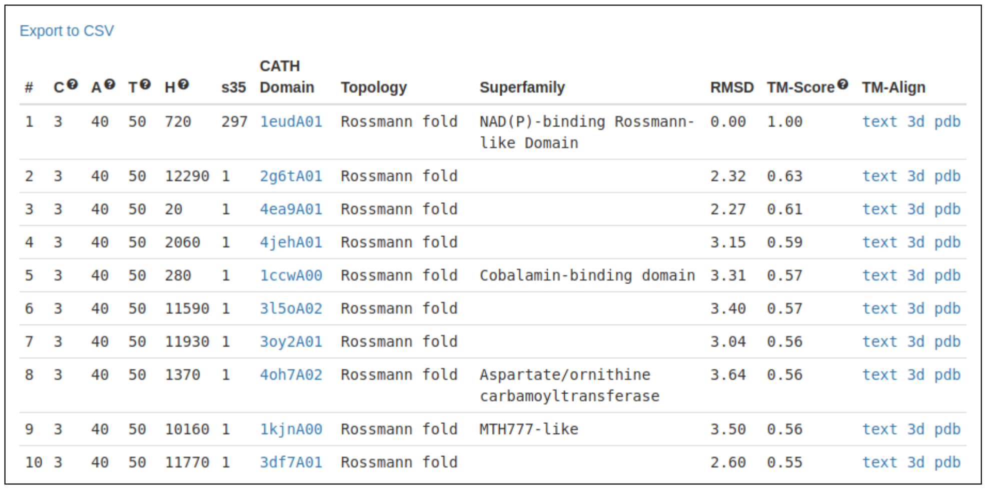
Search Types: Full-Length

Search Mode: Fast [✓ Top-Aligned] All-Aligned

Result Count: 400

search

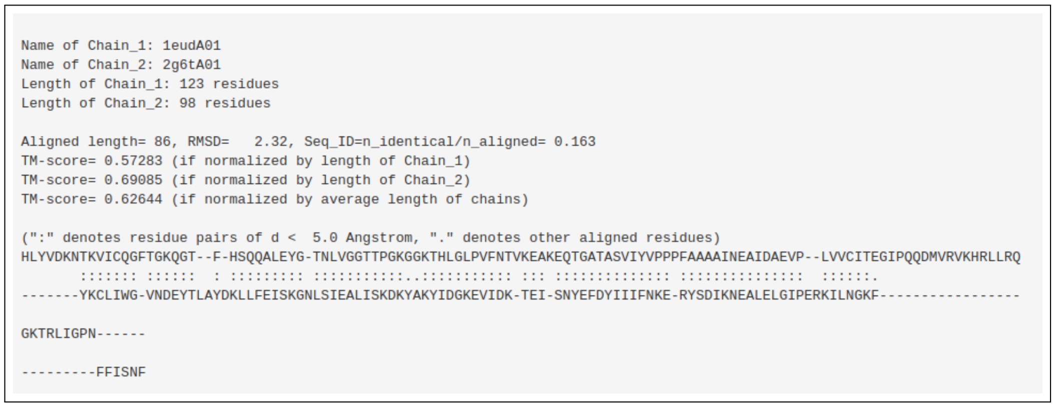
Figure 43: Example search using CATH domain 1eudA01 filtered by superfamily representatives


Export to CSV

#	C	A	T	H	s35	CATH Domain	Topology	Superfamily	RMSD	TM-Score	TM-Align
1	3	40	50	720	297	1eudA01	Rossmann fold	NAD(P)-binding Rossmann-like Domain	0.00	1.00	text 3d pdb
2	3	40	50	12290	1	2g6tA01	Rossmann fold		2.32	0.63	text 3d pdb
3	3	40	50	20	1	4ea9A01	Rossmann fold		2.27	0.61	text 3d pdb
4	3	40	50	2060	1	4jehA01	Rossmann fold		3.15	0.59	text 3d pdb
5	3	40	50	280	1	1ccwA00	Rossmann fold	Cobalamin-binding domain	3.31	0.57	text 3d pdb
6	3	40	50	11590	1	3l5oA02	Rossmann fold		3.40	0.57	text 3d pdb
7	3	40	50	11930	1	3oy2A01	Rossmann fold		3.04	0.56	text 3d pdb
8	3	40	50	1370	1	4oh7A02	Rossmann fold	Aspartate/ornithine carbamoyltransferase	3.64	0.56	text 3d pdb
9	3	40	50	10160	1	1kjnA00	Rossmann fold	MTH777-like	3.50	0.56	text 3d pdb
10	3	40	50	11770	1	3df7A01	Rossmann fold		2.60	0.55	text 3d pdb

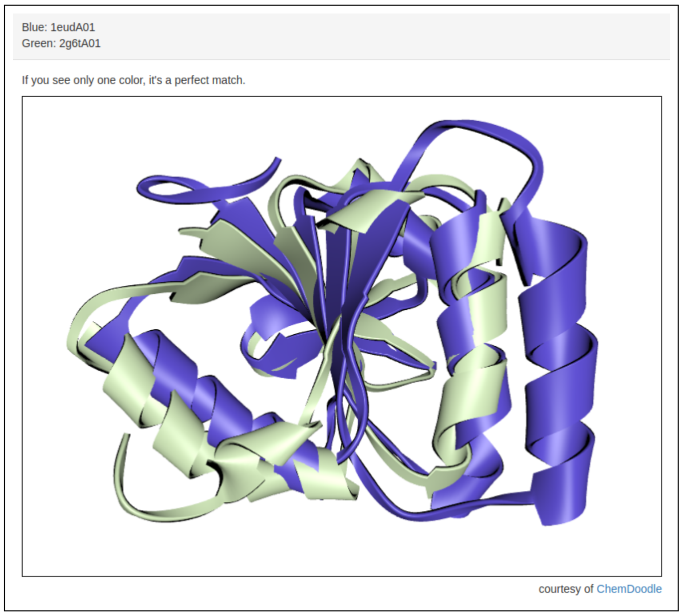
Figure 44: Top 10 scoring search results for CATH domain 1eudA01 filtered by superfamily representatives

Fig. 45 shows the text alignment for 1eudA01 and 2g6tA01 from the results in Fig. 44. The output for the text alignment is simply the exact output provided by the TM-align structure alignment tool [69] that is used internally for the final structure alignments. Fig. 46 shows the 3d alignment for 1eudA01 and 2g6tA01 that is produced by the same alignment file that can be downloaded separately by clicking the *pdb* link in the search results.



```
Name of Chain_1: 1eudA01
Name of Chain_2: 2g6tA01
Length of Chain_1: 123 residues
Length of Chain_2: 98 residues

Aligned length= 86, RMSD=   2.32, Seq_ID=n_identical/n_aligned= 0.163
TM-score= 0.57283 (if normalized by length of Chain_1)
TM-score= 0.69085 (if normalized by length of Chain_2)
TM-score= 0.62644 (if normalized by average length of chains)

(":" denotes residue pairs of d <  5.0 Angstrom, "." denotes other aligned residues)
HLYVDKNTKVICQGFTGKQGT--F-HSQQALEYG-TNLVGGTTPGKGGKTHLGLPVFNTVKEAKEQTGATASVIYVPPPFAAAAINEAIDAEVP--LVVCITEGIPQQDMVRVKHRLLRQ
       ::::::: ::::::  : ::::::::: :::::::::::..::::::::::: ::: :::::::::::::: :::::::::::::::  ::::::.
-------YKCLIWG-VNDEYTLAYDKLLFEISKGNLSIEALISKDKYAKYIDGKEVIDK-TEI-SNYEFDYIIIFNKE-RYSDIKNEALELGIPERKILNGKF----------------

GKTRLIGPN------

---------FFISNF
```

Figure 45: Text alignment of CATH domains 1eudA01 and 2g6tA01


Blue: 1eudA01
Green: 2g6tA01

If you see only one color, it's a perfect match.

courtesy of ChemDoodle

Figure 46: 3d alignment of CATH domains 1eudA01 and 2g6tA01

6.2 Installing and Using RUPEE Locally

Here we describe how to install the PostgreSQL, build RUPEE locally from the GitHub repository at `https://github.com/rayoub/rupee`, and run the RUPEE command-line interface (CLI) for administering and using the RUPEE protein structure search.

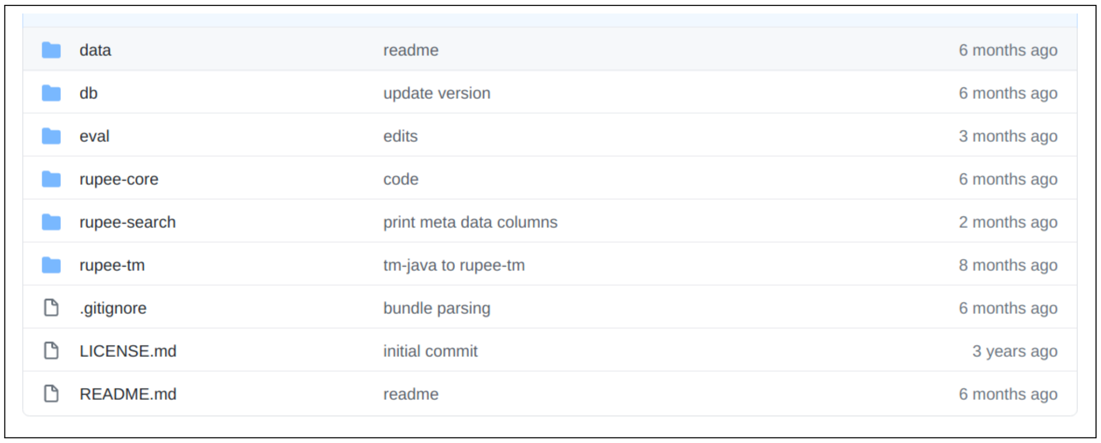
In addition to searching the PDB Chains, SCOPe, CATH, and ECOD databases like at the RUPEE web site at `https://ayoubresearch.com`, you also can search a local file system directory for convenience. For the `DB_TYPE` parameter defined below, the local file system directory `DB_TYPE` value is `DIR`. To keep things simple, we assume the local file system directory use-case until we get to Section 6.2.6 below, where we describe how to download and import the PDB Chains, SCOPe, CATH, and ECOD databases.

As for software dependencies, Java 8 and an installation of PostgreSQL 9.4 or above are required. The instructions below assume you are operating within a BASH shell under Ubuntu, the most popular version of Linux. However, all of the commands below can be run in other versions of Linux with minor alterations.

Fig. 47 shows the top-level directories of the RUPEE GitHub repository. When we refer to directories below, we are referring to the directories shown in Fig. 47.

6.2.1 Update Parameters

The first thing you should do is find the *Constants.java* file in the rupee-search Java project and update the `DB_NAME`, `DB_USER`, and `DB_PASSWORD` Java constants. The `DB_USER` should be the user name of the logged in Linux user that intends to run the



data	readme	6 months ago
db	update version	6 months ago
eval	edits	3 months ago
rupee-core	code	6 months ago
rupee-search	print meta data columns	2 months ago
rupee-tm	tm-java to rupee-tm	8 months ago
.gitignore	bundle parsing	6 months ago
LICENSE.md	initial commit	3 years ago
README.md	readme	6 months ago

Figure 47: RUPEE GitHub repository directories

RUPEE application. For everything below, it is assumed the database name is 'rupee' and the user name is 'ruser'.

6.2.2 Database Installation

At the command prompt, execute the following command to install PostgreSQL:

```
> sudo apt-get install postgresql
```

6.2.3 Database Creation

To manage PostgreSQL, you do not need the 'postgres' user password. Assuming you have root permissions, it is preferable to use `>sudo su` to become the 'postgres' user and then create your own user. First, create your PostgreSQL user with the following commands:

```
> sudo su postgres
postgres> createuser -s -P ruser
```

Next, create the PostgreSQL database and exit as the ‘postgres’ user:

```
postgres> createdb -O ruser rupee

postgres> exit
```

Now, locate the *pg_hba.conf* file in the PostgreSQL installation directories. Its location will vary. You should add the uncommented line below. The existing comments within the file are shown to provide context. This is a necessary step to access the database from the Java app using password authentication.

```
# TYPE  DATABASE    USER ADDRESS    METHOD

# password and auth for personal databases

local   RUPEE       ruser           md5
```

Reboot your computer to restart the PostgreSQL service with the new configuration. At this point, you should now get familiar with the PostgreSQL **psql** command line utility, which is the easiest way to manage a PostgreSQL database.

