A Data-Centric Approach for Investigation of Protein-Protein Interfaces in Protein Data Bank

by

Zeynep Abalı

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Data Science

KOÇ
ÜNİVERSİTESİ

January 28, 2021

A Data-Centric Approach for Investigation of Protein-Protein Interfaces in Protein Data Bank

Koç University
Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Zeynep Abalı

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Prof. Dr. Özlem Keskin (Advisor)

_____

Assoc. Prof. Dr. Mehmet Sayar

_____

Assoc. Prof. Dr. Nurcan Tunçbağ

Date: _____

ABSTRACT

A Data-Centric Approach for Investigation of

Protein-Protein Interfaces in Protein Data Bank

Zeynep Abalı

Master of Science in Data Science

January 28, 2021

Understanding the structural architecture of protein interfaces is one of the key challenges in explaining how proteins interact and function. Protein-protein interfaces can be important targets for drug discovery and repurposing studies; this is only possible if the structural data available can be utilized in a meaningful way. Advancements in experimental techniques make it possible for scientists to determine larger and more complex protein structures. As these tools are becoming more accessible, the number of protein structures deposited in PDB is also increasing rapidly. This increase in the size of the protein complexes and the size of the data is an invaluable tool for understanding protein interactions. Still, it does not come without its computational challenges. This study first focuses on identifying protein-protein interfaces and overcoming some of the difficulties rooting from growing structure sizes in PDB. We evaluated 169,681 available protein structures in PDB and extracted 499,169 protein-protein interfaces. A unique set of protein interfaces can only be created by structural comparison of interfaces. The increasing number of interface structures makes it almost impossible to compare all interface structures in a reasonable time frame. Therefore, computational solutions are needed to reduce the comparison number within a manageable range without losing structural information. The second part of this study explains an approach for filtering highly similar proteins using sequential and structural information. As a result of this process, we identified 331,231 interfaces, unique in terms of sequence and structure. The reduced number of interfaces results in almost a 2.3-fold reduction in the number of comparisons needed, without losing any structural information. Implementing these techniques can make future studies on protein interface comparison possible for large data sets.

ÖZETÇE

Protein Veri Bankası'nda Bulunan Protein-Protein Ara Yüzlerinin İncelenmesinde Veri Odaklı Bir Yaklaşım

Zeynep Abalı

Veri Bilimleri, Yüksek Lisans

28 Ocak 2021

Protein arayüzlerinin yapısal mimarisini anlamak, proteinlerin nasıl etkileşime girdiği ve fonksiyonlarını nasıl yerine getirdiğini anlamak için oldukça önemlidir. Protein-protein arayüzleri, ilaç keşfi ve yeniden konumlandırma çalışmaları için önemli hedefler olabilir; bu ancak mevcut yapısal verilerin anlamlandırılabilmesiyle mümkündür. Deneysel tekniklerdeki gelişmeler, daha büyük ve daha karmaşık protein yapılarının tespit edilmesini mümkün kılmaktadır. Bu araçlar daha erişilebilir hale geldikçe, PVB'de saklanan protein yapılarının sayısı da hızla artmaktadır. Protein komplekslerinin boyutundaki ve verilerin sayısındaki bu artış, protein etkileşimlerini anlamak için paha biçilmez bir araçtır; ancak bu veri bir takım hesaplamalı zorlukları beraberinde getirmektedir. Bu çalışma, öncelikle protein-protein arayüzlerini belirlemeye ve PVB'de bulunan yapı boyutlarının büyümesinden kaynaklanan ve verinin protein arayüzleri açısından analizinde ortaya çıkan bazı zorlukların çözümü için öneriler sunmaktadır. Bu çalışmada, PVB'de bulunan 169,681 protein yapısını değerlendirdik ve 499,169 protein-protein ara yüzü elde ettik. Benzersiz bir protein arayüzleri seti, tüm arayüzlerin yapısal olarak karşılaştırılmasıyla oluşturulabilir. Artan arayüz yapısı, tüm arayüz yapılarını makul bir zaman çerçevesinde karşılaştırmayı neredeyse imkansız kılmaktadır. Bu nedenle, karşılaştırma sayısını yapısal bilgileri kaybetmeden değerlendirilebilir bir aralığa getirebilmek için hesaplamalı çözümlere ihtiyaç vardır. Bu çalışmanın ikinci bölümü, protein sekans ve yapı bilgileri kullanılarak oldukça benzer proteinleri filtrelemek için kullanılabilecek bir yöntem sunmaktadır. Bu süreç sonucunda sıra ve yapı bakımından benzersiz 331.231 arayüz belirledik. Bu filtreleme sonucunda gereken karşılaştırma sayısı, herhangi bir yapısal bilgi kaybetmeden, 2,3 kat azalmıştır. Bu tekniklerin uygulanması, büyük veri kümeleri için protein arayüz karşılaştırması üzerine gelecekte yapılması planlanan çalışmaları mümkün kılabilir.

ACKNOWLEDGEMENTS

Foremost, I would like to thank my advisors Prof. Dr. Özlem Keskin and Prof. Dr Attila Gürsoy, for their support and guidance throughout this study. I am thankful beyond words for letting me be a part of COSBI group and being there whenever I needed help, and more.

I appreciate the time and guidance of my thesis jury members, Assoc. Prof. Dr. Mehmet Sayar, and Assoc. Prof. Dr Nurcan Tunçbağ. Their contributions through the years have been invaluable.

COSBI group is more than anyone can ask for; I am grateful for being friends with each and every amazing person in the group. Thank you for all the times we spent together, and your support. Simge, Kayra, Damla and Emre thank you for being part of weirdest questions and conversations. Your friendship is greatly appreciated, and I hope it will continue for years to come.

I would like to thank Beytullah Özgür; you have been there since almost day one. Today would probably not be possible without you. Thank you for never letting me feel lost and alone, and your patience with answering my endless questions, even when it is not your job description anymore. Dudu Erol, and Emre Ayar, thank you for putting up with me for over a decade; I know it is not always easy. Your support and friendship mean the world to me. I hope we have a few more decades in front of us, preferably in the same city again. Long distance thing is harder than one may assume.

I owe the deepest gratitude to my family. They always supported me through thick and thin, and all the somewhat crazy decisions throughout the years. I am so grateful for having them by my side.

Last but definitely not least; Alper Saydam, thank you for your endless love and support. I always had you by my side through the hills to climb (literally). This life would not be the same without you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

Å	Ångström
DB	Database
DF	Data Frame
HDF5	Hierarchical Data Format version 5
HPC	High-Performance Computing
NMR	Nuclear Magnetic Resonance
PDB	Protein Data Bank
PPI	Protein-Protein Interface
RDBMS	Relational Database Management System
RMSD	Root Mean Square Deviation
VDW	Van der Waals

Chapter 1:
INTRODUCTION

Proteins have crucial functions in many processes, ranging from structural building blocks to signaling or acting as catalysts to store and transport different molecules. Understanding how proteins carry out their functions is one of the critical challenges. Most functions of proteins are not conducted in isolation by a monomer but with the interaction of multiple partners. Defining and understanding how proteins interact with each other may help us explain mechanisms of different diseases, faster screening of already available drugs for drug repurposing, or even proposing new, more efficient drugs with reduced side effects.

Proteins interact with each other through regions called interfaces. Even though the number of protein structures determined each year is increasing, in 1996, it is estimated that there are around 1,000 types of different protein folds available in nature. In 2004 it was estimated that there should be approximately 10,000 structural protein interactions. [1, 2]. Identifying unique types of protein interactions can only be possible by utilizing the available structural data on proteins.

Protein Data Bank (PDB) provides information about the three-dimensional structures of proteins. It has been the single repository for structural information on proteins since 1971. More than 160,000 structures are deposited as of today, ranging from small monomer structures like ubiquitin to considerably large structures such as the entire HIV-1 capsid.

Protein structures deposited in PDB increase in number and size of structures each year due to advances in experimental protein structure determination techniques. Using this invaluable resource in structural biology is only possible with the efficient use of computational tools and resources.

Utilizing the computational power that comes with advances in technology, and tools such as machine learning algorithms, for structural biology, makes it possible to have a broader and more detailed perspective on the available data. Current structural data can only be handled effectively by using tools created for

big data studies. Efficient use of databases is required for handling and storage of the structural data.

The number of structures is not the only challenge. Large structures should also be handled differently to provide the necessary computational performance. Evaluation of these structures in a reasonable time frame requires selecting the right computational tools that can overcome the challenges created by larger structures.

In this study, we present a data set of protein-protein interface structures available in PDB. We explain the methods used to build the data set and propose techniques to overcome computational challenges created by the deposition of large structures. Also, we provide an analysis of the current interface data set in terms of sequential and structural similarities.

The outline of this thesis can be summarized as follows:

In Chapter 2, the literature information about the definition and properties of protein interfaces is given, and some of the currently available protein interface databases are demonstrated.

In Chapter 3, a detailed explanation of methods for the generation of interface data set is provided. Gathering of protein structures, evaluation of dimers, and extracting interface structures are covered in the first part. The second part explains the methods used for sequence-wise and structural comparisons of protein interfaces and interface chains.

In Chapter 4, we first present an analysis of structures in PDB throughout the years. Then, we provide an analysis of the current interface data set. Finally, the results of the sequence-wise grouping of interfaces and chains are explained in this chapter.

The last chapter includes the conclusion of the current study and planned future work.

Chapter 2:
LITERATURE REVIEW

2.1 Protein-Protein Interfaces

Proteins carry out most of their functions as part of macromolecular complexes. They need to interact with other molecules in their environment to function properly. Proteins interact with other proteins through regions called protein-protein interfaces. Figure 2.1 shows chain A and B and the interface region between them from *Aplysia* acetylcholine-binding protein (PDB ID: 2WNJ [3]) as an example interface. Even though there is an extensive number of possible protein-protein interactions, it is suggested that there is only a limited number of interface architectures for protein-protein interfaces [2]. Elucidating unique structures of protein interfaces may help us understand how proteins interact with each other and how they function. Even though, targeting protein interfaces for drug design is not easy [4], a unique set of interface structures may prove useful for protein interaction prediction, drug discovery, and repurposing studies [5].

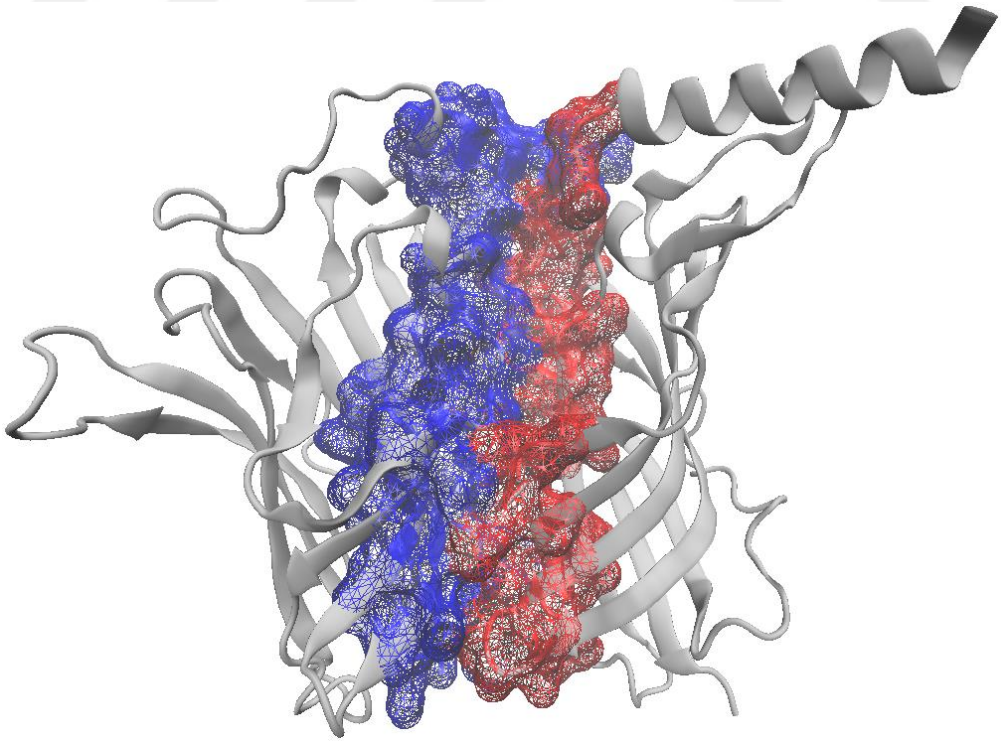


Figure 2.1 Protein and interface structure of 2WNJ chain A and B. Interface region is shown in blue and red for chains A and B, respectively.

Similar types of interactions take place in protein binding and folding; therefore, folding and binding are said to be similar processes [6]. The knowledge gained from protein-protein interactions may also help us with understanding protein folding mechanisms.

2.1.1 Properties of Protein Interfaces

Protein-protein interfaces form between two individual proteins when they establish a contact through their surface amino acids, under the effects of hydrophobic forces or complementary electrostatic interactions [7]. The average size of protein-protein interfaces is around 1200-2000Å [8]. The size of a protein interface affects its stability; it has been proposed that interfaces with sizes smaller than 1200Å are generally not stable [9].

A protein interface can be formed between identical or non-identical chains. If identical monomers form a protein complex, it is called a homooligomer. Furthermore, if each subunit is contacting through the same surface, they are named as isologous homooligomers. A complex structure with non-identical chains is termed as a heterooligomer. The same naming convention also holds for complexes with two chains: homodimers and heterodimers. Homodimers are the most common protein complex in nature [10]. It has been observed that homodimers have a larger surface area than heterodimers, they have fewer hydrogen bonds, and they are more hydrophobic [11].

Complex structures can be categorized as non-obligate/obligate complexes and transient/permanent complexes. In obligate complexes, the subunits of the complex are not stable as independent structures. A non-obligate complex is when subunits of a complex can also exist as separate stable structures [1]. Transient and permanent complexes are defined by the lifetime of the complex. Permanent complexes are generally stable, and subunits of permanent complexes usually only exist in complex structures. In contrast, transient complexes have a shorter lifetime [12]. The lifetime of a complex depends on the interaction strength, which is highly affected by the hydrophobic interactions, hydrogen bonds, salt bridges, and disulfide bonds that take part in forming the complex structure.

One of the strongest determinants in proteins' three-dimensional structure is the amino acid sequence of the protein [13]. Several studies have shown that two proteins with sequence identity higher than 35-40% are usually structurally similar [14, 15]. Also, the amino acid composition of proteins differs in interface regions than the rest of the structure [9, 16]. Some amino acids are less likely to be found on non-interface surfaces, especially hydrophobic residues, e.g., Met, Cys, Phe, Ile, Leu, and Val. They are located with a higher frequency on either interface surfaces or the protein cores. The opposite is observed for hydrophilic residues.

The secondary structure of proteins is an essential factor in how the chains interact with each other. Interfaces formed in homodimers or heterodimers show different preferences of secondary structure elements [17]. In homodimers, α-α interfaces, which is defined as interfaces with only alpha helices or coils in both chains, and α-β interfaces are more frequent than they are in heterocomplexes. Also, β-β interfaces are more frequent in heterocomplexes than they are in homodimers.

2.1.2 Definition of Protein-Protein Interfaces

Interface regions are formed by atoms on a protein surface where two different proteins interact to form a complex. There are several different interface definitions used in literature. Some of the most commonly used ones are presented here.

Heavy Atom Distance

A residue is defined as an interface residue if any heavy atom (non-hydrogen) of the residue is within a threshold distance from any heavy atom of the residue of the interacting chain. Threshold distances ranging from 4 to 6 Å are usually used [18, 19].

Atom-Atom Distance

A residue is defined as an interface residue if an atom of the residue is within a threshold distance from the interacting residue's atom. 6 Å is usually used as a threshold distance [12, 20].

CA-CA Distance

Two residues of different chains are defined as interacting residues if they have Cα atoms within a threshold distance. Different threshold distances are used, ranging from 8 Å to 12 Å [12, 21].

Van der Waals Distance

Two residues are defined as interface residues if they have atoms that are at most VDW radii of atoms plus a threshold distance away from each other. Threshold distance is usually defined as 0.5 Å [22-25]. Nearby or neighboring residues are also shown to be necessary for a more complete representation of the interface region. Nearby residues provide a supporting scaffold for contacting residues in interface regions [26]. A nearby distance threshold of 6 Å is used in previous studies. Any residue with a Cα atom within the distance threshold of a contacting residue atom is defined as a nearby residue [22, 23].

Delta Accessible Surface Area (ΔASA)

A residue is defined as an interface residue if its Accessible Surface Area (ASA) changes more than 1 Å between the individual state of the protein and when it is in a complex [16].

Voronoi Diagrams

Voronoi diagrams are a geometric approach. An interface surface formed by two proteins is represented as a subset of their Voronoi diagrams. This definition aims to propose a method for specifying the boundaries of interface regions while specifying interface residues. [27].

2.2 Protein Interface Databases

Worldwide Protein Data Bank (wwPDB) [28] is an organization that maintains the repository of the three-dimensional structures of proteins since 1971. wwPDB has four members: Research Collaboratory for Structural Bioinformatics Protein Database (RCSB PDB) [29], Protein Data Bank in Europe (PDBe) [30], Protein Data Bank in Japan (PDBj) [31], and Biological Magnetic Resonance Data Bank (BMRB) [32]. Members accept the structural data and process it; then the processed

data is stored on RCSB PDB so that there is a single version of collected data identical to all the users worldwide.

Several databases are built using the structural information about macromolecules available on PDB alone or in conjunction with other databases such as UniProt [33], SCOP [34, 35], Pfam [36], etc. These databases are important resources for scientists working on the structural and functional investigation of proteins. In this part, we present and summarize the functionalities of some of these databases that are recent and currently active.

2.2.1 PDBsum

PDBsum [37, 38] is a web server that provides image-based structural information on the structures available in the PDB. It shows the molecules that form the structure, including protein chains, DNA, ligands, and metal ions, and offers a schematic diagram about their interactions with each other. It is possible to visualize the structure with 3Dmol on the web server. PDBsum can be accessed from http://www.ebi.ac.uk/pdbsum.

Structures can be searched by their 4-character PDB code, by a text search that looks up TITLE, HEADER, COMPND, SOURCE, and AUTHOR records available in the PDB or by their sequence.

Information about a structure is presented on ten different tabs. Out of these ten tabs, the top page summarizes the structure regarding molecules it contains, GO functional assignments, and selected figures from key references. The protein tab shows topology diagrams by CATH domain, and if available, information about residue conservation. Prot-prot tab shows the schematic diagrams of any protein-protein interfaces that the structure includes and the residue-residue interactions occurring between them. The rest of the tabs contain information about DNA/RNA, ligands, clefts, pores, tunnels, and external links to the structure.

As of December 31, 2020, PDBsum contains 177,061 PDB entries, including superseded ones. One pitfall PDBsum has, as of this writing it does not process large structures.

2.2.2 PIFACE

PIFACE [22] is a data set of non-redundant unique protein-protein interface structures. It includes protein-protein interfaces from PDB, which are clustered according to their structural similarity. The interface definition of the data set depends on the distances between the atoms of two chains. The data set is being used as template for predicting protein-protein interactions on PRISM [24, 39]. PIFACE can be accessed from http://prism.ccbb.ku.edu.tr/piface/.

The data set can be used through a web server. Search options include an interface search and a domain search. Interface search can be used to list similar interfaces with a PDB ID, and chain IDs interface is located. Domain search can be used to search Pfam domains in an interface with a PDB ID as an input, or it can be used to find interfaces with a given Pfam ID. It is possible to compare two interfaces and generate PDB files for an interface through the web server. Cluster information is available to download in text format.

The search results include information about the size of interfaces and chains, accessible surface area (ASA), taxonomy information about chains, structural information like experimental method, and resolution for interface search. It also lists similar interfaces. Interface comparison search shows similarity and RMSD values for searched interfaces and also visualizes the structures with JSmol. Domain search shows the corresponding residues and Pfam IDs on each chain.

PIFACE database includes the structures in PDB until January 2013. It contains 130,209 interface structures coming from 39,196 complex structures. There are 22,604 clusters presented in the current data set.