Navigate to the db/ directory and login to the rupee database by executing the following command:

```
> psql rupee
```

Finally, within the **psql** prompt, execute the following command:

```
\i y_create_all.sql
```

6.2.4 Maven Build

The simplest scenario is when you have a single directory of PDB files containing single chains. First, in the *Constants.java* file, edit the `DIR_PATH` constant to point to the local directory containing the PDB files you wish to search. In the same *Constants.java* file, edit the `DATA_PATH` constant to point to a local directory containing an upload/ subdirectory. Then, build the 3 Java projects in this order:

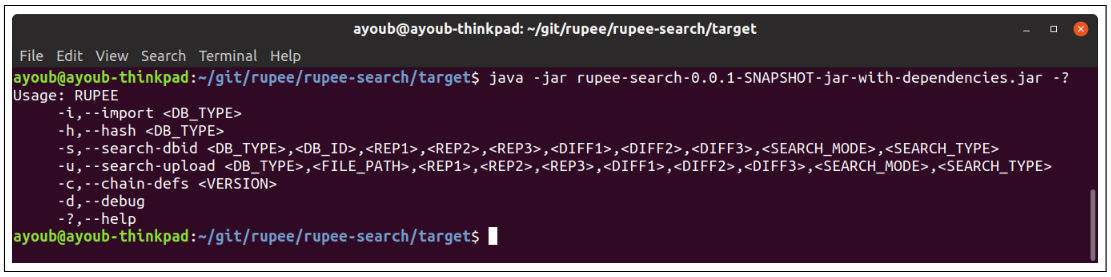
1. rupee-tm
2. rupee-core
3. rupee-search

To build, execute the following command from the root directory of each project:

```
> mvn clean package install
```

6.2.5 rupee-search Application

Once built, navigate to the rupee-search/target/ directory and issue the command shown in Fig. 48.


ayoub@ayoub-thinkpad: ~/git/rupee/rupee-search/target
File Edit View Search Terminal Help

```
ayoub@ayoub-thinkpad:~/git/rupee/rupee-search/target$ java -jar rupee-search-0.0.1-SNAPSHOT-jar-with-dependencies.jar -?
Usage: RUPEE
     -i,--import <DB_TYPE>
     -h,--hash <DB_TYPE>
     -s,--search-dbid <DB_TYPE>,<DB_ID>,<REP1>,<REP2>,<REP3>,<DIFF1>,<DIFF2>,<DIFF3>,<SEARCH_MODE>,<SEARCH_TYPE>
     -u,--search-upload <DB_TYPE>,<FILE_PATH>,<REP1>,<REP2>,<REP3>,<DIFF1>,<DIFF2>,<DIFF3>,<SEARCH_MODE>,<SEARCH_TYPE>
     -c,--chain-defs <VERSION>
     -d,--debug
     -?,--help
ayoub@ayoub-thinkpad:~/git/rupee/rupee-search/target$
```

Figure 48: RUPEE CLI Help

The following table briefly describes the options shown in Fig. 48.

Table 10: RUPEE command line options

Option	**Description**
-i	parse PDB files in the data directories and populate _grams tables
-h	min-hash grams in the _grams tables and populate the _hashes tables
-s	search for similar structures with a db id
-u	search for similar structures using a file path
-?	prints the available options

The following table provides a key for the optional parameters shown in Fig. 48.

Table 11: RUPEE command line option parameters

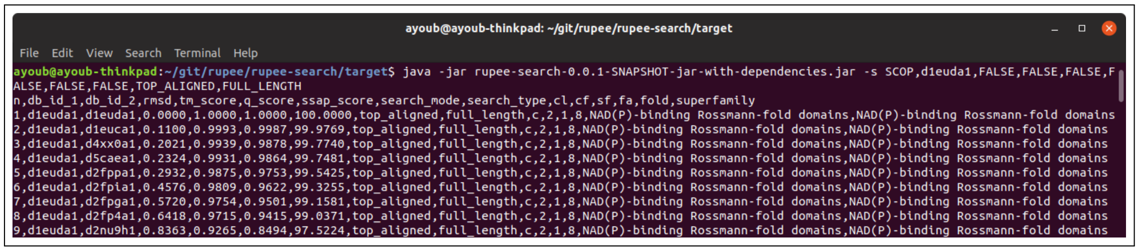
Parameter	**Definition**
⟨DB_TYPE⟩	DIR | SCOP | CATH | ECOD | CHAIN
⟨SEARCH_MODE⟩	FAST | TOP_ALIGNED | ALL_ALIGNED
⟨SEARCH_TYPE⟩	FULL_LENGTH | CONTAINED_IN | CONTAINS
⟨REP#⟩	TRUE | FALSE
⟨DIFF#⟩	TRUE | FALSE

To process the PDB files at `DIR_PATH`, execute the following commands:

```
> java -jar rupee.jar -i DIR

> java -jar rupee.jar -h DIR
```

Ignore the warnings, or alternatively, suppress the warnings using the java option `-Dlog4j.configurationFile=log4j2.xml`. The *log4j2.xml* file should be in the root of the target directory.

Once the data is done processing, you can now search. Fig. 49 shows the execution of an example search along with some of the top-scoring results. The first row of the results are the column headers. For searching a local directory, replace `SCOP` with `DIR` and use the file name as the `DB_ID`.


ayoub@ayoub-thinkpad: ~/git/rupee/rupee-search/target
File Edit View Search Terminal Help

```
ayoub@ayoub-thinkpad:~/git/rupee/rupee-search/target$ java -jar rupee-search-0.0.1-SNAPSHOT-jar-with-dependencies.jar -s SCOP,d1euda1,FALSE,FALSE,FALSE,F
ALSE,FALSE,FALSE,TOP_ALIGNED,FULL_LENGTH
n,db_id_1,db_id_2,rmsd,tm_score,q_score,ssap_score,search_mode,search_type,cl,cf,sf,fa,fold,superfamily
1,d1euda1,d1euda1,0.0000,1.0000,1.0000,100.0000,top_aligned,full_length,c,2,1,8,NAD(P)-binding Rossmann-fold domains,NAD(P)-binding Rossmann-fold domains
2,d1euda1,d1euca1,0.1100,0.9993,0.9987,99.9769,top_aligned,full_length,c,2,1,8,NAD(P)-binding Rossmann-fold domains,NAD(P)-binding Rossmann-fold domains
3,d1euda1,d4xx0a1,0.2021,0.9939,0.9878,99.7740,top_aligned,full_length,c,2,1,8,NAD(P)-binding Rossmann-fold domains,NAD(P)-binding Rossmann-fold domains
4,d1euda1,d5caea1,0.2324,0.9931,0.9864,99.7481,top_aligned,full_length,c,2,1,8,NAD(P)-binding Rossmann-fold domains,NAD(P)-binding Rossmann-fold domains
5,d1euda1,d2fppa1,0.2932,0.9875,0.9753,99.5425,top_aligned,full_length,c,2,1,8,NAD(P)-binding Rossmann-fold domains,NAD(P)-binding Rossmann-fold domains
6,d1euda1,d2fpia1,0.4576,0.9809,0.9622,99.3255,top_aligned,full_length,c,2,1,8,NAD(P)-binding Rossmann-fold domains,NAD(P)-binding Rossmann-fold domains
7,d1euda1,d2fpga1,0.5720,0.9754,0.9501,99.1581,top_aligned,full_length,c,2,1,8,NAD(P)-binding Rossmann-fold domains,NAD(P)-binding Rossmann-fold domains
8,d1euda1,d2fp4a1,0.6418,0.9715,0.9415,99.0371,top_aligned,full_length,c,2,1,8,NAD(P)-binding Rossmann-fold domains,NAD(P)-binding Rossmann-fold domains
9,d1euda1,d2nu9h1,0.8363,0.9265,0.8494,97.5224,top_aligned,full_length,c,2,1,8,NAD(P)-binding Rossmann-fold domains,NAD(P)-binding Rossmann-fold domains
```

Figure 49: RUPEE CLI example search of SCOPe using the domain d1euda1

6.2.6 Importing the CHAIN, SCOPe, CATH, and ECOD databases

Some files, especially data files, are too numerous or too large to include in the GitHub repository. The hidden *.gitignore* file list the files and directories that have been explicitly excluded from the repository.

6.2.7 db/

This directory contains SQL definition files. All files except files prefixed with x_ or y_ contain SQL definitions.

The files prefixed with x_ are used for populating tables and should only be run when parsed data files are present. Please note, the `x_` files contain **hard-coded references** to file locations that should be changed to match your Linux home directory. Unfortunately, the PostgreSQL `COPY` command does not accept relative directories.

6.2.8 data/

This directory contains all data files and scripts used in parsing the files.

The subdirectories shown in Table 12, along with brief descriptions, are excluded from the repository entirely so you must create them. These directories will hold the PDB files. You can create them now or create them as you go. Do not extract the downloaded archive files unless explicitly stated.

Table 12: RUPEE GitHub excluded data directories

Excluded Directory	**Description**
data/pdb/pdb/	From /pub/pdb/data/structures/all/pdb at ftp.wwpdb.org
data/pdb/obsolete/	From /pub/pdb/data/structures/obsolete/pdb at ftp.wwpdb.org
data/pdb/bundles/	From /pub/pdb/compatible/pdb_bundle at ftp.wwpdb.org
data/chain/pdb/	parsed pdb files containing whole chains
data/scop/pdb/	parsed pdb files based on scop definitions
data/cath/pdb/	parsed pdb files based on cath definitions
data/ecod/pdb/	parsed pdb files based on ecod definitions
data/upload/	directory used for temporary storage of uploaded pdb files

First, the data/pdb/ local directory has to be populated with files downloaded from the wwpdb FTP site. If using FileZilla, you should set the connection timeout to at least 1000 seconds in the **Edit-Settings** dialog. Click the data/pdb/ local directory to select the destination for the files. Click the /pub/pdb/data/structures/all/pdb/ remote directory containing the files you want to download. It will take a few minutes to obtain the directory listing and may even appear non-responsive. Once you have obtained the directory listing, right-click the pdb/ remote directory and select download. This also will create the pdb/ local directory under the data/pdb/ directory if it has not already been created.

If the data/pdb/obsolete/ local directory is not already created, then create it now. To populate the data/pdb/obsolete/ local directory, the actions are different from above because the remote files are organized into subdirectories. First, if using FileZilla, select the data/pdb/obsolete/ local directory and select the /pub/pdb/data/structures/obsolete/pdb/ remote directory. Then, go to the **Server-Search Remote Files** dialog. For search conditions, add a filename ends with *ent.gz* rule and click search. It should take about 10 minutes for the search to complete. Once the search is complete, in the **Search** dialog, select all files to be downloaded using Ctrl-A. Right-click and choose download. Choose to flatten remote paths and click OK.

If the data/pdb/bundles/ local directory is not already created, then create it now. Like the obsolete files, the remote files are organized into subdirectories. First, if using FileZilla, select the data/pdb/bundles/ local directory and select the /pub/pdb/compatible/pdb_bundle/ remote directory. Then, go to the **Server-Search Remote Files** dialog. Remove all search conditions that may already be present and click search. It should take about 10 minutes for the search to complete. Once the search is complete, in the **Search** dialog, select all files to be downloaded using Ctrl-A. Right-click and choose download. Choose to flatten remote paths and click OK.

The archive files in the /data/pdb/bundles/ local directory do have to be extracted.

To extract, while in the /data/pdb/bundles/ directory, execute the following commands:

```
> gunzip *.gz
> find . -name "*.tar" -exec tar xvf {} \;
```

Once downloaded, the files can be processed to populate the remaining data/pdb/ directory in addition to the data/chain/, data/scop/, data/cath/ and data/ecod/ directories. The data/pdb/ directory must be processed first followed by the data/chain/ directory. Then the remaining directories can be processed independently. The following is what to do in each directory with it set as your working directory.

6.2.9 data/pdb/

If you downloaded the PDB data on the date of MM/DD/YYYY, then your version is 'vMM_DD_YYYY' and needs to be passed as the argument to the *do_all.sh* bash script. Below is an example:

```
> ./do_all.sh v08_28_2020
```

This script chops bundle files and puts them in the data/pdb/chopped/ directory. Next, files are created in the data/pdb/chain/ directory that define the chains present in the PDB files, including bundle files and obsolete files. The passed in version is used to name these definition files.

6.2.10 data/chain/

Using the same version as above, type the following command:

```
> ./do_all.sh v08_28_2020
```

6.2.11 data/scop/, data/cath/, and data/ecod/

Simply type in the following command in each directory in any order:

```
> ./do_all.sh
```

6.2.12 Final Remarks

You may need to get more familiar with the *do_all.sh* scripts. When a script errors out midway through, carefully comment out the completed lines, address the issue, and run again. Sometimes a modified *do_all.sh* script is checked in with some lines commented out. Before an initial run make sure all lines are uncommented.

If you have successfully processed one of the data directories, you can now execute searches with the rupee-search application.

6.3 Case Studies

Protein structure plays a major role in modern medical research. The aggregation of amyloid-β proteins is a critical factor in the pathogenesis of Alzheimer's disease [29]. A mutation in the α-synuclein protein causes an error in folding, the process by which a protein assumes its 3-dimensional shape, and is believed to be a major factor in Parkinson's disease [52]. Another protein structure topic dominating popular discussion today is the current COVID-19 global pandemic, the disease caused by the infamous SARS CoV-2 spike protein.

In this section, we will demonstrate how the search capabilities of RUPEE can be used to investigate some of the protein structures central to Alzheimer's disease, Parkinson's disease, and the COVID-19 global pandemic. While we do not expect the findings we present here to be groundbreaking discoveries, we do believe that the search techniques we present will provide adequate guidance to investigators on how to make their own discoveries using the RUPEE protein structure search as an invaluable tool.

6.3.1 Alzheimer's Disease

Alzheimer's disease (AD) is characterized by the presence of plaques between and tangles within the neurons of the brain. The plaques are formed from aggregates of misfolded amyloid-β proteins that result from incorrectly cleaved fragments of a transmembrane protein known as the amyloid precursor protein (APP). These amyloid-β proteins first accumulate into a few stacked β-sheets and then into longer fibers of stacked β-sheets, which then combine to form the amyloid plaques. The tangles are formed from misfolded tau proteins within the neurons that then form fibrils and eventually aggregate in a less orderly manner into tangles rather than fibers and plaques. In both cases of plaques and tangles, misfolded proteins aggregate into deposits within the brain that interfere with the normal function of the brain, contributing to the symptoms of dementia that are associated with AD.