2.2.3 ProtCID

ProtCID (Protein Common Interface Database) [40, 41] provides structural information about the interaction of proteins and individual protein domains with other molecules. These interactions include four different types: chain interfaces, Pfam domain interfaces, Pfam-peptide interfaces, and Pfam-ligand/nucleic acid interactions. The current version of the database is based on Pfam v32. ProtCID can be accessed from http://dunbrack2.fccc.edu/ProtCiD/.

ProtCID aims to identify and cluster homodimeric and heterodimeric interfaces seen in multiple crystal forms of homologous proteins and their interactions with peptides and ligands. The information on chain interface and domain interface clusters has a possible use case of identifying biological complexes, especially the ones like asymmetric homodimers with weak interactions.

The web server has a search functionality with the following input options: PDB code, Pfam ID or Pfam accession code, sequence, or UniProt IDs. If a single Pfam ID is given as an input, then the output is a list of PDB entries that contain the queried ID. If the input is two Pfam IDs, then the output is a list of PDB entries that include common domain-domain interactions. A similar search function also available for sequences as well.

The results of search queries include the number of crystal forms that contain a common interface, the number of PDB entries, the number of PDB and PISA biological assembly annotations that have the same interface, the average surface area, and the minimum sequence identity of proteins that have the interface.

ProtCID was updated currently on June 5, 2020. The current version includes 145,878 single-chain architectures in chain-based interface clusters and 133,130 Pfam domains in domain-based interface clusters.

2.2.4 3did

3did (Database of Three-Dimensional Interacting Domains) [42] provides structural templates for domain-domain interactions of high-resolution structures in PDB. It includes template information between globular domains and also domain-peptide interactions. The most recent version of the database is based on Pfam version 32.0. 3did can be accessed at https://3did.irbbarcelona.org/.

The database can be searched with the help of a web server. The search functionality includes the following options: domain name, Pfam access code, PDB ID, motif name, or a GO term. All of the data available on the webserver can be downloaded as flat files or MySQL tables. Data from older versions are also available for download.

Results provided for a search term includes a graph that shows the chains of the structure with Pfam domains, and interactions of domains between chains,

visualization of protein structure with Jmol, list of the domain architectures of each chain, which also gives detailed information about the location of the domain on the chain, and list of interactions that involve the given chain.

The current version of 3did uses PDB data from January 2020 and includes 573,127 structures with 14,278 domain-domain interactions. The database is being updated fairly regularly since 2011.

2.2.5 PDBbind

PDBbind [43] provides information about experimentally measured binding affinity data for available biomolecular complexes in PDB. The database's primary goal is to show the connection between energetic and structural information of the molecular complexes. It includes binding data about protein-ligand, nucleic acid-ligand, protein-nucleic acid, and protein-protein complexes. PDBbind can be accessed at http://www.pdbbind-cn.org/.

The basic information about complexes is available for access from the “browse” tab. Users can search for a complex with a PDB ID. The information included here is a visualization of the molecule by 3Dmol, type of complex, protein and ligand name, EC. Number, resolution of the deposited structure, binding affinity, release year, protein or nucleic acid sequence, primary reference related to the structure, and the links to the external databases. Additional search functions and downloading of the data are only available after registration, free for academic or industrial users.

The most recent version of the PDBbind database is released on January 12, 2020, based on the available PDB data until January 1, 2019. It includes binding data of 21,382 biomolecular complexes in total. The database is being updated yearly since 2007.

Chapter 3:
MATERIALS AND METHODS

This study aims to achieve two main objectives: identification and extraction of protein-protein interfaces and comparison of interfaces and interface chains in terms of sequence and structure. The first part of materials and methods covers a detailed explanation of interface identification and extraction process. The second part covers the methods for comparison of interface chains and interfaces. The main aim of sequence and structure comparison is to build a roughly non-redundant dataset of interface structures. An overview of the methods is given in Figure 3.1.

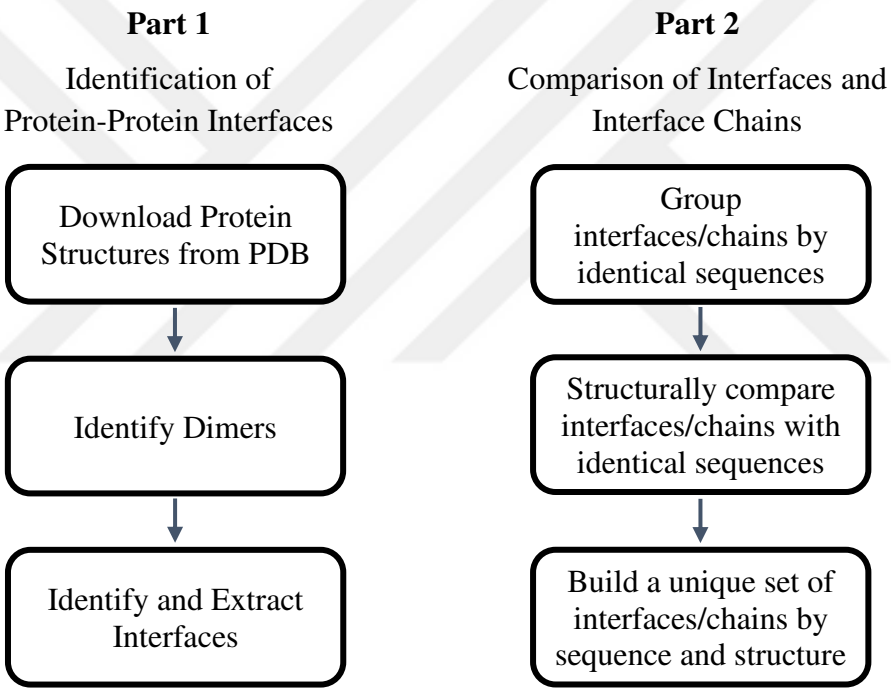


Figure 3.1 Overview of methods

3.1 Identification of Protein-Protein Interfaces from PDB

Identification and extraction of interface structures are made by an in-house software that is available in the COSBI GitHub (pifaceDataPrep, pifaceFunctions) repository. Codes are written in Python 3.8 [44], and Pandas [45], NumPy [46], SciPy [47], Biotite [48], Scikit-learn [49], and Sqlite3 [50] libraries are used to handle the data, calculations, and deposition of the data to the database. The software allows the user to build a new dataset or update an existing one easily

using terminal commands. All data created are deposited in a single SQLite database for structures. Data handling and calculations are done on Koç University High-Performance Computing cluster (KUACC). All functions are written with NumPy matrices to enable the use of matrix operations or list comprehensions to avoid for-loops for performance.

There is a simple naming convention for dimers and interfaces. This naming convention changed in this study due to the addition of multi-character chain IDs. In the previous studies [22, 23, 51], the naming convention was PDB ID plus the chain IDs, e.g., 1GQPAB. But since 2014, the increase in the chain count per structure caused the need for multi-character chain IDs. It is not possible to identify chain IDs for multi-character chain ID, using the previous naming convention. Therefore, in this study, we separated the PDB ID and chain IDs with an underscore ("_") character, e.g., 3J3Q_gU_iN.

All protein visualizations in this thesis are created with VMD [52] and PyMOL [53]. Statistical graphs are prepared with Matplotlib [54].

3.1.1 Downloading Molecule Structures from PDB

Research Collaboratory for Structural Bioinformatics (RCSB) PDB publishes a summary of all available entries in "txt" format, named "entries.idx", with every weekly update on Wednesday 00:00 UTC. This file holds the following information about each structure: PDB ID, header, accession date, compound, source, author list, resolution, and experiment type. Before downloading the structures, entries.idx file is downloaded, and information available in this file is parsed and deposited to the database. Keeping data from the entries table enables the user to analyze it with the available information when needed. The user does not need to download this file manually; it is an automatic process at the start of the run. If the dataset is a fresh build, then all available PDB IDs in the file are deposited; if the run is an update for an existing database, only the PDB IDs that are not deposited before gets evaluated. This update procedure is done by comparing two different Pandas data frames: one build with the information from already existing PDB IDs in the database and one from the newly downloaded entries.idx file. It is necessary to make a full comparison between these data frames using PDB IDs instead of

checking the structures by their accession dates. Accession dates for structures may differ from their release dates; this means a currently released PDB entry may have an older deposition date, resulting in missing those PDB IDs when the accession date is used. For example, PDB ID: 6YJB is deposited on April 04, 2020, but released for access on December 23, 2020.

FASTA files for each structure are also kept in a different table in the same database. FASTA files provide information about each chain's source organism in a structure, apart from their sequence information. This source organism information is necessary in organism-based studies, for example, if one only wants to work with interfaces that include a chain coming from viruses. Even though RCSB PDB advanced search option provides the possibility to list full structures according to their source organism (taxonomy) information, this is not enough for identifying which chain belongs to the organism of interest. FASTA table in the database helps with the accurate identification of organisms per chain.

All of the structures are downloaded in PDBx/mmCIF format, the standard PDB archive format since 2014. The importance of mmCIF format is that it can represent large structures (containing >62 chains and/or 99999 ATOM records), which cannot be represented with a single PDB file [55]. Before 2014 these large structures were represented in multiple PDB files, which hindered identifying interfaces that are not in the same file. Using mmCIF files enables us to identify all interfaces independent of the molecule's size; this becomes increasingly important as the structure size in PDB gets larger. After the download of structures, chain count and count of possible dimer combinations of each structure are deposited in a table called “cif” in the database. For the rest of the analysis and calculations, this structure is used in a single run. The full structure is not saved to the database because keeping the full PDB archive takes too much space, and the molecules' complete structures are not needed to be stored locally for interface studies.

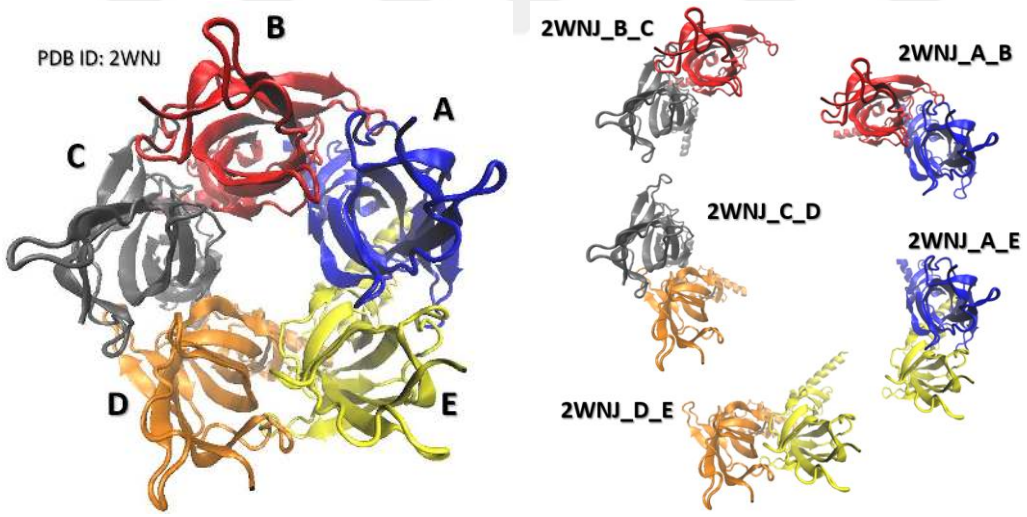
3.1.2 Identifying Possible Dimers

Distance-based approaches to identify interfaces are computationally costly. Therefore, methods are needed to decrease the number of comparisons without sacrificing accuracy or missing interface structures. We are using a Solvent

Accessible Surface Area (SASA) [56] based approach to identify monomer combinations that may be interacting through a contact region (Figure 3.2). SASA calculates a given molecule's surface area, which can be occupied by a probe with the volume of a water molecule. We use the following comparison to identify whether two molecules are in close enough proximity to interact:

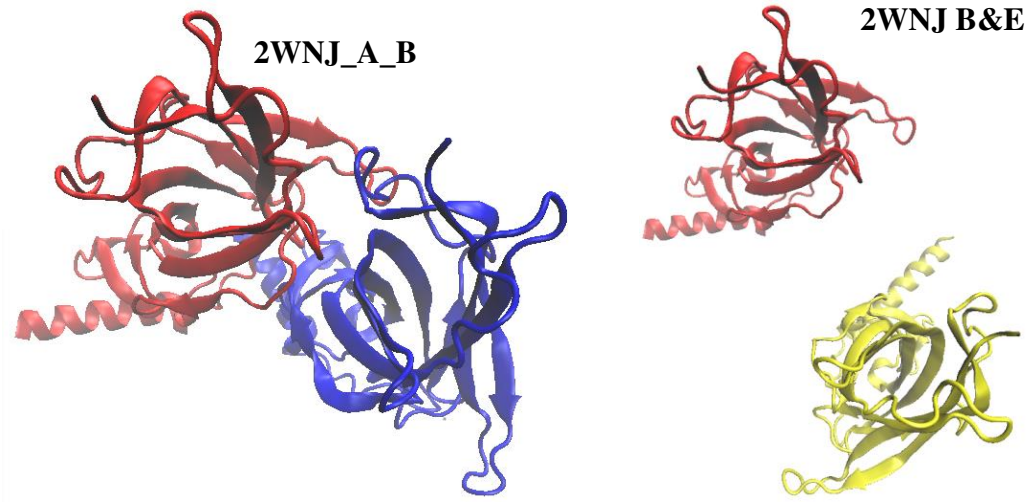
$$SASA(Monomer_1) + SASA(Monomer_2) \geq SASA(Dimer_{M1+M2}) + 1\text{Å}^2 \quad (Equation\ 3.1)$$

So, if the sum of SASA for monomers is more than at least 1 Å, we say that these two monomers are possibly forming an interface through their contacting residues (Figure 3.3). The user can change the SASA criterion with command-line arguments. SASA calculations are done using the "sasa" function in the Biotite package, which uses the Shrake-Rupley algorithm [57]. For structures with more than one model, only the first model is evaluated.



(a) (b)

Figure 3.2 Full structure and dimers for PDB ID: 2WNJ [3] a) Full protein structure. b) Dimers identified in the protein



(a) (b)

Figure 3.3 Two possible dimer combinations for SASA calculation in 2WNJ a) Chains A and B have a SASA difference of 2,783.9 Å, and the complex is identified as a dimer. b) Chains B and E have a SASA difference of 0 Å, and they are not identified as dimers.

Information about dimers is kept in two different tables in the database. "dimers" table contains information about monomer and dimer SASA values for each chain in a structure if the condition in Equation 3.1 holds. The "dimers_cif" table contains the contents of the mmCIF file for the dimer. These dimer files contain only the amino acids in the dimers. This filtering for amino acids is done with the help of the "filter_amino_acids" function in the Biotite package.

Too large structures in terms of chain count pose a problem of being computationally too costly. Without any elimination, if we check SASA values for each possible dimer combination, it sums up to $\frac{N*(N-1)}{2}$ combinations. While it is not too much for small molecules, it gets larger as the chain size increases. To decrease the number of SASA comparisons in large molecules, if the structure has more than 30 chains, we use the following approach:

1. We find the largest chain in the structure in terms of end-to-end distance (R). For this step, we use an algorithm called Quickhull. Quickhull is implemented in the SciPy package; it helps find the points on the convex

hull for a given 3D object. Convex hull is defined as the smallest set of convex that contains the given structure, in our case, the atoms of a chain.

2. We find the geometric center of each chain.
3. SASA difference for each chain pair is only calculated if the distance between their centers is smaller than $1.2 * R$.

For example, PDB ID: 3J3Q [58] has 1356 chains (Figure 3.4). 918,690 comparisons are necessary when comparing each chain with every other chain. This number of comparisons takes around 6-7 days on a single core. When we follow the approach mentioned above, the number of comparisons decreases to 26,371, and the SASA comparisons take approximately 3 hours.

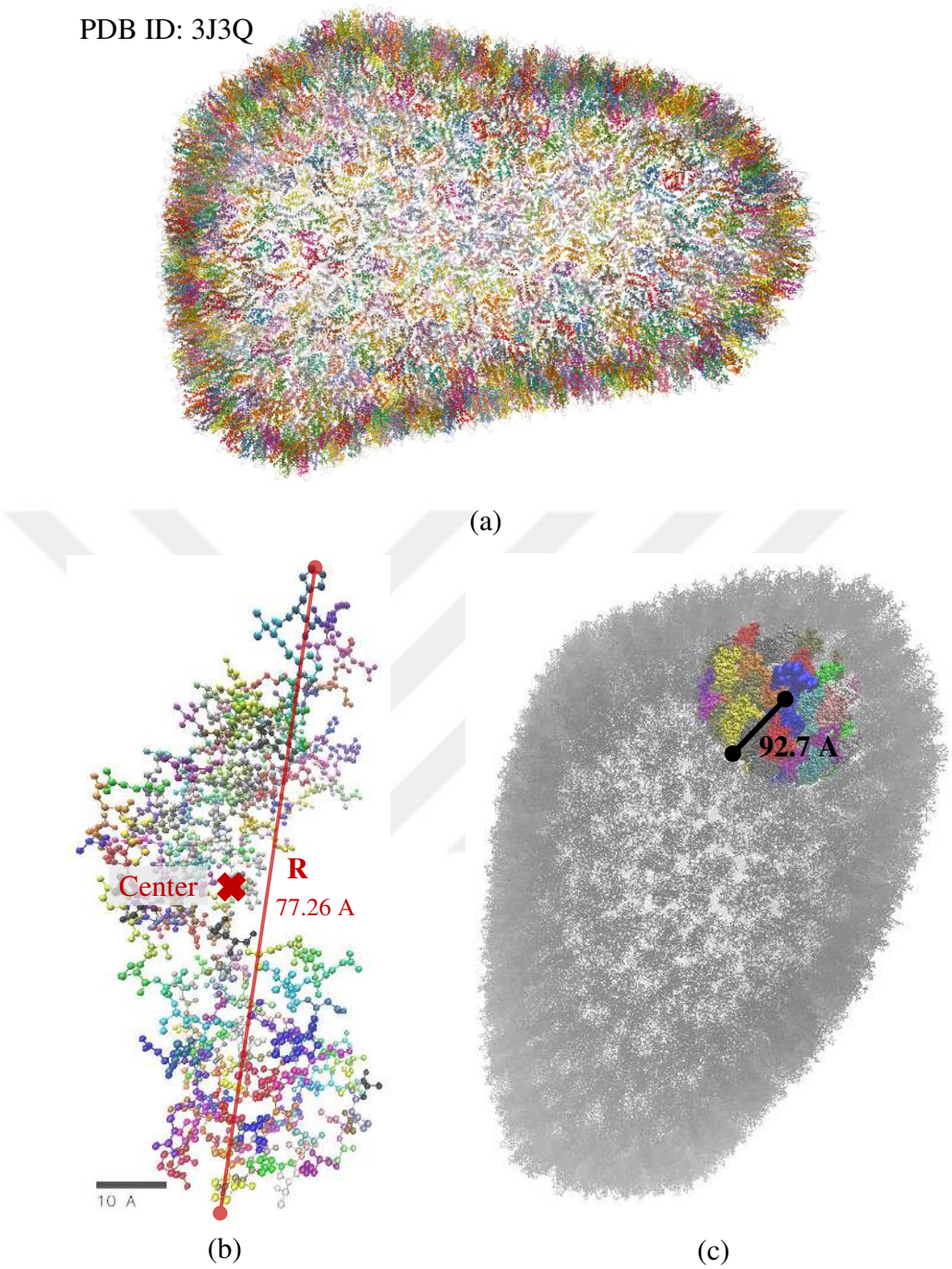


Figure 3.4 Example of distance filtering – 3J3Q a) Structure of 3J3Q. Each color represents a chain in cartoon representation. b) Chain A of 3J3Q and representative view of the end-to-end distance (R) and geometric center of the chain. Each color represents a different residue in CPK representation. c) Approximate view of the area left to compare SASA differences after filtering by distance. Blue VDW representation with larger spheres is chain A, and different colors represent different chains. Chains represented with grey are not compared with chain A.

3.1.3 Identifying Interfaces Between Dimers

In this study, protein-protein interfaces are defined using the distance between two atoms of the residues from two chains of a protein. Suppose the distance between two atoms of different chains is less than the sum of their corresponding Van der Waals (VDW) radii plus 0.5 Å (Equation 3.1). In that case, they are said to be "contacting residues." Any residue with a Cα atom within 6 Å distance (Equation 3.2) from any atom of a contacting residue is termed as a "nearby residue" [23] (Figure 3.5).