Fig. 50 shows SCOP domain d3nyla_, the APP E2 domain, aligned with SCOP domain d3fzfb1, a Spectrin repeat-like domain within a family of molecular chaperones. We found this match by running a full-length search on the d3nyla_ domain against the

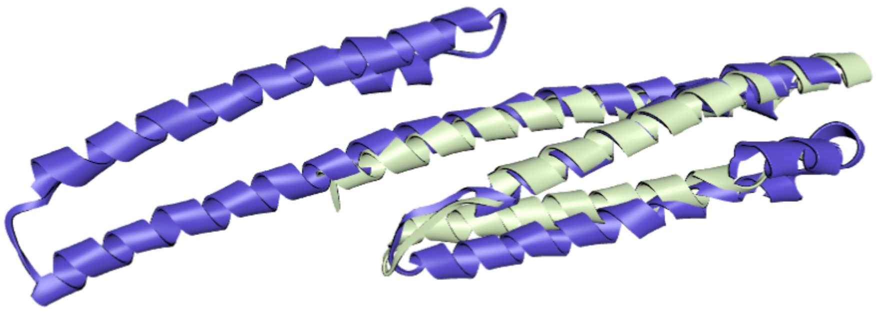
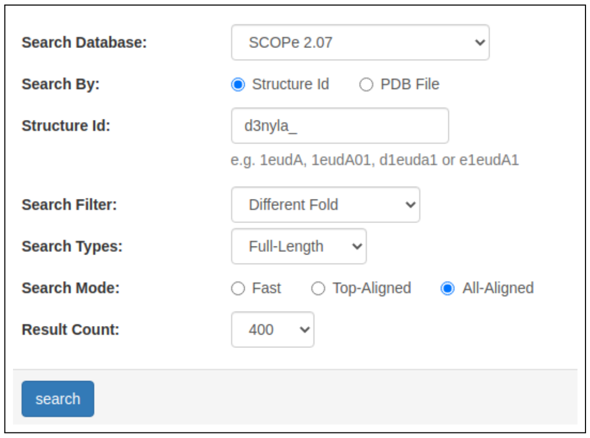


Figure 50: SCOP domain d3nyla_ (blue) aligned with SCOP domain d3fzfb1 (green)

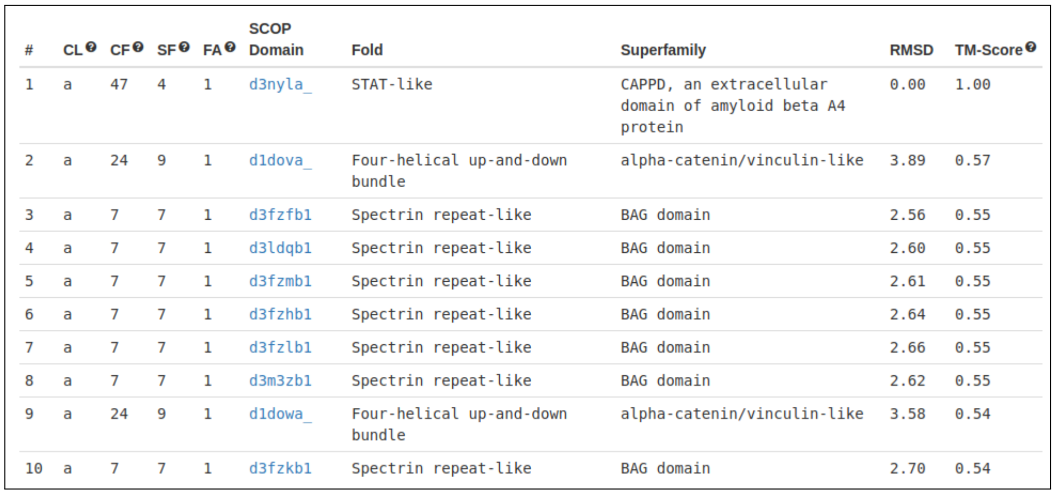


Search Database:	SCOPe 2.07
Search By:	[✓ Structure Id] PDB File
Structure Id:	d3nyla_
	e.g. 1eudA, 1eudA01, d1euda1 or e1eudA1
Search Filter:	Different Fold
Search Types:	Full-Length
Search Mode:	Fast Top-Aligned [✓ All-Aligned]
Result Count:	400

Figure 51: Search criteria used to find the d3fzfb1 match shown in Fig. 50

SCOP database in all-aligned search mode using the *Different Fold* filter. When normalized by the length of the d3fzfb1 domain, the TM-score is quite high at 0.721. However, as a full-length match, the TM-score is only 0.555. Although the resemblance of the APP E2 domain to Spectrin repeat-like domains has already been noted as a surprising similarity [63], it appeared among the top-scoring results for our search criteria.

Fig. 51 shows the criteria used to find the d3fzfb1 match shown in Fig. 50. Fig. 52 shows the top-10 results returned for the criteria shown in Fig. 51.



#	CL	CF	SF	FA	SCOP Domain	Fold	Superfamily	RMSD	TM-Score
1	a	47	4	1	d3nyla_	STAT-like	CAPPD, an extracellular domain of amyloid beta A4 protein	0.00	1.00
2	a	24	9	1	d1dova_	Four-helical up-and-down bundle	alpha-catenin/vinculin-like	3.89	0.57
3	a	7	7	1	d3fzfb1	Spectrin repeat-like	BAG domain	2.56	0.55
4	a	7	7	1	d3ldqb1	Spectrin repeat-like	BAG domain	2.60	0.55
5	a	7	7	1	d3fzmb1	Spectrin repeat-like	BAG domain	2.61	0.55
6	a	7	7	1	d3fzhb1	Spectrin repeat-like	BAG domain	2.64	0.55
7	a	7	7	1	d3fzlb1	Spectrin repeat-like	BAG domain	2.66	0.55
8	a	7	7	1	d3m3zb1	Spectrin repeat-like	BAG domain	2.62	0.55
9	a	24	9	1	d1dowa_	Four-helical up-and-down bundle	alpha-catenin/vinculin-like	3.58	0.54
10	a	7	7	1	d3fzkb1	Spectrin repeat-like	BAG domain	2.70	0.54

Figure 52: Search results returned for the criteria shown in Fig. 51. d3fzfb1 is #3

A more surprising similarity that we have not found in a previously published paper is shown in Fig. 53, which shows SCOP domain d3nyla_, the APP E2 domain, aligned with SCOP domain d5vxka_, the Invasin ipaD domain from the bacteria *Shigella flexneri*. The Invasin ipaD domain is used for host cell pore creation and penetration. We found this match by running a contained-in search on the d3nyla_ domain against the SCOP database in all-aligned search mode using the *Different Fold* filter. Once again, this match appeared among the top-scoring results for our search criteria. In this case, the TM-score as a full-length match is 0.631, even higher than the match shown in Fig. 50.

Fig. 54 shows the criteria used to find the d5vxka_ match shown in Fig. 53. Fig. 55 shows the top-10 results returned for the criteria shown in Fig. 54.

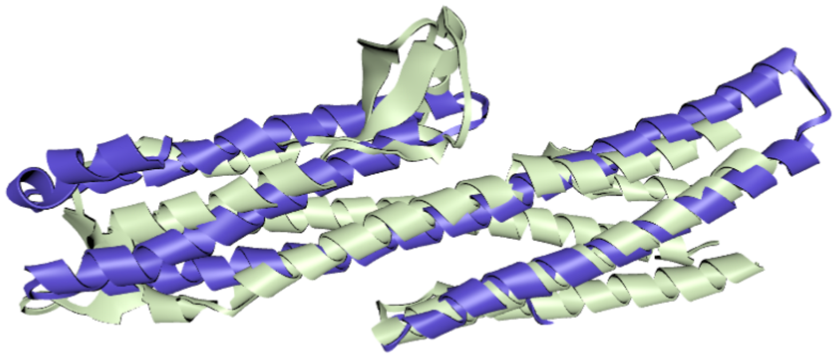
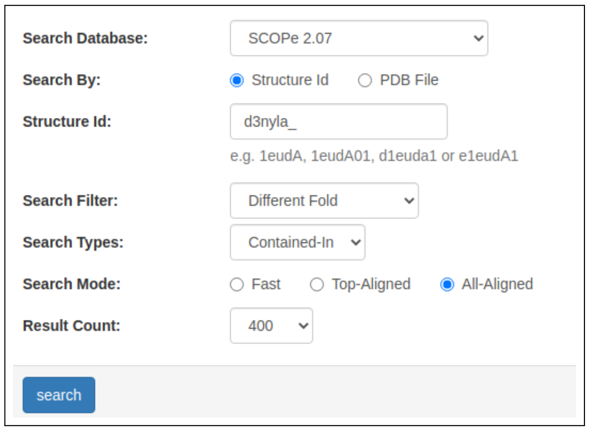
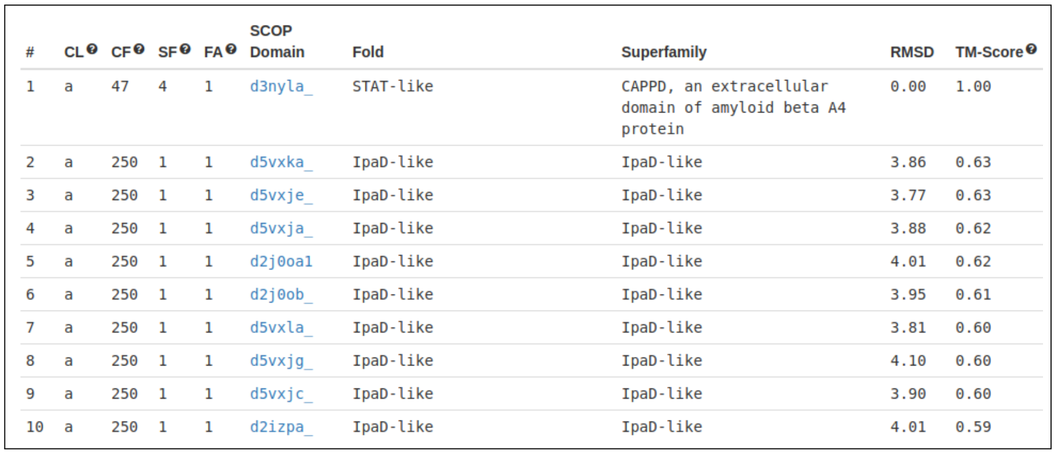


Figure 53: SCOP domain d3nyla_ (blue) aligned with SCOP domain d5vxka_ (green)



Search Database:	SCOPe 2.07
Search By:	[✓ Structure Id] PDB File
Structure Id:	d3nyla_ e.g. 1eudA, 1eudA01, d1euda1 or e1eudA1
Search Filter:	Different Fold
Search Types:	Contained-In
Search Mode:	Fast Top-Aligned [✓ All-Aligned]
Result Count:	400

Figure 54: Search criteria used to find the d5vxka_ match shown in Fig. 53



#	CL	CF	SF	FA	SCOP Domain	Fold	Superfamily	RMSD	TM-Score
1	a	47	4	1	d3nyla_	STAT-like	CAPPD, an extracellular domain of amyloid beta A4 protein	0.00	1.00
2	a	250	1	1	d5vxka_	IpaD-like	IpaD-like	3.86	0.63
3	a	250	1	1	d5vxje_	IpaD-like	IpaD-like	3.77	0.63
4	a	250	1	1	d5vxja_	IpaD-like	IpaD-like	3.88	0.62
5	a	250	1	1	d2j0oa1	IpaD-like	IpaD-like	4.01	0.62
6	a	250	1	1	d2j0ob_	IpaD-like	IpaD-like	3.95	0.61
7	a	250	1	1	d5vxla_	IpaD-like	IpaD-like	3.81	0.60
8	a	250	1	1	d5vxjg_	IpaD-like	IpaD-like	4.10	0.60
9	a	250	1	1	d5vxjc_	IpaD-like	IpaD-like	3.90	0.60
10	a	250	1	1	d2izpa_	IpaD-like	IpaD-like	4.01	0.59

Figure 55: Search results returned for the criteria shown in Fig. 54. d5vxka_ is #2

6.3.2 Parkinson's Disease

Parkinson's disease (PD) is characterized by the presence of Lewy bodies, aggregates of misfolded α-synuclein proteins, within neurons of the brain. The presence of Lewy bodies also characterizes another form of neurodegenerative disease known as Lewy Body Dementia (LBD). Lewy bodies are believed to interfere with dopamine receptors and acetylcholine production. Dopamine regulates everything from movement and fine motor control to mood and sleep. Acetylcholine is critical to learning and memory function. Lewy body interference with dopamine regulation and acetylcholine production is believed to be responsible for the common symptoms associated with PD.

Fig. 56 shows SCOP domain d1xq8a_, a *Human* α-Synuclein protein, aligned with SCOP domain d1l6lc_, a *Human* Apolipoprotein, a lipid binding protein involved in the transportation of lipids. We found this match by running a full-length search on the d1xq8a_ domain against the SCOP database in all-aligned search mode. When normalized by the length of the d1l6lc_ domain, the TM-score is notably 0.567. However, as a full-length match the TM-score is only 0.439. Fig. 56 highlights the highly conserved α-helix lipid-binding motif shared by both proteins that have been previously noted in a review of synucleins [27].

Fig. 57 shows the criteria used to find the d1l6lc_ match shown in Fig. 56. Fig. 58 shows the top-10 results returned for the criteria shown in Fig. 57.

A more surprising similarity that we have not found in a previously published paper is shown in Fig. 59, which shows SCOP domain d1xq8a_, a *Human* α-Synuclein protein, aligned with ECOD domain e4wsnl3, the C-terminal domain of a proteasome

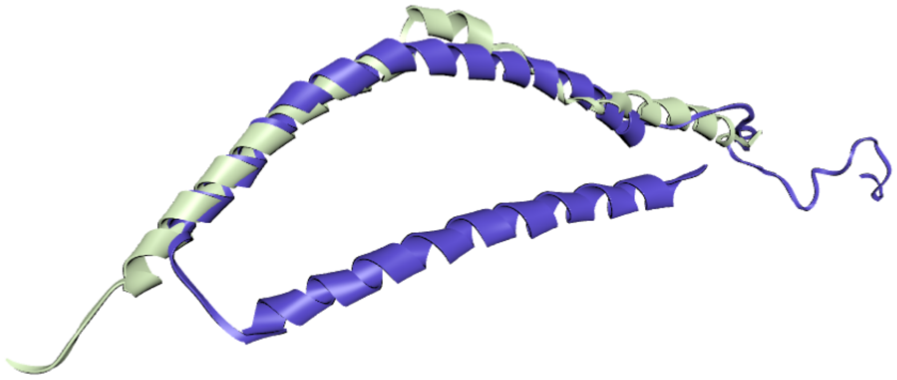
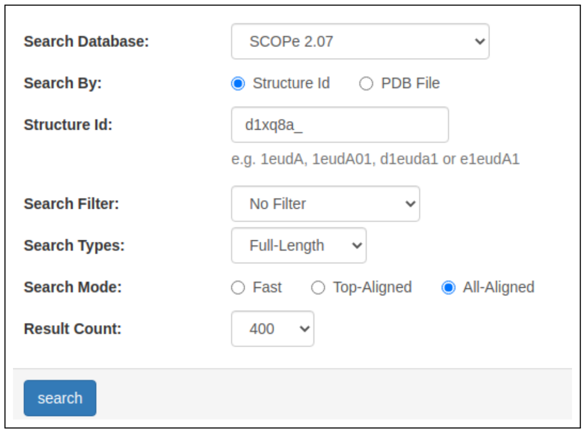
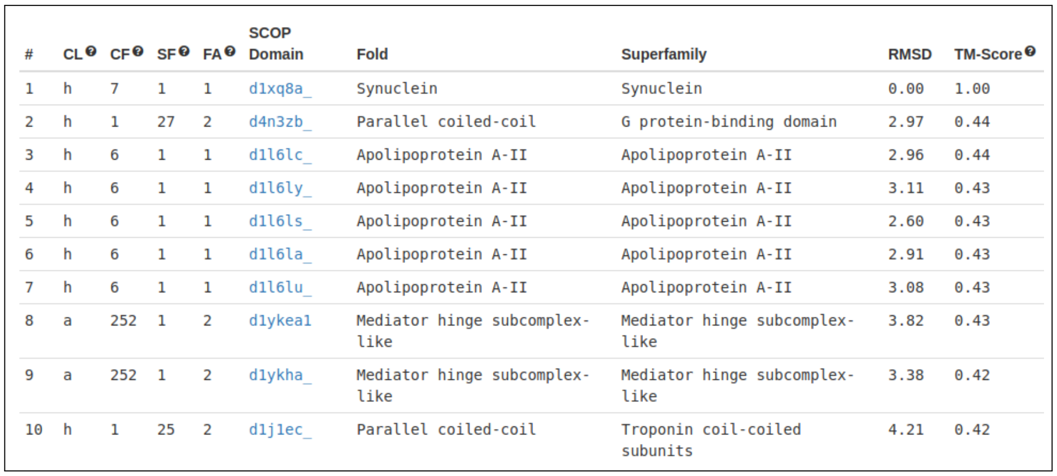


Figure 56: SCOP domain d1xq8a_ (blue) aligned with SCOP domain d1l6lc_ (green)



Search Database:	SCOPe 2.07
Search By:	[✓ Structure Id] PDB File
Structure Id:	d1xq8a_
	e.g. 1eudA, 1eudA01, d1euda1 or e1eudA1
Search Filter:	No Filter
Search Types:	Full-Length
Search Mode:	Fast Top-Aligned [✓ All-Aligned]
Result Count:	400

Figure 57: Search criteria used to find the d1l6lc_ match shown in Fig. 56



#	CL	CF	SF	FA	SCOP Domain	Fold	Superfamily	RMSD	TM-Score
1	h	7	1	1	d1xq8a_	Synuclein	Synuclein	0.00	1.00
2	h	1	27	2	d4n3zb_	Parallel coiled-coil	G protein-binding domain	2.97	0.44
3	h	6	1	1	d1l6lc_	Apolipoprotein A-II	Apolipoprotein A-II	2.96	0.44
4	h	6	1	1	d1l6ly_	Apolipoprotein A-II	Apolipoprotein A-II	3.11	0.43
5	h	6	1	1	d1l6ls_	Apolipoprotein A-II	Apolipoprotein A-II	2.60	0.43
6	h	6	1	1	d1l6la_	Apolipoprotein A-II	Apolipoprotein A-II	2.91	0.43
7	h	6	1	1	d1l6lu_	Apolipoprotein A-II	Apolipoprotein A-II	3.08	0.43
8	a	252	1	2	d1ykea1	Mediator hinge subcomplex-like	Mediator hinge subcomplex-like	3.82	0.43
9	a	252	1	2	d1ykha_	Mediator hinge subcomplex-like	Mediator hinge subcomplex-like	3.38	0.42
10	h	1	25	2	d1j1ec_	Parallel coiled-coil	Troponin coil-coiled subunits	4.21	0.42

Figure 58: Search results returned for the criteria shown in Fig. 57. d1l6lc_ is #3

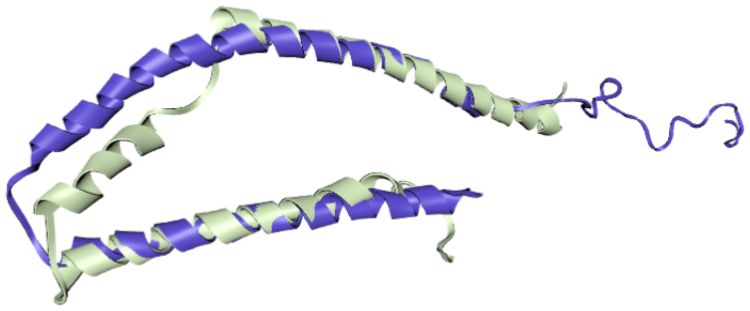
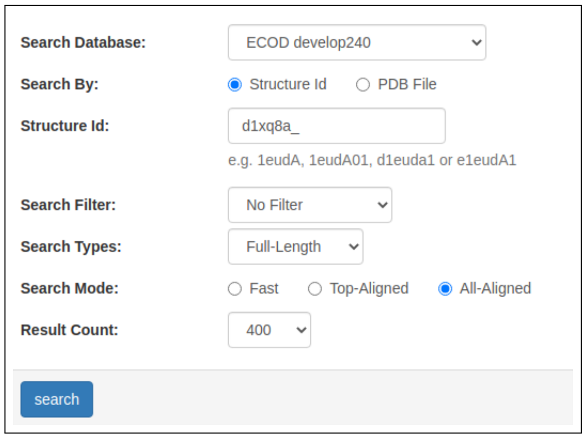


Figure 59: SCOP domain d1xq8a_ (blue) aligned with ECOD domain e4wsnl3 (green)

regulatory protein involved in the breakdown of proteins marked for destruction. We found this match by running a full-length search on the d1xq8a_ domain against the ECOD database in all-aligned search mode. In this case, the TM-score as a full-length match is 0.496, higher than the match shown in Fig. 56. We were only able to find this match by using a cross-database search with an entry from the SCOP database against the ECOD database, a useful and convenient feature of RUPEE.

Fig. 60 shows the criteria used to find the e4wsnl3 match shown in Fig. 59. Fig. 61


Search Database: ECOD develop240

Search By: [✓ Structure Id] PDB File

Structure Id: d1xq8a_

e.g. 1eudA, 1eudA01, d1euda1 or e1eudA1

Search Filter: No Filter

Search Types: Full-Length

Search Mode: Fast Top-Aligned [✓ All-Aligned]

Result Count: 400

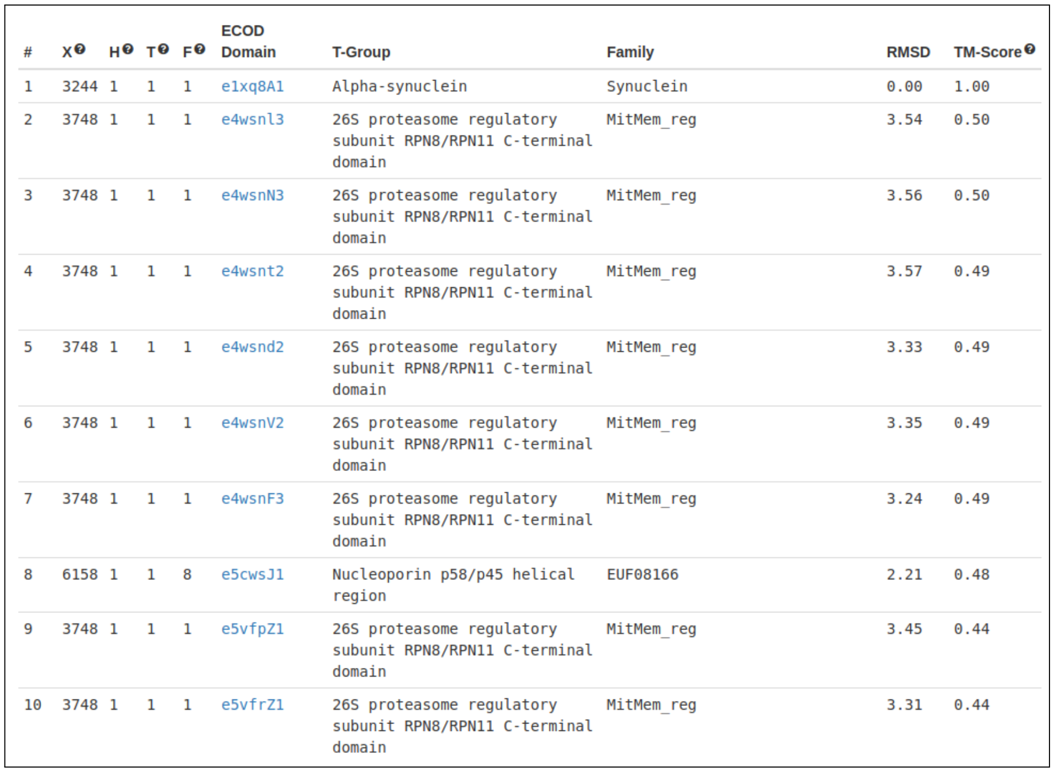
Figure 60: Search criteria used to find the e4wsnl3 match shown in Fig. 59

shows the top-10 results returned for the criteria shown in Fig. 60.

6.3.3 SARS

Severe Acute Respiratory Syndrome (SARS) is the condition associated with a class of viruses known as coronaviruses. In 2002, a SARS outbreak began in China and spread to 4 other countries. The coronavirus associated with this first outbreak of SARS was called SARS CoV. In 2019, another SARS outbreak occurred in China and quickly spread throughout the world, becoming a global pandemic. The coronavirus associated with this second outbreak of SARS was called SARS CoV-2 and the associated disease has been given the name COVID-19.

As shown to the left of Fig. 62, the most prominent feature of the SARS CoV and SARS CoV-2 viruses are the now infamous spike proteins spread across the surface of the virion particles. The spike proteins are responsible for both binding to cellular receptors and fusing the virion particle with the cell for the injection of the genetic material. To the right of Fig. 62, a ribbon representation of a spike protein is shown binding to the ACE2



#	X	H	T	F	ECOD Domain	T-Group	Family	RMSD	TM-Score
1	3244	1	1	1	e1xq8A1	Alpha-synuclein	Synuclein	0.00	1.00
2	3748	1	1	1	e4wsnl3	26S proteasome regulatory subunit RPN8/RPN11 C-terminal domain	MitMem_reg	3.54	0.50
3	3748	1	1	1	e4wsnN3	26S proteasome regulatory subunit RPN8/RPN11 C-terminal domain	MitMem_reg	3.56	0.50
4	3748	1	1	1	e4wsnt2	26S proteasome regulatory subunit RPN8/RPN11 C-terminal domain	MitMem_reg	3.57	0.49
5	3748	1	1	1	e4wsnd2	26S proteasome regulatory subunit RPN8/RPN11 C-terminal domain	MitMem_reg	3.33	0.49
6	3748	1	1	1	e4wsnV2	26S proteasome regulatory subunit RPN8/RPN11 C-terminal domain	MitMem_reg	3.35	0.49
7	3748	1	1	1	e4wsnF3	26S proteasome regulatory subunit RPN8/RPN11 C-terminal domain	MitMem_reg	3.24	0.49
8	6158	1	1	8	e5cwsJ1	Nucleoporin p58/p45 helical region	EUF08166	2.21	0.48
9	3748	1	1	1	e5vfpZ1	26S proteasome regulatory subunit RPN8/RPN11 C-terminal domain	MitMem_reg	3.45	0.44
10	3748	1	1	1	e5vfrZ1	26S proteasome regulatory subunit RPN8/RPN11 C-terminal domain	MitMem_reg	3.31	0.44

Figure 61: Search results returned for the criteria shown in Fig. 60. e4wsnl3 is #2

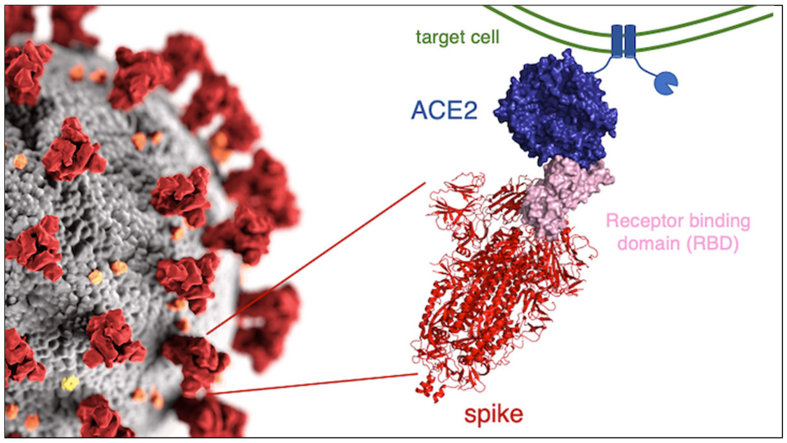


Figure 62: SARS virion particle and associated spike proteins (Photo Source: Tyler Starr/Bloom Lab and Alissa Eckert/MSMI; Dan Higgins/MAMS)

cellular receptor at the site of the receptor binding domain at the end of the spike protein.