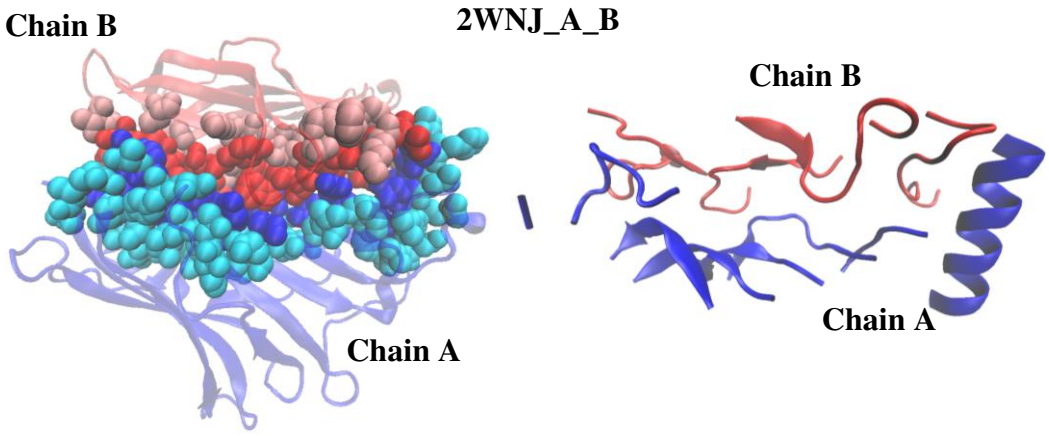
Where $Atom_A$ is an atom from chain A and $Atom_B$ is from chain B, and $Atom_{A-VDW}$ and $Atom_{B-VDW}$ denotes the VDW radii for given atoms in chain A and B, respectively, contacting residues can be easily formulized as:

$$Distance(Atom_A, Atom_B) \leq Atom_{A-VDW} + Atom_{B-VDW} + 0.5 \quad (Equation\ 3.1)$$

Where $Atom_{Contacting}$ is an atom from a contacting residue in a chain and $Residue_{C\alpha}$ is any Cα atom in the same chain, nearby residues can be formulized as:

$$Distance\left(Atom_{Contacting}, Residue_{C\alpha}\right) \leq 6\ \text{Å} \quad (Equation\ 3.2)$$

Another criterion necessary to define an interaction as an interface is that both interface chains should have at least five contacting residues [22]. VDW range for contacting residues, the distance for nearby residues, and the minimum number of contacting residues can be changed by the user with command line arguments when running the code. Van der Waals radii for distance calculation between two atoms are taken from the CHARMM (Chemistry at HARvard Macromolecular Mechanics) force field [59] definitions for VDW parameters (Supplementary Table 2). The VDW parameters are different in this study than the previous one by Çukuroğlu et al. (2014). These parameters are also the ones used by Keskin et al. (2004).



(a) (b)

Figure 3.5 Interface Representation for 2WNJ_A_B a) "blue/red – VDW" representations show the contacting residues in chain A and B; "light blue/pink – VDW" representations show nearby residues for chain A and B, respectively. b) Structure of interface. "blue - new cartoon" represents chain A, and "red - new cartoon" represents chain B.

Sometimes PDB structures include some atoms that are not defined in CHARMM definitions or followed by digits that are not in the definitions (H13, H14, etc.). For these cases, we have two default dictionaries built by the largest possible value for a given atom in CHARMM definitions, irrespective of which amino acid they belong to. The first dictionary is with the trailing digits (Supplementary Table 3). A given atom is first searched in the dictionary with the trailing digits. If no match is found in the first dictionary, the second dictionary has atom names without trailing digits (Supplementary Table 4). The trailing digits are also removed from the name of the given atom and searched in this dictionary. The aim of choosing the largest value for a given atom is to include the maximum number of atoms and not risk missing an interface with a small margin. Both CHARMM values and default dictionary values are given in Appendix A as supplementary tables.

Python for-loops are inherently slow; to overcome this, using NumPy arrays and intrinsic operations increases performance significantly. Identification of residues within the contacting range between two chains is made by comparing two

NumPy matrices to benefit from the performance increase that comes with them. The first matrix contains the VDW radii plus 0.5 for every possible pair in two chains, “critdist_matrix” (Equation 3.3). The second matrix has the real distances between every possible pair, “realdist_matrix” (Equation 3.5). Real distance is defined as the Euclidean distance between two atoms (Equation 3.4). Euclidean distance calculation is available in the SciPy library.

$Chain_1$ and $Chain_2$ denotes two chains of a possible interface where $a_1, \dots, a_j$ and $b_1, \dots, b_j$ denotes the atoms of $Chain_1$ and $Chain_2$:

$$Chain_1 = [a_1, a_2, a_3, \dots, a_i]$$
$$Chain_2 = [b_1, b_2, b_3, \dots, b_j]$$

$$critdist\_matrix = \begin{bmatrix} a_{1_{vdw}} + b_{1_{vdw}} + 0.5 & a_{1_{vdw}} + b_{2_{vdw}} + 0.5 & \cdots & a_{1_{vdw}} + b_{j_{vdw}} + 0.5 \\ a_{2_{vdw}} + b_{1_{vdw}} + 0.5 & a_{2_{vdw}} + b_{2_{vdw}} + 0.5 & \dots & a_{2_{vdw}} + b_{j_{vdw}} + 0.5 \\ \vdots & \vdots & \ddots & \vdots \\ a_{i_{vdw}} + b_{1_{vdw}} + 0.5 & a_{i_{vdw}} + b_{2_{vdw}} + 0.5 & \dots & a_{i_{vdw}} + b_{j_{vdw}} + 0.5 \end{bmatrix}_{i \times j}$$

(*Equation* 3.3)

If a_i has coordinates (x_1, y_1, z_1) and b_j has coordinates (x_2, y_2, z_2) then;

$$Eucledian\ distance\ a_i, b_j \rightarrow dist(a_j, b_j) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

(*Equation* 3.4)

$$realdist\_matrix = \begin{bmatrix} dist(a_1, b_1) & dist(a_1, b_2) & \cdots & dist(a_1, b_j) \\ dist(a_2, b_1) & dist(a_2, b_2) & \dots & dist(a_2, b_j) \\ \vdots & \vdots & \ddots & \vdots \\ dist(a_i, b_1) & dist(a_i, b_2) & \dots & dist(a_i, b_j) \end{bmatrix}_{i \times j}$$

(*Equation* 3.5)

NumPy returns a Boolean matrix, a matrix of True and False values, as a result of the $realdist\_matrix \leq critdist\_matrix$ comparison. It checks whether the condition holds elementwise. Contacting atoms and residues are selected as explained in the pseudo-code below:

```
1  calculate elements of critdist_matrix and realdist_matrix
2  compare critdist_matrix with realdist_matrix elementwise
3  if realdist_matrix(i, j)≤critdist_matrix(i, j):
         comparison_matrix(i, j) = True
   else: comparison_matrix(i, j) = False
4  if a row (i) in comparison_matrix has True in any column (j):
         then mark the atom of Chain1 with the index corresponding
         to that row as contacting
5  if a column (j) in comparison_matrix has True in any row (i):
         then mark the atom of Chain2 with the index corresponding
         to that column as contacting
7  get residue ids of the atoms that are marked as contacting
8  return arrray(residue ids)
```

Indexes are identified as the contacting atoms, and the residues these atoms belong to are identified as contacting residues.

Identification of nearby residues is done by first filtering Cα atoms in each chain. Then Euclidean distance between contacting atoms in a chain and every Cα atom in that chain are calculated (Equation 3.6).

$Chain_{Contacting}$ denotes contacting atoms in a chain and $Chain_{C\alpha}$ denotes all Cα atoms in a chain. dist_contacting_matrix is the matrix of distances between every contacting atom and every Cα in a chain.

$$Chain_{Contacting} = [a_{c_1}, a_{c_2}, .. a_{c_i}]$$

$$Chain_{\mathrm{C}\alpha} = [c_1, c_2, \dots, c_j]$$

$$dist\_contacting\_matrix = \begin{bmatrix} dist(a_{c_1}, c_1) & dist(a_{c_1}, c_2) & \cdots & dist(a_{c_1}, c_j) \\ dist(a_{c_2}, c_1) & dist(a_{c_2}, c_2) & \dots & dist(a_{c_2}, c_j) \\ \vdots & \vdots & \ddots & \vdots \\ dist(a_{c_i}, c_1) & dist(a_{c_i}, c_2) & \dots & dist(a_{c_i}, c_j) \end{bmatrix}_{i \times j}$$

(*Equation* 3.6)

For every row and column, the $dist\_contacting\_matrix \leq 6$ Å condition holds, an atom at the given index (c_i) is identified as a nearby atom if it is not in a contacting residue itself, and the residue this atom belongs to is identified as a nearby residue. Nearby residues are selected as explained in the pseudo-code below:

```
1  calculate elements of dist_contacting_matrix
2  if dist_contacting_matrix(i, j) ≤ 6:
3        comparison_matrix(i, j) = True
   else: comparison_matrix(i, j) = False
5  if a column (j) in comparison_matrix has True in any row (i):
         then mark the atom of Chain with the index corresponding
         to that column as nearby
6  get residue ids of the atoms that are marked as nearby
7  return array(residue ids)
```

A challenge we encountered while identifying interfaces in dimers is with structures that have too many atoms in a single chain. In this case, it is not possible to keep the distance matrices in memory since they get too large too quickly. To prevent crashing the calculations due to a memory overuse, the approximate size in bytes each matrix may become is calculated with the following code:

```
~size of matrix in bytes =
     np.float64(1).itemsize *
     np.prod([# of atoms in Chain₁ * # of atoms in Chain₂])
```

To explain this code, np.float64(1) creates a single instance of a float64 element, and .itemsize is an attribute for an element with size information in bytes. np.prod calculates the product of array elements. Therefore, this code returns the approximate size of the NumPy matrix with the given dimensions; in this case, atom counts of a dimer in two opposing chains.