Fig. 63 shows PDB chain 6m17E, the receptor binding domain for SARS CoV-2, aligned with PDB chain 3bgfS, the receptor binding domain for SAR CoV. We found this match by running a full-length search on the 6m17E PDB chain against the PDB chain database in all-aligned search mode. This match is not a surprise and only serves to highlight the strong similarity between SARS CoV and SARS CoV-2 spike protein receptor binding domains. As a full-length match, the TM-score is high at 0.928.

Fig. 64 shows the criteria used to find the 3bgfS match shown in Fig. 63. Fig. 65 shows the top-10 results returned for the criteria shown in Fig. 64.

On the other hand, a more surprising match is shown in Fig. 66, which shows PDB chain 6m17E, the receptor binding domain for SARS CoV-2, aligned with SCOP domain d2dehf_, a Dodecin-like lumichrome binding domain-like protein subunit from the *Thermus thermophilus* bacteria. We found this match by running a contains search on

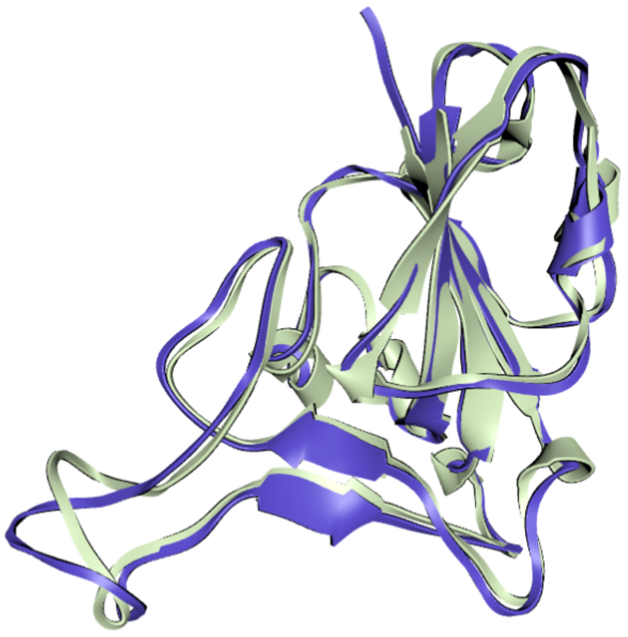
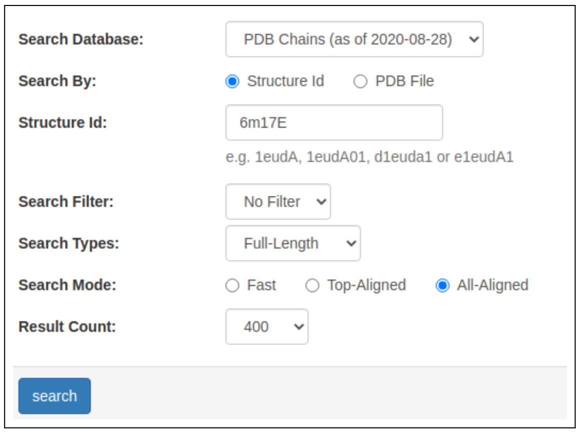
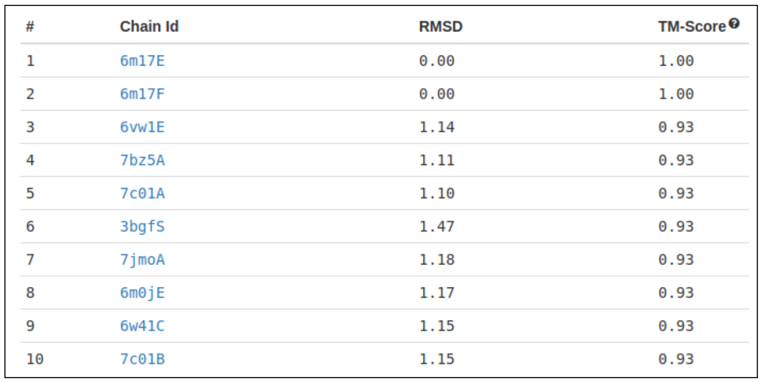


Figure 63: PDB chain 6m17E (blue) aligned with PDB chain 3bgfS (green)



Search Database:	PDB Chains (as of 2020-08-28)
Search By:	[✓ Structure Id] PDB File
Structure Id:	6m17E
	e.g. 1eudA, 1eudA01, d1euda1 or e1eudA1
Search Filter:	No Filter
Search Types:	Full-Length
Search Mode:	Fast Top-Aligned [✓ All-Aligned]
Result Count:	400

Figure 64: Search criteria used to find the 3bgfS match shown in Fig. 63



#	Chain Id	RMSD	TM-Score
1	6m17E	0.00	1.00
2	6m17F	0.00	1.00
3	6vw1E	1.14	0.93
4	7bz5A	1.11	0.93
5	7c01A	1.10	0.93
6	3bgfS	1.47	0.93
7	7jmoA	1.18	0.93
8	6m0jE	1.17	0.93
9	6w41C	1.15	0.93
10	7c01B	1.15	0.93

Figure 65: Search results returned for the criteria shown in Fig. 64. 3bgfS is #6

the 6m17E PDB chain against the SCOP database in all-aligned search mode. When normalized by the length of the d2dehf_ domain, the TM-score is 0.532. While this match is a bit messy and may not be functionally relevant, it demonstrates a remarkable similarity that we were only able to find by using all-aligned mode using a cross-database search with an entry from the PDB chain database against the SCOP database. We anticipate the flexibility offered by cross-database searches will prove useful for investigators interested in discovering previously unknown structural similarities.

Fig. 67 shows the criteria used to find the d2dehf_ match shown in Fig. 66. Fig. 68 shows 10 results starting at #25 returned for the criteria shown in Fig. 67. We decided to start the results in Fig. 68 at #25 because the earlier results were dominated by SARS receptor-binding domains. When performing a cross-database search using RUPEE, it is not possible to filter these highly similar domains out. RUPEE does not support filters for cross-database searches because the query structure and the structures in the searched database do not belong to the same classification hierarchy.

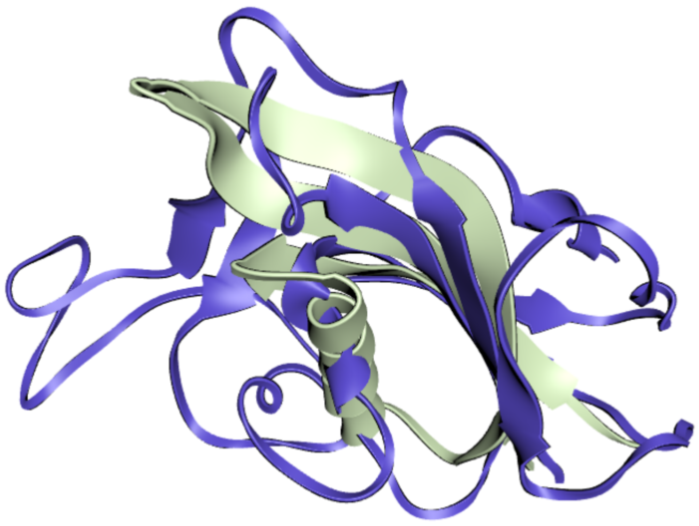
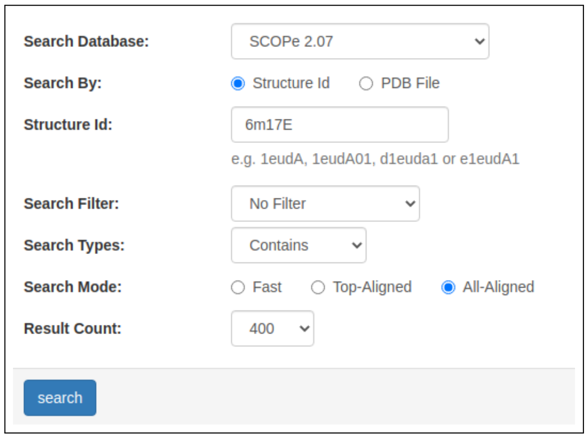
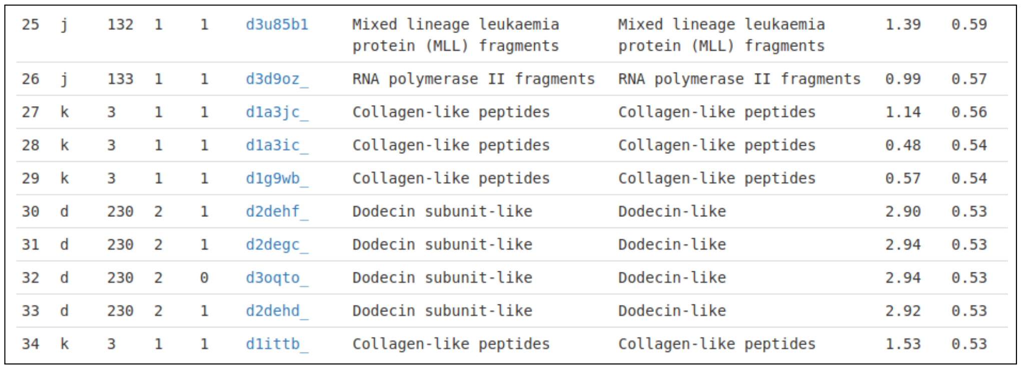


Figure 66: PDB chain 6m17E (blue) aligned with SCOP domain d2dehf_ (green)



Search Database:	SCOPe 2.07
Search By:	[✓ Structure Id] PDB File
Structure Id:	6m17E
	e.g. 1eudA, 1eudA01, d1euda1 or e1eudA1
Search Filter:	No Filter
Search Types:	Contains
Search Mode:	Fast Top-Aligned [✓ All-Aligned]
Result Count:	400

Figure 67: Search criteria used to find the d2dehf_ match shown in Fig. 66



25	j	132	1	1	d3u85b1	Mixed lineage leukaemia protein (MLL) fragments	Mixed lineage leukaemia protein (MLL) fragments	1.39	0.59
26	j	133	1	1	d3d9oz_	RNA polymerase II fragments	RNA polymerase II fragments	0.99	0.57
27	k	3	1	1	d1a3jc_	Collagen-like peptides	Collagen-like peptides	1.14	0.56
28	k	3	1	1	d1a3ic_	Collagen-like peptides	Collagen-like peptides	0.48	0.54
29	k	3	1	1	d1g9wb_	Collagen-like peptides	Collagen-like peptides	0.57	0.54
30	d	230	2	1	d2dehf_	Dodecin subunit-like	Dodecin-like	2.90	0.53
31	d	230	2	1	d2degc_	Dodecin subunit-like	Dodecin-like	2.94	0.53
32	d	230	2	0	d3oqto_	Dodecin subunit-like	Dodecin-like	2.94	0.53
33	d	230	2	1	d2dehd_	Dodecin subunit-like	Dodecin-like	2.92	0.53
34	k	3	1	1	d1ittb_	Collagen-like peptides	Collagen-like peptides	1.53	0.53

Figure 68: Search results returned for the criteria shown in Fig. 67. d2dehf_ is #30

CHAPTER 7

CONCLUSION

As shown above, a purely geometric big data approach to protein structure search can compete with the best available protein structure searches. Nonetheless, there remains room for improvement and opportunities for future work.

7.1 Room for Improvement

7.1.1 Containment Searches

While containment searches for all-aligned search mode consider the containment criterion at every step in the search process, as shown in Fig. 20 from Chapter 4, this is not the case for containment searches in fast and top-aligned search modes due to their dependence on min-hashing and LSH for the initial filtering, which filters for full-length matches only. While fast and top-aligned search modes do use the containment criterion for the top-scoring 40000 structures returned from the min-hashing and LSH initial filtering, many legitimate containment matches are inevitably missed.

One improvement for RUPEE would be to incorporate the containment criterion into the min-hashing and LSH step. While not specifically designed for containment searches, the initial min-hashing and LSH step does operate fast enough, within seconds, that it presents the possibility of executing multiple min-hashing and LSH lookups within the context of a single search. For instance, protein structures could be broken into multiple overlapping fragments with each successive fragment having a 75% overlap with the previous fragment. The protein structure fragments can then be indexed as described in Chapter 3. Then, when a Contains search is executed, the query structure can be fragmented in the same manner and for each fragment separate min-hashing and LSH lookups can be executed to identify candidate structures that are contained in the query structure.

We have tried the idea of overlapping fragments to implement containment searches and have found it to be effective. However, we do not have the resources to offer this at our web site, which as previously mentioned is running on a single AWS elastic compute unit.

7.1.2 Results Threshold

One possible weakness for RUPEE is that it only returns the top 400 results. With more resources, this number can be increased but there remains a need for some kind of cut-off. Without a cut-off, the RUPEE search would reduce to performing full TM-align protein structure alignments on all available structures, which can take several hours and even as much as a full day for larger proteins. Nonetheless, in most search use-cases, having to look past the first few hundred results usually indicates an ineffective search strategy.

To account for the limited number of results, RUPEE provides search filters that can be used for exploring structure space more efficiently. For the SCOPe, CATH, and ECOD protein structure databases, the user can instruct the search to only return domains that differ from the query structure at a chosen hierarchy level classification. Additionally, for the CATH database, the user can filter results based on hierarchy level representatives. Since RUPEE does not rely on sequence clusters or pre-calculated results, these kinds of filters are easy to implement, do not reduce the number of returned results, and allow for the discovery of unexpected structural similarities across classification hierarchies. Expanding the list of available search filters could further increase the utility of RUPEE.

7.1.3 Loop Regions

Another challenge for RUPEE is to effectively encode the loop regions of proteins. For short loops containing simple motifs such as the β-turn-β motif shown in Fig. 17, shingle matches correctly factor into RUPEE similarity comparisons. However, for longer and more complex loops, random matching and mismatching of shingles takes a toll further down in the list of ranked results. In one experiment, for longer loops, we truncated the center of the loops, leaving only a maximum of 5 residues at the start and end of each loop. We treated the truncated loops as normal gaps and did not shingle across the gaps. In this experiment, RUPEE only performed marginally worse than normal, providing one piece of evidence that longer loops are not being encoded as effectively as we would like.

7.1.4 Gene and Protein Names

Currently, when not uploading a PDB file, RUPEE requires a structure id. This structure id must be a valid PDB id followed by a chain identifier or a valid id from either SCOP, CATH, or ECOD protein structure classification databases. On the other hand, RUPEE does not accept common gene and protein names. This can lead to confusion for the casual user who may have a common gene or protein name in mind but does not have a specific structure id. For instance, a user may want to view proteins similar to the SARS CoV-2 spike protein and types in ‘sars cov-2 spike.’ In this case, the user will receive an ‘Invalid Structure Id’ error. A better response would be to present a pop-up dialog to the user with valid PDB ids that correspond to the common name entered by the user. This also would be instructive for the user, who will discover that there will usually be multiple structure ids for a given common gene or protein name.

For RUPEE, as with most research projects, there is room for improvement. With that in mind, we are always ready to consider any suggestions users and other researchers may want to contribute.

7.2 Future Work

7.2.1 Standalone Version

RUPEE has been used by users from around the world including China, France, Greece, India, Italy, Mexico, Switzerland, Turkey, and the United States. In the United States, RUPEE has been used by researchers at Rockefeller University in New York City, the San Diego Supercomputer Center, the National Institute of Health in Bethesda, and the Dana-Farber Cancer Institute in Boston. In particular, researchers from the Dana-Farber Cancer Institute have expressed interest in a standalone local version of RUPEE to integrate into their protein structure prediction pipeline. We suspect that the use of RUPEE would increase if a standalone local version was available.

While RUPEE can be installed locally by cloning the GitHub repository and working through the README file, or alternatively, working through Section 6.2 in this document, the process is quite involved. For instance, a PostgreSQL database has to be created and populated with data obtained from several sources available on the web. Additionally, several BASH and AWK shell scripts have to be run for the initial parsing of the data and 3 Java projects have to be compiled and run to process the data after setting up several configuration files.

A standalone local version of RUPEE should limit the dependencies described above as much as possible. A copy-and-paste installation would be ideal. For this to happen, the data layer would have to be either based on flat-files or a pre-populated database like SQLite. Since the size of these files exceed GitHub limits, they would have to be served via an FTP server or as large objects in AWS S3 storage in the cloud. Further, we would have to ensure no operating system dependencies are inadvertently introduced. Since Java is ubiquitous, this version of RUPEE would likely be a Java 8 console application that can also be automated for integration into a workflow.

7.2.2 Flexible Structure Alignments

We also have considered creating a more flexible version of RUPEE by using a more flexible protein structure alignment tool that accounts better for topological permutations and twists in the protein chain. Using a more flexible protein structure alignment along with a purely geometric protein structure search may allow RUPEE to be more effective at detecting remote homology.

Topological permutations such as circular permutations, segment-swapping, and changing secondary structures within homologous proteins are not uncommon [2]. As mentioned in Section 1.3.2, the FATCAT [66] protein structure alignment tool allows for a constrained number of twists in the protein chain to find a more flexible alignment before minimizing RMSD. Furthermore, to complement CE [60], CE-CP [9] was created to allow for circular permutations in CE protein structure alignments.

Since RUPEE depends on rigid TM-align structure alignments in its final step, it can be thought of more specifically as a *rigid* protein structure search. However, for fast and top-aligned search modes, the min-hashing and LSH steps impose no strict order requirements beyond the 3-residue sequences that are used to define the shingles. With a little work, and with the use of a flexible protein structure alignment tool in the final steps, a *flexible* version of the RUPEE protein structure search could be created.

As mentioned in Chapter 1, while high sequence similarity usually indicates high structure similarity [18], high structure similarity has been observed even for structures with low sequence similarity since structure is more conserved in evolution than sequence [35]. The case of high structure similarity coupled with low sequence similarity is either an instance of convergent evolution or remote homology. Several computational methods have been proposed for detecting remote homology in protein sequences [16]; however, a structural approach using the RUPEE protein structure search to detect remote homology may be more effective.

The combination of topological permutations and remote homology that can only be detected through structural similarity would require a flexible and purely geometric protein structure search. With modifications to RUPEE to allow for the option of a flexible structure search together with its current purely geometric approach, perhaps RUPEE can be used to address this combined problem effectively and find new and interesting evolutionary relationships between proteins.

7.2.3 Drug Discovery

The process of drug discovery for a particular disease begins with target identification, which typically identifies a protein as the target molecule that plays a role in the disease. Once a target protein has been identified, a library of small molecule drug candidates are screened to determine if they bind with the target. A suitable drug will bind the target in some manner, possibly causing an allosteric change in the target structure or simply preventing other molecules from binding the target. In this manner, the function of a protein in a disease can be disrupted.

Currently, drug discovery is a process burdened with high financial costs, high attrition rates, and slow time to market. Identifying proteins that are structurally similar to the target protein may help identify drug candidates that bind the target because if they are known to bind the structurally similar proteins they may bind the target. RUPEE full-length searches can be used to identify those structurally similar proteins, and RUPEE containment searches can be used to focus the search on identifying proteins containing substructures that are similar to possible binding sites on the target protein.

7.3 Final Remarks

With the 10% yearly growth rate of solved structures deposited in the PDB [57], the need for a fast, scalable, and accurate protein structure search is growing as well. Using run position encoded shingles of residue descriptors combined with min-hashing and LSH, we have shown that RUPEE in fast search mode can provide good results in seconds running on a single AWS elastic compute unit. Currently, RUPEE in fast search mode is the fastest available protein structure search providing the demonstrated level of accuracy. For RUPEE in top-aligned search mode, we have shown that a purely geometric big data approach to protein structure search can produce results equal to or better than the current state of the art protein structure searches that variously depend on protein sequences, clustered sequences or structures, pre-calculated results, or the exclusive use of secondary structure elements. Further, for RUPEE in all-aligned search mode, we have shown that RUPEE effectively addresses the problem of searching on protein structure predictions and is uniquely suited to support the development of protein structure prediction methods. Considered together, these accomplishments confirm that the RUPEE protein structure search can serve as an invaluable component of any bioinformatics toolkit.

In addition to RUPEE, we have introduced two items that may find wider applicability. The first item is the introduction of a new polar torsion angle plot that maintains the continuity of permissible torsion angle regions while maintaining the familiar torsion angle ranges used in Ramachandran plots. The second item is the *run position encoding* heuristic, which may find wider applicability due to its simplicity and generality.

APPENDIX A

BENCHMARKS OF KNOWN PROTEIN STRUCTURES

A.1 scop_d360

```
d1a1ca_,d1a92b_,d1ao6b1,d1b0ea_,d1bhaa_,d1bspa_,d1bwza1,d1bzqd_,
d1c1ba2,d1c3pa_,d1c81a_,d1cklf1,d1d5nc1,d1d6ha1,d1d8la1,d1de0a_,
d1di0c_,d1djwb2,d1e0ca1,d1e9ha_,d1egya_,d1ek1a2,d1f9qa_,d1ffxa_,
d1fh9a_,d1fmta2,d1fyfa1,d1fyfa2,d1g3it_,d1g63i_,d1gg3b2,d1gk8k_,
d1gqme_,d1gtea4,d1gvra_,d1gztd_,d1hnvb_,d1hzyb_,d1i7pa2,d1ia2a_,
d1ii2a2,d1ik4f_,d1isba2,d1iwao1,d1j6va_,d1jdya3,d1jeqa1,d1jh2a_,
d1jjid_,d1jrqb2,d1jxja2,d1jyzm3,d1k9jb_,d1kkmb_,d1kkmj_,d1kooc1,
d1kpsc_,d1kq4a1,d1krbc2,d1ksda1,d1l6kj_,d1lcpa1,d1liwa6,d1lvha_,
d1m80a3,d1mokc1,d1mpsh1,d1mwia_,d1ndqa_,d1ni6c_,d1nkza_,d1nzwb_,
d1o1cv_,d1odsf_,d1ogcd_,d1oh9a_,d1ojna2,d1p9ka1,d1perl_,d1pf9i3,
d1px3b4,d1q4qf_,d1q6za1,d1q86d_,d1qb8a_,d1qj3b_,d1qrqa_,d1qs7c_,
d1qvrc3,d1r67a1,d1roma_,d1s5ma_,d1sbka1,d1sg9a_,d1smyk2,d1srgb_,
d1sw6b_,d1sxua_,d1szsa1,d1th8a1,d1tjta2,d1tkab1,d1tkba3,d1tr9a1,
d1ttob_,d1tyzd_,d1tzyd_,d1u9ma_,d1uc8a1,d1uf5a_,d1uj1b_,d1uj4a1,
d1upla_,d1v1tb1,d1v54u_,d1vq5d1,d1w7ab2,d1w7vd2,d1wpua_,d1x0sa_,
d1x1yd1,d1x92b1,d1xckd1,d1xmzb1,d1xn2c_,d1xnwe2,d1xuoa_,d1xvaa_,
d1xyla_,d1y3be2,d1y69k1,d1yf1c_,d1ygcl_,d1yhut_,d1yima_,d1yl4w1,
```

d1yo6c_,d1yq2a4,d1ywhm3,d1z25a3,d1z3qa_,d1zcfg_,d1zdja_,d1zqqa4,
d220la_,d2avyh1,d2b66k2,d2b7aa_,d2b9nf1,d2b9no1,d2bb3a1,d2bfea1,
d2bm4a1,d2bq8x_,d2bsqc1,d2c64a2,d2c9lz1,d2ds0a_,d2dtyd_,d2dxia2,
d2e1va2,d2f3fa_,d2fdsa1,d2fk3c1,d2fugu1,d2g7ga1,d2gdub2,d2gjwd2,
d2gmqa1,d2hbxb_,d2hd1a_,d2hh2a_,d2hhhc2,d2idrb_,d2igad1,d2itjb_,
d2iw6b1,d2j31a1,d2j62b2,d2juaa1,d2lq7a1,d2noea1,d2nuwa_,d2o1ba_,
d2ofea_,d2oqef1,d2pf2a2,d2plsf2,d2puga2,d2q7qh_,d2qbij1,d2qbra_,
d2qexc1,d2qp1s1,d2quea_,d2ripa1,d2rmaq_,d2uxcg1,d2uynb_,d2v4eg_,
d2v6ma2,d2v7xa2,d2vggb2,d2vv6d_,d2vy0b_,d2wipa1,d2wipc1,d2wogb_,
d2x7yb_,d2yzdg1,d2z0ac1,d2z27b_,d2z4ng1,d2zfzd_,d2zwib_,d3a5cc3,
d3abmi_,d3aend_,d3ah3b_,d3anzi1,d3aofb_,d3ayzb_,d3b6oa_,d3bc5a1,
d3bp9f_,d3bwia_,d3bzub2,d3c6qb_,d3cjbg1,d3cojc2,d3d5dp1,d3dh1b1,
d3dllw1,d3dugf2,d3dyab_,d3e6kb_,d3ej8a_,d3f1sa_,d3f3yb_,d3flqa_,
d3foub_,d3fvlc_,d3g3cb1,d3guag1,d3gw1a_,d3gw9b_,d3h66b_,d3h90c2,
d3hjjc_,d3hlib_,d3hqpf2,d3ikab_,d3is3a_,d3k3ai_,d3k8ca_,d3kfab1,
d3kpbd_,d3l7uc_,d3l7zd2,d3m5ka_,d3mg8r_,d3n6fb_,d3ne4a_,d3nfdf_,
d3nvya2,d3nyxa1,d3o92b_,d3ohuf_,d3om5a_,d3ozqa1,d3pwsb2,d3q6ja3,
d3q7jb1,d3qpkb3,d3qu1b_,d3r83a_,d3resb_,d3so1c_,d3tbla1,d3tcrb_,
d3ubnc_,d3ujeb1,d3umld_,d3uwlb_,d3uyta_,d3vdba5,d3w7ob_,d3wefb_,
d3wrta_,d3wyla_,d3ximd_,d3ziaf3,d4a1ua_,d4a75g1,d4au0a_,d4b1tb_,
d4b7ba_,d4b7qd_,d4bpzb1,d4cqli_,d4d2oa_,d4d8gb1,d4db5a_,d4e4nb_,
d4f5lb1,d4fkwa_,d4frtb_,d4h0ta2,d4hhyd1,d4hx6a1,d4i5bb1,d4ii4b1,

d4isob_,d4iuaa1,d4ixzc_,d4j7xa_,d4jarc1,d4jcfa_,d4jhxc2,d4jyka2,
d4jyox_,d4k1wc2,d4k64c1,d4kv5a_,d4ltco_,d4lzwb_,d4mq6b_,d4mswc_,
d4mv6a2,d4mw5b1,d4napb_,d4nbjh_,d4nhzc2,d4otaa_,d4p3ya3,d4pd4f_,
d4pitb1,d4pvma_,d4q1la_,d4q1sy_,d4q2va_,d4qseb_,d4qv0c_,d4qv4o_,
d4qv5e_,d4qv8l_,d4qzwy_,d4r17d_,d4ryva_,d4tvab4,d4u39g1,d4unwa_,
d4xheh1,d4xkdf_,d4yuya_,d4zgla_,d5acra_,d5cpya_,d5rlaa_,d7ccpa_