If the approximate size of a matrix is bigger than one-third of the given value for memory in bytes, then an HDF5 file is used to keep matrices in disk instead of memory. Although this approach sacrifices some performance, it enables evaluating large chains, which would not be possible normally. NumPy arrays and matrices can directly be written and read from HDF5 files. This file is discarded after the evaluation of the interface is done. One example of such structures is PDB ID:

6WOU [60]. It has a total of 125,244 atoms (Figure 3.6), 59,780 atoms in Chain A and 59,779 atoms in Chain B. To identify the interface region between Chain A and B (Figure 3.7), we need two matrices with a size of 59780 x 59779, which results in 3,573,588,620 elements for a single matrix of size around 26.6 GB. This big of a matrix would not fit in a reasonable size memory. Therefore, an HDF5 file is used to identify the interface region for this dimer.

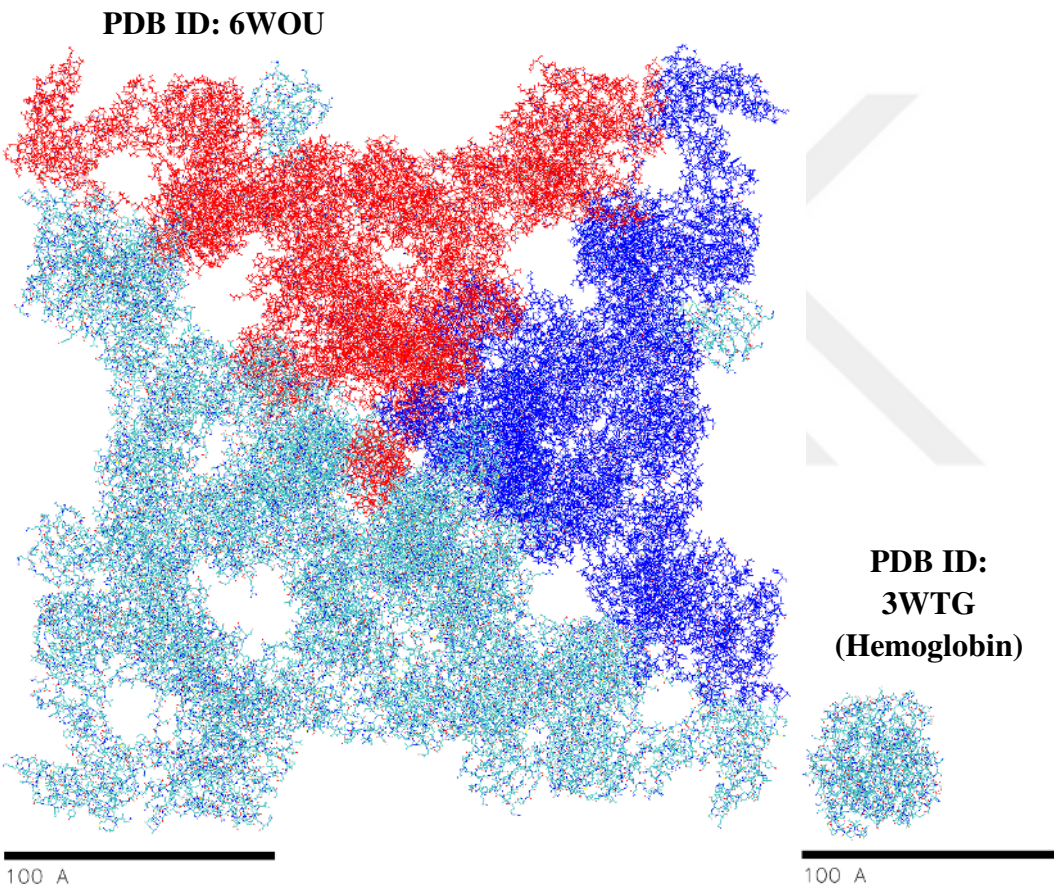


Figure 3.6 Lines representation for PDB ID: 6WOU. Chain A and B are represented with blue and red, respectively. PDB ID: 3WTG is for scale.

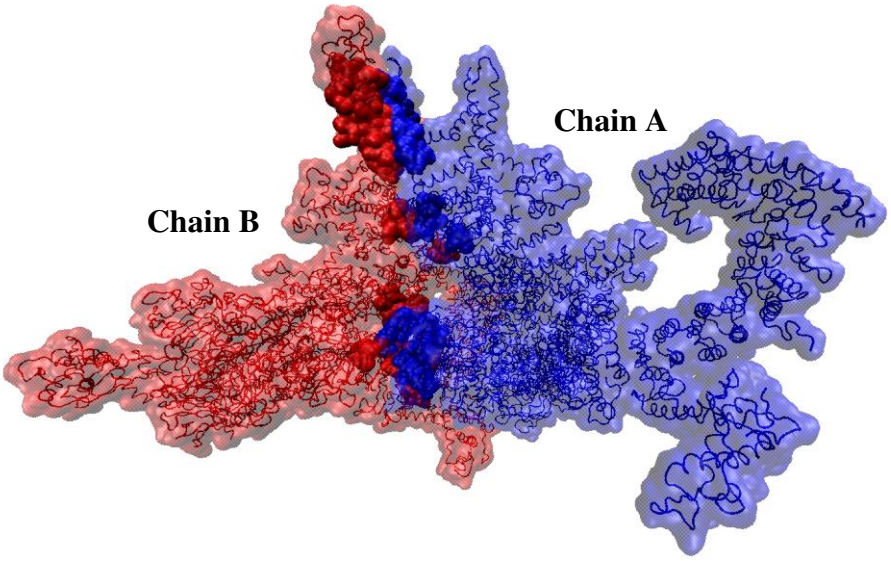


Figure 3.7 Interface ID: 6WOU_A_B. Interface region represented with VDW representation. The rest of the chains are represented with surface and cartoon representation.

Information about interfaces is stored in three different tables in the database. The “interfaces” table holds the information about residue ids in each chain belonging to contacting residues and nearby residues separately for further analysis. The “interfaces_cif” table stores the mmCIF file for the interface region only as a single structure, and “interfaces_seq” keeps the information about residue sequences of interfaces.

3.1.4 Database Design for Storing Structure Information

A single SQLite database [50] named PIFace is used to store all information related to protein, dimer, and interface structures. SQLite is a small, self-contained, and full-featured relational database management system. It is not a client-server database engine, which means that it does not need a separate server running, and this makes it suitable to use directly on the HPC cluster at Koç University. It is an on-disk database that is stored in a single ordinary disk file; this allows for easy transfer of the whole database with a simple copy operation.

While deciding the tables and contents of tables, the priority was to keep all necessary information needed for future analysis and studies without going over the structural files again. Even though this approach increases the database's size (231

GB as of December 25, 2020), I believe these data will prove useful in the upcoming studies. The list of tables is given in Table 3.1, and the database schema is shown in Figure 3.8.

Table 3.1 List of PIFace DB Tables

Tables in PIFace Database
entries
fasta
cif
dimers
dimers_cif
interfaces
interface_cif
interfaces_seq

Figure 3.8 PIFace DB Schema - Light blue highlighted rows denote the unique keys.

3.1.5 Workflow and Code Overview for Interface Identification

Interface identification code flow can be divided into three steps: downloading 3D molecular structures, identifying dimers from 3D structures, and identifying interfaces from dimers.

Figure 3.9 shows an overview of the workflow. For a full (including all current structures in PBD) and new interface database, the user does not need to give any input. If an existing database will be updated and the database has a different name than the default (piface.db) or in a different path than the current directory, then the database's absolute path should be given.

In an update for a current database, the code checks if the database is set-up correctly by checking table names, columns, and indexes. If the database structure does not match the needed structure, the user gets an error about the difference, and the run is terminated. If it is a new database, then the tables are created automatically.

At the beginning of the code, the user is asked if they are in an existing or clean PIFace database directory; this is intended for the user to check one last time that they are starting the code in the correct directory. This question can be answered automatically by passing "--yes" as a command-line argument. This automatic pass is necessary for systems where user input cannot be provided interactively.

At the beginning of the run, a log file named "current_config.log" is created, containing the information about parameters (variables) to run the code. This file makes sure that calculations continue with the same parameters in case of an unexpected termination of the code. If this file exists and the code starts automatically following a stop, then the "--restart" flag sets by default and checks for previous progress and parameters before starting. It is also possible to set --restart manually. Progress can be checked by terminal output or system created output file. At the end of the run, "current_config.log" becomes renamed into "{date}_config.log".

The user can provide almost all parameters necessary throughout the code if a different value than the default is needed. Arguments that can be provided by the user, their default values, and definitions are given in Table 3.2.

Table 3.2 Possible user arguments for interface identification code. Arguments with a grey background color are mutually exclusive. Arguments with an orange background color may change the resulting interfaces, both in terms of interface size and containing PDB IDs.

Argument	Short Notation	Default Value	Description
--restart	-r	False	Whether the run is a restart or not. If it is an automatic restart (e.g., cluster failure, auto resubmission), it is set by default
--new	-n	False	Whether the run is a new PIFace build or an update
--error	-e	False	Try to run for PDB IDs that have a processed code = 2
--current	-c	False	Run for current PDB IDs that have a processed code = 0
--file	-f	False	Provide a file with PDB IDs instead of downloading the latest entries.idx
--prefix	-p	Current Directory	Run the program in a different directory than the default
--yes	-y	False	User input for whether the run is starting in the correct directory
--database	-db	piface	Give a different name than the default to the database
--sasa	-S	1.0	Assign custom SASA difference threshold for dimer identification.
--vdw	-V	0.5	Assign custom VDW distance threshold for identification of contacting residues
--mincontacting	-M	5	Assign a minimum number of residues per chain for interfaces
--nearby	-N	6.0	Assign custom distance value for identification of nearby residues
--memory	-m	16.0	Maximum amount of memory (RAM) usage by the program