A.2 scop_d62

d1ao6b1,d1b0ea_,d1brwa1,d1bspa_,d1bzqd_,d1c1ba2,d1cklf1,d1d8la1,
d1di0c_,d1dm5c_,d1e9ha_,d1egya_,d1f9qa_,d1fyfa1,d1g3it_,d1gg3b2,
d1gqme_,d1hc7c3,d1hnvb_,d1i6hf_,d1ia2a_,d1isba2,d1iwao1,d1j6va_,
d1jrqb2,d1jxja2,d1kkmj_,d1kpsc_,d1ndqa_,d1nzwb_,d1o1cv_,d1ogbb1,
d1pf9i3,d1px3b4,d1qj3b_,d1qrqa_,d1rhgc_,d1roma_,d1s5ma_,d1srgb_,
d1szsa1,d1tyzd_,d1tzyd_,d1v1tb1,d1v54u_,d1xyla_,d1zdja_,d1zqqa4,
d1zr4e2,d220la_,d2c64a2,d2f4vp1,d2igad1,d2iw6b1,d2nvqi2,d2oqef1,
d2puga2,d2rmaq_,d3ximd_,d4otaa_,d5rlaa_,d7ccpa_

A.3 cath_d99

1bxoA02,1byiA00,1hdoA00,1jfbA00,1k5nA01,1l3kA01,1lqtB02,1n3lA01,
1n5uA03,1nkiA00,1nwwA00,1nz0D00,1psrA00,1r0mA01,1rl6A02,1rtqA00,
1swyA00,1unqA00,1vimC00,1wmwB00,1xmkA00,2bw4A01,2dkjA02,2fvyA01,
2nrlA00,2o9sA00,2ob3A00,2oizA01,2osxA01,2rbkA01,2vb1A00,2vxnA00,

2w8tA01,2wn9D00,2wurA00,2ylbC00,2zcmA00,3a02A00,3dlcA00,3e2oA01,
3go9A02,3goeA00,3h7iA02,3hj4A02,3iohA00,3kffA00,3ks3A00,3ku3A02,
3ku3B00,3lqbA00,3nbcA00,3sovA02,3t5tB01,3u7qA02,3uljB00,3w5hA01,
3wh2A00,3ziyA03,4bj0A00,4cayB00,4cbuA01,4cvrA00,4d3tA02,4d3tA03,
4dd5A01,4ep4A00,4f1vA02,4fvyA01,4g1qA04,4k8gA02,4l8aA00,4mf5A02,
4mtuA00,4oh7A02,4pf3A00,4r2xD00,4ua6A00,4unuA00,4xemA01,4yapA01,
4z8jA00,4zflD00,5a71A00,5avdA02,5cgqB01,5cphA00,5dp2A01,5dzeA00,
5hyvA02,5ibnA00,5jbxB01,5jryA01,5jryA02,5k8sB00,5kvsA02,5lvoA01,
5lvoA02,5m17A00,5sy4A00

APPENDIX B

BENCHMARKS OF PROTEIN STRUCTURE PREDICTIONS

B.1 casp_d250

```
T0953s1TS043-D1,T0953s1TS089-D1,T0953s1TS145-D1,
T0953s1TS196-D1,T0953s1TS197-D1,T0953s1TS224-D1,
T0953s1TS261-D1,T0953s1TS322-D1,T0953s1TS354-D1,
T0953s1TS498-D1,T0957s2TS043-D1,T0957s2TS089-D1,
T0957s2TS145-D1,T0957s2TS196-D1,T0957s2TS197-D1,
T0957s2TS224-D1,T0957s2TS261-D1,T0957s2TS322-D1,
T0957s2TS354-D1,T0957s2TS498-D1,
T0960TS043-D2,T0960TS089-D2,T0960TS145-D2,T0960TS196-D2,
T0960TS197-D2,T0960TS224-D2,T0960TS261-D2,T0960TS322-D2,
T0960TS354-D2,T0960TS498-D2,T0963TS043-D2,T0963TS089-D2,
T0963TS145-D2,T0963TS196-D2,T0963TS197-D2,T0963TS224-D2,
T0963TS261-D2,T0963TS322-D2,T0963TS354-D2,T0963TS498-D2,
T0968s1TS043-D1,T0968s1TS089-D1,T0968s1TS145-D1,
T0968s1TS196-D1,T0968s1TS197-D1,T0968s1TS224-D1,
T0968s1TS261-D1,T0968s1TS322-D1,T0968s1TS354-D1,
T0968s1TS498-D1,T0968s2TS043-D1,T0968s2TS089-D1,
T0968s2TS145-D1,T0968s2TS196-D1,T0968s2TS197-D1,
```

T0968s2TS224-D1,T0968s2TS261-D1,T0968s2TS322-D1,

T0968s2TS354-D1,T0968s2TS498-D1,

T0969TS043-D1,T0969TS089-D1,T0969TS145-D1,T0969TS196-D1,

T0969TS197-D1,T0969TS224-D1,T0969TS261-D1,T0969TS322-D1,

T0969TS354-D1,T0969TS498-D1,T0975TS043-D1,T0975TS089-D1,

T0975TS145-D1,T0975TS196-D1,T0975TS197-D1,T0975TS224-D1,

T0975TS261-D1,T0975TS322-D1,T0975TS354-D1,T0975TS498-D1,

T0980s1TS043-D1,T0980s1TS089-D1,T0980s1TS145-D1,

T0980s1TS196-D1,T0980s1TS197-D1,T0980s1TS224-D1,

T0980s1TS261-D1,T0980s1TS322-D1,T0980s1TS354-D1,

T0980s1TS498-D1,T0986s2TS043-D1,T0986s2TS089-D1,

T0986s2TS145-D1,T0986s2TS196-D1,T0986s2TS197-D1,

T0986s2TS224-D1,T0986s2TS261-D1,T0986s2TS322-D1,

T0986s2TS354-D1,T0986s2TS498-D1,T0987TS043-D1,

T0987TS043-D2,T0987TS089-D1,T0987TS089-D2,T0987TS145-D1,

T0987TS145-D2,T0987TS196-D1,T0987TS196-D2,T0987TS197-D1,

T0987TS197-D2,T0987TS224-D1,T0987TS224-D2,T0987TS261-D1,

T0987TS261-D2,T0987TS322-D1,T0987TS322-D2,T0987TS354-D1,

T0987TS354-D2,T0987TS498-D1,T0987TS498-D2,T0989TS043-D1,

T0989TS043-D2,T0989TS089-D1,T0989TS089-D2,T0989TS145-D1,

T0989TS145-D2,T0989TS196-D1,T0989TS196-D2,T0989TS197-D1,

T0989TS197-D2,T0989TS224-D1,T0989TS224-D2,T0989TS261-D1,

T0989TS261-D2,T0989TS322-D1,T0989TS322-D2,T0989TS354-D1,
T0989TS354-D2,T0989TS498-D1,T0989TS498-D2,T0990TS043-D1,
T0990TS043-D3,T0990TS089-D1,T0990TS089-D3,T0990TS145-D1,
T0990TS145-D3,T0990TS196-D1,T0990TS196-D3,T0990TS197-D1,
T0990TS197-D3,T0990TS224-D1,T0990TS224-D3,T0990TS261-D1,
T0990TS261-D3,T0990TS322-D1,T0990TS322-D3,T0990TS354-D1,
T0990TS354-D3,T0990TS498-D1,T0990TS498-D3,T0998TS043-D1,
T0998TS089-D1,T0998TS145-D1,T0998TS196-D1,T0998TS197-D1,
T0998TS224-D1,T0998TS261-D1,T0998TS322-D1,T0998TS354-D1,
T0998TS498-D1,T1000TS043-D2,T1000TS089-D2,T1000TS145-D2,
T1000TS196-D2,T1000TS197-D2,T1000TS224-D2,T1000TS261-D2,
T1000TS322-D2,T1000TS354-D2,T1000TS498-D2,T1001TS043-D1,
T1001TS089-D1,T1001TS145-D1,T1001TS196-D1,T1001TS197-D1,
T1001TS224-D1,T1001TS261-D1,T1001TS322-D1,T1001TS354-D1,
T1001TS498-D1,T1010TS043-D1,T1010TS089-D1,T1010TS145-D1,
T1010TS196-D1,T1010TS197-D1,T1010TS224-D1,T1010TS261-D1,
T1010TS322-D1,T1010TS354-D1,T1010TS498-D1,
T1015s1TS043-D1,T1015s1TS089-D1,T1015s1TS145-D1,
T1015s1TS196-D1,T1015s1TS197-D1,T1015s1TS224-D1,
T1015s1TS261-D1,T1015s1TS322-D1,T1015s1TS354-D1,
T1015s1TS498-D1,T1017s2TS043-D1,T1017s2TS089-D1,
T1017s2TS145-D1,T1017s2TS196-D1,T1017s2TS197-D1,

T1017s2TS224-D1,T1017s2TS261-D1,T1017s2TS322-D1,
T1017s2TS354-D1,T1017s2TS498-D1,T1021s3TS043-D1,
T1021s3TS043-D2,T1021s3TS089-D1,T1021s3TS089-D2,
T1021s3TS145-D1,T1021s3TS145-D2,T1021s3TS196-D1,
T1021s3TS196-D2,T1021s3TS197-D1,T1021s3TS197-D2,
T1021s3TS224-D1,T1021s3TS224-D2,T1021s3TS261-D1,
T1021s3TS261-D2,T1021s3TS322-D1,T1021s3TS322-D2,
T1021s3TS354-D1,T1021s3TS354-D2,T1021s3TS498-D1,
T1021s3TS498-D2,T1022s1TS043-D1,T1022s1TS089-D1,
T1022s1TS145-D1,T1022s1TS196-D1,T1022s1TS197-D1,
T1022s1TS224-D1,T1022s1TS261-D1,T1022s1TS322-D1,
T1022s1TS354-D1,T1022s1TS498-D1

B.2 casp_ssm_d248

Same as casp_d250 excluding:

T0968s1TS261-D1,T0989TS043-D1

B.3 casp_cathedral_d247

Same as casp_d250 excluding:

T0953s1TS498-D1,T0969TS498-D1,T0980s1TS498-D1

B.4 casp_vast_d199

Same as casp_d250 excluding:

```
T0953s1TS498-D1,T0968s1TS196-D1,
T0969TS196-D1,T0969TS354-D1,T0975TS043-D1,T0975TS089-D1,
T0975TS196-D1,T0975TS261-D1,T0975TS322-D1,T0975TS354-D1,
T0975TS498-D1,
T0980s1TS224-D1,T0980s1TS261-D1,T0980s1TS322-D1,
T0986s2TS354-D1,
T0987TS089-D2,T0987TS145-D2,T0987TS261-D2,T0987TS322-D2,
T0987TS354-D2,T0987TS498-D2,T0989TS145-D2,T0989TS196-D2,
T0990TS043-D3,T0990TS089-D1,T0990TS089-D3,T0990TS145-D1,
T0990TS196-D1,T0990TS196-D3,T0990TS197-D1,T0990TS224-D1,
T0990TS224-D3,T0990TS261-D1,T0990TS261-D3,T0990TS322-D1,
T0990TS322-D3,T0990TS354-D1,T0990TS498-D3,T0998TS043-D1,
T0998TS089-D1,T0998TS145-D1,T0998TS196-D1,T0998TS261-D1,
T0998TS322-D1,T0998TS354-D1,T0998TS498-D1,T1010TS043-D1,
T1010TS089-D1,T1010TS261-D1,T1010TS354-D1,
T1021s3TS043-D1
```

REFERENCE LIST

[1] AlQuraishi, M. AlphaFold at CASP13. *Bioinformatics* (05 2019).

[2] Andreeva, A., Prlić, A., Hubbard, T. J., and Murzin, A. G. SISYPHUS: Structural alignments for proteins with non-trivial relationships. *Nucleic Acids Res. 35*, 1 (2007), 253–259.

[3] Anfinsen, C. B. Principles that govern the folding of protein chains. *Science 181*, 4096 (1973), 223–230.

[4] Aungand, Z., and Tan, K.-L. Rapid 3D protein structure database searching using information retrieval techniques. *Bioinformatics 20*, 7 (2004), 1045–1052.

[5] Ayoub, R., and Lee, Y. RUPEE: Scalable protein structure search using run position encoded residue descriptors. In *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)* (Nov 2017), pp. 74–78.

[6] Ayoub, R., and Lee, Y. RUPEE: A fast and accurate purely geometric protein structure search. *PLOS ONE 14*, 3 (03 2019), 1–17.

[7] Ayoub, R., and Lee, Y. Protein structure search to support the development of protein structure prediction methods. *Proteins: Structure, Function, and Bioinformatics 89*, 6 (2021), 648–658.

[8] Bellman, R. Dynamic programming. *Science 153*, 3731 (1966), 34–37.

[9] Bliven, S. E., Bourne, P. E., and Prlić, A. Detection of circular permutations within protein structures using CE-CP. *Bioinformatics 31*, 8 (2015), 1316–1318.

[10] Broder, A. Z. On the resemblance and containment of documents. In *Proc. Compression and Complexity of Sequences* (Positano, Italy, 1997), pp. 21–29.

[11] Broder, A. Z., Charikar, M., Frieze, A. M., and Mitzenmacher, M. Min-wise independent permutations. In *ACM Symposium on Theory of Computing* (Dallas, USA, 1998), pp. 327–336.

[12] Broder, A. Z., Glassman, S. C., Manasse, M. S., and Zweig, G. Syntactic clustering of the Web. *Computer Networks and ISDN Systems 29*, 8-13 (1997), 1157–1166.

[13] Brudno, M., Malde, S., Poliakov, A., Do, C. B., Couronne, O., Dubchak, I., and Batzoglou, S. Glocal alignment: finding rearrangements during alignment. *Bioinformatics 19* (07 2003), i54–i62.

[14] Budowski-Tal, I., Nov, Y., and Kolodny, R. FragBag, an accurate representation of protein structure, retrieves structural neighbors from the entire PDB quickly and accurately. *Proceedings of the National Academy of Sciences 107*, 8 (2010), 3481–3486.

[15] Burley, S. K., Berman, H. M., Christie, C., Duarte, J. M., Feng, Z., Westbrook, J., Young, J., and Zardecki, C. RCSB protein data bank: Sustaining a living digital data resource that enables breakthroughs in scientific research and biomedical education. *Protein Science 27*, 1 (2018), 316–330.

[16] Chen, J., Guo, M., Wang, X., and Liu, B. A comprehensive review and comparison of different computational methods for protein remote homology detection. *Briefings in Bioinformatics 19*, 2 (11 2016), 231–244.

[17] Cheng, H., Schaeffer, R. D., Liao, Y., Kinch, L. N., Pei, J., Shi, S., Kim, B. H., and Grishin, N. V. ECOD: An evolutionary classification of protein domains. *PLoS Computational Biology 10*, 12 (2014).

[18] Chothia, C., and Lesk, A. The relation between the divergence of sequence and structure in proteins. *The EMBO Journal 5*, 4 (1986), 823–826.

[19] Consortium, T. U. UniProt: The universal protein knowledgebase. *Nucleic Acids Research 45*, D1 (2017), D158–D169.

[20] Cornell, B. *BioNinja*, 2016. Availble at `https://ib.bioninja.com.au` [Online; accessed April 26, 2021].

[21] Cox, A. L. L. D. L. N. M. M. *Lehninger principles of biochemistry*. W.H. Freeman, 2005, ch. 4.

[22] Crick, F. Central dogma of molecular biology. *Nature 227*, 5258 (1970), 561–563.

[23] Dawson, N. L., Lewis, T. E., Das, S., Lees, J. G., Lee, D., Ashford, P., Orengo, C. A., and Sillitoe, I. CATH: An expanded resource to predict protein function through structure and sequence. *Nucleic Acids Res. 45*, D1 (2017), D289–D295.

[24] de Chadarevian, S. John Kendrew and myoglobin: Protein structure determination in the 1950s. *Protein Science 27*, 6 (2018), 1136–1143.

[25] Dong, R., Pan, S., Peng, Z., Zhang, Y., and Yang, J. mTM-align: A server for fast protein structure database search and multiple protein structure alignment. *Nucleic Acids Research 46*, July (2018), 380–386.

[26] Fox, N. K., Brenner, S. E., and Chandonia, J.-M. SCOPe: Structural Classification of Proteins - extended, integrating SCOP and ASTRAL data and classification of new structures. *Nucleic Acids Res. 42*, 1 (2014), 304–309.

[27] George, J. M. The synucleins. *Genome Biology 3* (2001), 1474–1476.

[28] Gibrat, J. F., Madej, T., and Bryant, S. H. Surprising similarities in structure comparison. *Current Opinion in Structural Biology 6*, 3 (1996), 377–385.

[29] Hardy, J., and Higgins, G. Alzheimer's disease: The amyloid cascade hypothesis. *Science 256*, 5054 (1992), 184–185.

[30] Hirschberg, D. S. Algorithms for the longest common subsequence problem. *J. ACM 24*, 4 (10 1977), 664–675.

[31] Hollingsworth, S. A., and Karplus, P. A. A fresh look at the Ramachandran plot and the occurrence of standard structures in proteins. *Biomol. Concepts 1* (2010), 271–283.

[32] Holm, L., and Rosenström, P. Dali server: Conservation mapping in 3D. *Nucleic Acids Research 38*, SUPPL. 2 (2010), 1–5.

[33] Holm, L., and Sander, C. Dali: A network tool for protein structure comparison. *Trends in Biochemical Sciences 20*, 11 (1995), 478 – 480.

[34] Hunt, J. W., and Szymanski, T. G. A fast algorithm for computing longest common subsequences. *Communications ACM 20*, 5 (1977), 350–353.

[35] Illergard, K., Ardell, D. H., and Elofsson, A. Structure is three to ten times more conserved than sequence - a study of structural response in protein cores. *Proteins: Structure, Function, and Bioinformatics 77*, 3 (2009), 499–508.

[36] Indyk, P., and Motwani, R. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *ACM Symposium on Theory of Computing* (Dallas, USA, 1998), pp. 604–613.

[37] J Xu, Y. Z. How significant is a protein structure similarity with TM-score = 0.5? *Bioinformatics 26*, 7 (2010), 889–895.

[38] Jing Han, Haihong E, Guan Le, and Jian Du. Survey on NoSQL databases. In *2011 6th International Conference on Pervasive Computing and Applications* (2011), pp. 363–366.

[39] Joshi, S., Contractor, D., Ng, K., Deshpande, P. M., and Hampp, T. Auto-grouping emails for faster e-discovery. In *Proceedings of Very Large Databases Endowment 2011* (2011), vol. 4, pp. 1284–1294.

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

[41] Karp, R. M., and Rabin, M. O. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development 31*, 2 (1987), 249–260.

[42] Katib, A., Slavov, V., and Rao, P. RIQ: Fast processing of SPARQL queries on RDF quadruples. *Journal of Web Semantics 37-38* (2016), 90–111.

[43] Kinch, L. N., Kryshtafovych, A., Monastyrskyy, B., and Grishin, N. V. CASP13 target classification into tertiary structure prediction categories. *Proteins: Structure, Function, and Bioinformatics 87*, 12 (2019), 1021–1036.

[44] Krissinel, E., and Henrick, K. Secondary-structure matching (SSM), a new tool for fast protein structure alignment in three dimensions. *Acta Crystallographica Section D: Biological Crystallography 60*, 12 I (2004), 2256–2268.

[45] Levandowsky, M., and Winter, D. Distance between sets. *Nature 234*, 5323 (1971), 34–35.

[46] Moult, J., Fidelis, K., Kryshtafovych, A., Schwede, T., and Tramontano, A. Critical assessment of methods of protein structure prediction: Progress and new directions in round XI. *Proteins: Structure, Function, and Bioinformatics 84*, S1 (2016), 4–14.

[47] Moult, J., Pedersen, J. T., Judson, R., and Fidelis, K. A large-scale experiment to assess protein structure prediction methods. *Proteins: Structure, Function, and Bioinformatics 23*, 3 (1995), ii–iv.

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

[49] Orengo, C., Mitchie, A., Jones, S., Jones, D. T., Swindells, M., and Thornton, J. M. CATH - A hierarchic classification of protein domain structures. *Structure 5*, 8 (1997), 1093–1109.

[50] Orengo, C. A. Protein structure alignment. *J. Mol. Biol.* (1989).

[51] *PDB-101*, 2000. Available at `http://pdb101.rcsb.org/` [Online; accessed April 26, 2021].

[52] Polymeropoulos, M. H., Lavedan, C., Leroy, E., Ide, S. E., Dehejia, A., Dutra, A., Pike, B., Root, H., Rubenstein, J., Boyer, R., Stenroos, E. S., Chandrasekharappa, S., Athanassiadou, A., Papapetropoulos, T., Johnson, W. G., Lazzarini, A. M., Duvoisin, R. C., Di Iorio, G., Golbe, L. I., and Nussbaum, R. L. Mutation in the α-synuclein gene identified in families with Parkinson's disease. *Science 276*, 5321 (1997), 2045–2047.

[53] Rajaraman, A., and Ullman, J. D. *Mining of massive datasets*. Cambridge University Press, 2012, ch. 3, pp. 53–70.

[54] Ramachandran, G. N., and Sasisekharan, V. Conformation of polypeptides and proteins. *Adv. Protein Chem. 23* (1968), 283–438.

[55] Redfern, O. C., Harrison, A., Dallman, T., Pearl, F. M., and Orengo, C. A. CATHEDRAL: A fast and effective algorithm to predict folds and domain boundaries from multidomain protein structures. *PLoS Comput. Biol. 3*, 11 (2007), e232.

[56] Rohl, C. A., Strauss, C. E., Misura, K. M., and Baker, D. Protein structure prediction using Rosetta. In *Numerical Computer Methods, Part D*, vol. 383 of *Methods in Enzymology*. Academic Press, 2004, pp. 66–93.

[57] Rose, P. W., Prlić, A., Altunkaya, A., Bi, C., Bradley, A. R., Christie, C. H., Costanzo, L. D., Duarte, J. M., Dutta, S., Feng, Z., Green, R. K., Goodsell, D. S., Hudson, B., Kalro, T., Lowe, R., Peisach, E., Randle, C., Rose, A. S., Shao, C., Tao, Y.-P., Valasatava, Y., Voigt, M., Westbrook, J. D., Woo, J., Yang, H., Young, J. Y., Zardecki, C., Berman, H. M., and Burley, S. K. The RCSB protein data bank: Integrative view of protein, gene and 3D structural information. *Nucleic Acids Res. 45*, D1 (2017), D271–D281.

[58] Sanger, F., and Coulson, A. A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase. *Journal of Molecular Biology 94*, 3 (1975), 441 – 448.

[59] Schuster, S. C. Next-generation sequencing transforms today's biology. *Nature Methods 5* (2008), 16–19.

[60] Shindyalov, I. N., and Bourne, P. E. Protein structure alignment by incremental combinatorial extension (CE) of the optimal path. *Protein Eng. Des. Sel. 11*, 9 (1998), 739–747.

[61] Smith, T., and Waterman, M. Identification of common molecular subsequences. *Journal of Molecular Biology 147*, 1 (1981), 195–197.

[62] *VAST Help Document*, 2016. Available at `https://structure.ncbi.nlm.nih.gov/Structure/VAST/vasthelp.html` [Online; accessed October 4, 2020].

[63] Wang, Y., and Ha, Y. The X-ray structure of an antiparallel dimer of the human amyloid precursor protein E2 domain. *Molecular Cell 15*, 3 (2004), 343–353.

[64] Wikipedia contributors. *Dihedral angles – Wikipedia, The Free Encyclopedia*, 2021. Availble at `https://en.wikipedia.org/wiki/Dihedral_angle` [Online; accessed April 26, 2021].

[65] Wikipedia contributors. *Needleman-Wunsch algorithm – Wikipedia, The Free Encyclopedia*, 2021. Availble at `https://en.wikipedia.org/wiki/Needleman-Wunsch_algorithm` [Online; accessed April 26, 2021].

[66] Yuzhen Ye, A. G. Flexible structure alignment by chaining aligned fragment pairs allowing twists. *Bioinformatics 19*, 2 (2003), 246–255.

[67] Zhang, L., Bailey, J., Konagurthu, A. S., and Ramamohanarao, K. A fast indexing approach for protein structure comparison. *BMC Bioinformatics 11*, Suppl 1 (2010), S46.

[68] Zhang, Y., and Skolnick, J. Scoring function for automated assessment of protein structure template quality. *Proteins 57*, 4 (2004), 702–710.

[69] Zhang, Y., and Skolnick, J. TM-align: A protein structure alignment algorithm based on the TM-score. *Nucleic Acids Research 33*, 7 (2005), 2302–2309.

VITA

Ronald Ayoub obtained his Bachelor of Science degree from the University of Maryland in 1999 and his Master of Science degree from the Johns Hopkins Whiting School of Engineering in 2007. Since his graduation from the University of Maryland up until his entry into the Interdisciplinary PhD program at UMKC, Ronald has worked in the industry as a software engineer and software engineering lead at a variety of companies including Lockheed Martin, Thomson Reuters, and finally Epiq Systems, Inc. here in Kansas City. He is currently settled in the Kansas City area with his wife, two kids, and two dogs. His ambition following graduation is to return to work in the industry as a computer scientist with a focus on computational biology.