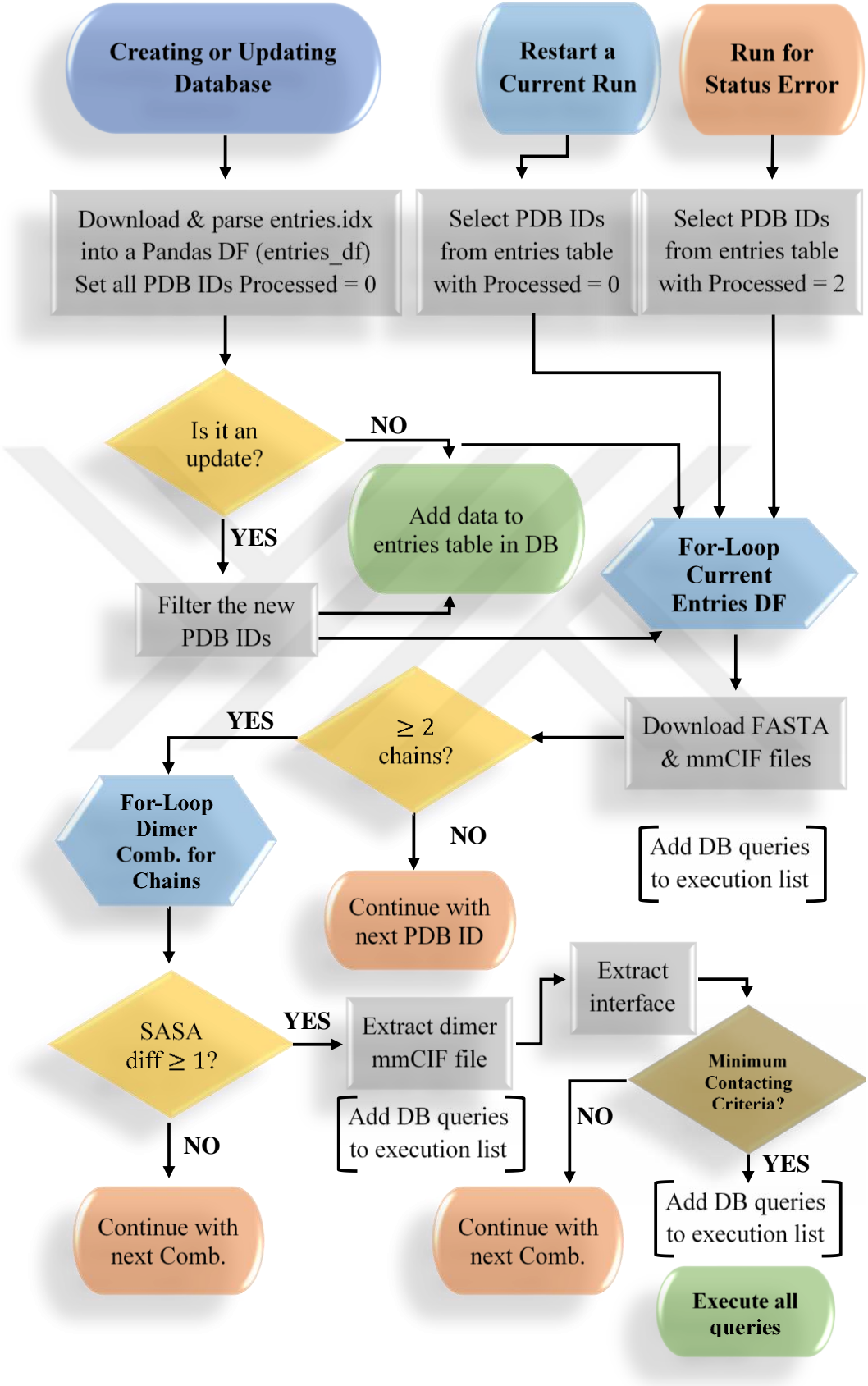


Figure 3.9 Interface identification and extraction code flow overview

The code adds every SQL (DB) query to be executed into a list first, without executing, until the iteration of the for-loop is finished for a given PDB ID. If the iteration's execution is finalized successfully, then the SQL queries listed are executed, and the data is added to the database. After the iteration of a PDB ID is complete, the “Processed” column's value in the entries table becomes updated to “1”. Possible errors are handled with a try/except block. Suppose an error occurs at any point during the iteration. In that case, the query list built until that point is discarded, PDB ID is added to the error log with available traceback information, and the “Processed” column in the entries table for that PDB ID is set to “2”, denoting an error. The “Processed” column helps with restarts and error runs. When a restart is necessary, current entries data frame is built by checking for PDB IDs with processed is 0. Likewise, if a run is needed for PDB IDs that previously had an error, PDB IDs with processed is “2” are selected. Table 3.3 shows the list of possible codes for the “Processed” column in the entries table. Supplementary Code 1 in Appendix B shows a set of example SQL queries for insert and update operations.

Table 3.3 Possible codes and definitions in entries table for “Processed” column

Code	Definition
0	Not Yet Processed
1	Successfully Processed
2	Process Failed

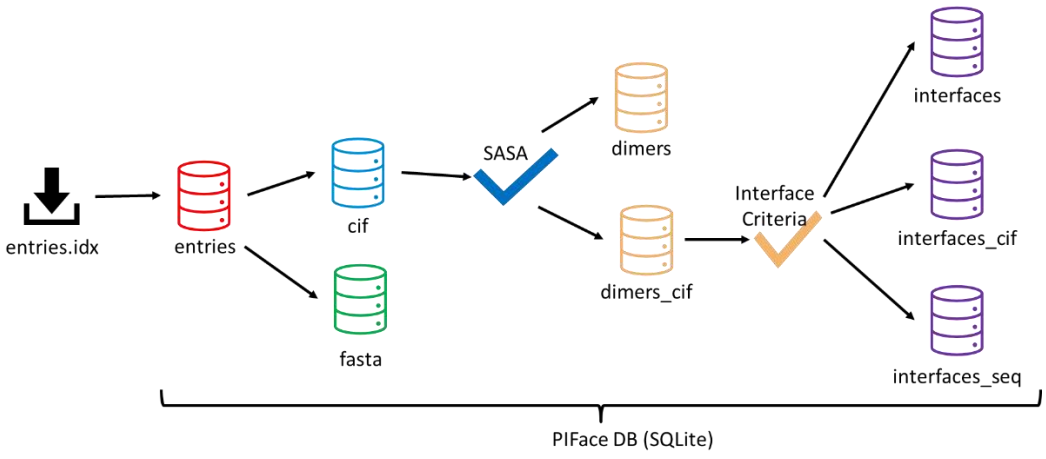


Figure 3.10 Flow and Storage Overview for Comparison of Interfaces

3.2 Comparison of Interfaces and Interface Chains

In this study, we compared interfaces both in terms of sequence and structure. Sequence comparison is made for finding identical interfaces available in PDB. RCSB PDB provides chain-wise sequence clusters with each weekly deposition for different threshold values (100%, 95%, 90%, 70%, 50%, 40%, 30%). This data can be downloaded from RCSB PDB, but chain-wise sequence clusters are fairly different than interface sequence clusters. Since interfaces cover a smaller, and possibly more conserved, region on proteins then chains, comparing interface sequences proved to be useful for getting a more focused clustering. Also, identical or similar chain sequences may not yield similar interface sequences due to various interface regions' positioning on chains.

Finding only sequence clusters by similarity, even if identical, is not enough to conclude that two interfaces are structurally similar. Identical sequences may yield different structural folds in proteins [61], which is also true for interfaces. Therefore, a structural similarity measure is a must before concluding that two interfaces are structurally the same. In this study, we used MM-align [62] for comparison of complete interface structures (interface structure with both chains) and TM-align [63] for comparison of one-side (single chain) of interfaces.

3.2.1 Interface Sequence Comparisons

We have compared two different types of sequences for this study. First, we compared complete interface sequences, and secondly, we compared single-chain sequences from interfaces. Since we are only looking for 100% similar sequences, we used a direct string comparison with clean sequence data. Data of residue sequences are collected directly from identified interfaces. For increased performance, all sequences are collected in a Pandas data frame for batch comparison. A pivot table detects interfaces with the same sequences; with indexes set as sequences and values as a list of interface IDs with those sequences. In the case of interface comparisons, chains are delimited with a vertical bar (“|”) to show the location where two chains separate. Table 3.4 shows an example of interface sequence representation.

We have implemented three different methods to factor in segmentation of sequences during comparisons since interfaces may not be continuous; without any residues that are not on the interface (non-spaced), all residues that are not on the interface replaced with the same number of underscore characters (multi-spaced), all residues that are not on the interface replaces with a single underscore character (single-spaced). After comparing these three options, we concluded that the most appropriate way to represent chain and interface sequences is to use the single-spaced representation. The comparisons of three different options on chain sequences can be found in section 4.3.1.

Table 3.4 Example of an interface sequence representation.

Dimer ID: 2WNJ_A_B
Single-Spaced
LEU,HIS,SER,GLN,ALA,ASN,LEU,MET,ARG,LEU,LYS,SER,ASP,LEU,PHE, _,PHE,THR,LEU,GLN,ASP,ILE,VAL,LYS,ALA,_,VAL,ASP,LEU,VAL,TYR, TYR,GLU,GLN,_,PHE,ARG,THR,SER,ALA,_,VAL,GLN,VAL,LEU,SER,PRO, GLN,ILE,ALA,VAL,_,PHE,ILE,PRO,ALA,GLN,ARG,LEU,_,SER,TYR,TYR, ALA,SER,SER,LYS,TYR,_,GLU,ARG|SER,PRO,MET,TYR,PRO,GLY,PRO, THR,LYS,ASP,ASP,PRO,_,ASP,SER,SER,THR,ASN,GLU,VAL,_,PRO,ASP, ILE,THR,ALA,TYR,SER,SER,THR,ARG,PRO,VAL,_,SER,PHE,MET,CYS, ASP,PRO,THR,_,CYS,_,LYS,PHE,GLY,SER,TRP,VAL,TYR,SER,GLY,PHE, GLU,ILE

Chains are named as dimer ID and chain ID delimited with a hashtag (“#”) character.

Table 3.5 shows an example of possible options for a chain sequence segmentation.

Table 3.5 Example set of possible segmentation options for a chain.

Dimer ID + Chain ID: 2WNJ_A_B#A
Non-Spaced
LEU,HIS,SER,GLN,ALA,ASN,LEU,MET,ARG,LEU,LYS,SER,ASP,LEU,PHE, PHE,THR,LEU,GLN,ASP,ILE,VAL,LYS,ALA,VAL,ASP,LEU,VAL,TYR,TYR, GLU,GLN,PHE,ARG,THR,SER,ALA,VAL,GLN,VAL,LEU,SER,PRO,GLN,ILE, ALA,VAL,PHE,ILE,PRO,ALA,GLN,ARG,LEU,SER,TYR,TYR,ALA,SER,SER, LYS,TYR,GLU,ARG
Single-Spaced
LEU,HIS,SER,GLN,ALA,ASN,LEU,MET,ARG,LEU,LYS,SER,ASP,LEU,PHE, _,PHE,THR,LEU,GLN,ASP,ILE,VAL,LYS,ALA,_,VAL,ASP,LEU,VAL,TYR, TYR,GLU,GLN,_,PHE,ARG,THR,SER,ALA,_,VAL,GLN,VAL,LEU,SER,PRO, GLN,ILE,ALA,VAL,_,PHE,ILE,PRO,ALA,GLN,ARG,LEU,_,SER,TYR,TYR, ALA,SER,SER,LYS,TYR,_,GLU,ARG
Multi-Spaced
LEU,HIS,SER,GLN,ALA,ASN,LEU,MET,ARG,LEU,LYS,SER,ASP,LEU,PHE, _,PHE,THR,LEU,GLN,ASP, ILE,VAL,LYS,ALA,_,_,_,_,_,_,VAL,ASP,LEU,VAL,TYR,TYR,GLU,GLN, _,PHE,ARG,THR,SER,ALA, _,VAL,GLN,VAL,LEU,SER,PRO,GLN, ILE,ALA,VAL,_,_,_,_,_,_,_,_,_,PHE,ILE,PRO,ALA,GLN,ARG,LEU,_,_, _, _,_,_,_,_,_,_,_,_,_,_,_,_,SER,TYR,TYR,ALA,SER,SER,LYS,TYR,_,_,_, _,GLU, ARG

Sequence comparison of interfaces also requires some possible switching with the placement of chains. Considering two possible chain orders make it possible to match all identical interfaces irrespective of their placements. An example of both possibilities for an interface is given in Table 3.6. Supplementary Code 2A in Appendix B shows the Python code for selecting comparisons, starting TM-align (or MM-align), reading results from TM-align or MM-align, and depositing results to MySQL database on a remote server. Supplementary Code 2B shows a part of the input file.

For chain comparisons and interface comparisons, the grouping of dimer IDs with the same sequences is done with the same steps over the created Pandas data frame. Steps of grouping identical sequence IDs are as follows:

1. Group the IDs by their sequences so that sequences become the indexes and the IDs become the values in a list.
2. Assign group IDs to each list. A single ID (chain or interface) may have more than one group.
3. Create a new data frame with two columns; chain or interface IDs and a list of the groups the given ID is in.
4. Find group members for each ID by checking with other IDs and add members to a new column.
5. Drop duplicated columns to find unique sequence groups.

Table 3.6 Possible chain orders for interface ID: "6AZM_A_E"
C1: Chain 1 / C2: Chain 2

Sequence	Chain Order
SER,SER,GLN,SER,VAL,LEU,TYR,SER,SER,ASN,ASN, LYS,ASN,TYR,LEU, _ ,TYR,TRP,ALA,SER,THR, _ , GLY,THR, _ ,GLN,TYR,TYR|ALA,ASN,PRO,ASN,ALA, ASN,PRO	C1 – C2
ALA,ASN,PRO,ASN,ALA,ASN,PRO|SER,SER,GLN,SER, VAL,LEU,TYR,SER,SER,ASN,ASN,LYS,ASN,TYR,LEU, _ ,TYR,TRP,ALA,SER,THR, _ ,GLY,THR, _ ,GLN, TYR,TYR	C2 – C1

3.2.2 Interface Structure Comparison

We compared both sequentially identical chains and interfaces structurally within identity groups. The comparison of a single chain of interfaces is made with TM-align. The comparison of complete interface structures is made with MM-align. There are $\frac{N*(N-1)}{2}$ comparisons for each group with N interfaces. Table 3.7 shows the members of an example group and structural comparisons done within the group.

Table 3.7 Example group and set of comparisons.

Group Members:	5O4I_B_C#C [64], 5O4Q_B_C#C [65], 6FC7_E_F#F [66]
Comparisons Within Group	
Chain 1	Chain 2
5O4I_B_C#C	5O4Q_B_C#C
5O4I_B_C#C	6FC7_E_F#F
5O4Q_B_C#C	6FC7_E_F#F

A Python code is used to build comparison lists for each group and start TM-align or MM-align programs for chains or interfaces. The input for the code is the list of groups with dimer IDs. Each line is a new group, and dimer IDs in groups are comma-separated. GNU Parallel [67] is used for parallel execution of TM-align and MM-align runs. Alignment results are read and parsed in batches. Insertion of results into MySQL database is done with a single insert query. The maximum number of inserts in a single execution is set at 10,000; the user can change this limit, but it is not recommended for the best performance. Twelve million chain comparisons with TM-align on 8 cores with a single driver code running takes around three days. This performance may be further increased by collecting smaller batches in a larger one. Full comparison code (comparison of all interfaces in PDB with each other, not only sequence groups) is running with a collection of smaller batches approach due to highly increased comparison numbers.

3.2.3 Database Design for Results of Chain and Interface Comparisons

Compared interfaces and comparison results are kept in a MySQL database on the COSBI Interactome server. MariaDB version 5.5 is used as an RDBMS on Interactome. Comparison Python code is written so that resulting data is directly deposited to the database on our server. Python and MySQL connection is established using SQLAlchemy [68]. Unlike SQLite, MySQL is a server/client database, which means that a server is necessary for MySQL to work; this allows MySQL to have multiple connections and write operations concurrently. Also, it

makes it possible to access the database remotely, which is necessary for this study since the calculations are done on KUACC, and data is kept on a different server. MySQL is better suited to support big data than SQLite since it is easier to scale securely. Table 3.8 shows the list of tables currently in the Comparisons DB. “interfaces” table keeps the interface IDs that have already been compared. This table is used for updates. New interface IDs are identified by checking from this table, and unnecessary multiple comparisons are prevented.

“interface_comparison_seq” keeps the results of interface comparisons with identical sequences. “chain_comparison_seq” holds the results for comparisons of chains with identical sequences, respectively. Figure 3.11 shows the database schema.

Table 3.8 List of Comparisons DB Tables

Tables in PIFace Database
interfaces
chains
interface_comparison_seq
chain_comparison_seq

interfaces | chains		
id	bigint(20)	auto_inc.
dimer_id	varchar(16)	
group_id	int(11)	
processed	int(11)	
insert_time	datetime	

comparison_seq | chain_comparison_seq		
cid	bigint(20)	auto_inc.
interface_1	varchar(32)	
interface_2	varchar(32)	
group_id	int(11)	
aligned_length	int(11)	
rmsd	float	
sequence_identity	float	
tmscore_int1	float	
tmscore_int2	float	
aligned_residues	text	
t0	float(25,10)	
t1	float(25,10)	
t2	float(25,10)	
u00	float(25,10)	
u01	float(25,10)	
u02	float(25,10)	
u11	float(25,10)	
u12	float(25,10)	
u20	float(25,10)	
u21	float(25,10)	
u22	float(25,10)	
processed	int(11)	
insert_time	datetime	

Figure 3.11 Comparisons DB Schema – Light blue highlighted rows denote the unique keys. interface_1 and interface_2 is unique as a pair.

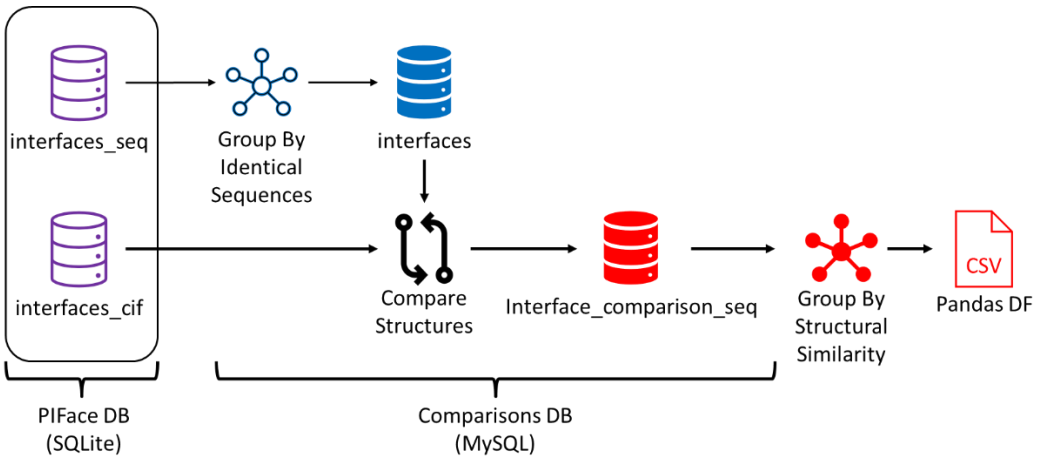


Figure 3.12 Flow and Storage Overview for Comparison of Interfaces

3.2.4 Finding Structural Groups and Selecting Representatives

Comparisons of interfaces or interface chains with TM-align or MM-align programs result in several metrics to compare two structures. These metrics are aligned length, aligned residues, sequence identity, RMSD, TM-Score 1 and TM-Score 2. Since our comparisons are within sequentially identical structures, aligned length, aligned residues, and sequence identity metrics are usually the same, as these metrics are related to sequence similarity. To evaluate structures based on structure similarity, we needed to choose TM-score or RMSD as a similarity metric. TM-score is a metric sensitive to the global topology than the local variations of a structure [69]. Also, the TM-score value is normalized so that the structures' size does not affect the metric. TM-score has a value between 0 and 1. A value equal or less than 0.17 indicates that structures are random; a value equal or more than 0.5 shows resemblance between two compared structures. Even though TM-score is a useful metric for comparing varied structures, in our case, local variations of a structure are also significant. Besides, since we compare structures with identical sequences, they have the same length, to begin with. Therefore, we decided to use RMSD as a similarity metric.

A threshold value of 1.5Å RMSD is used within sequence groups. Before selecting a representative structure, we divided groups into sub-groups, ensuring that all structures in a sub-group have a pairwise RMSD value less than 1.5Å. Groups are numbered starting from 0 as they are evaluated in structural comparison.

If a group is divided into sub-groups, then the sub-groups are named with small letters, e.g., 4557_a, 4557_b, etc.

Agglomerative clustering is used to form sub-groups for groups that have pairwise RMSD values of more than 1.5Å. Agglomerative clustering is an unsupervised machine learning algorithm that uses a bottom-up clustering approach. It starts from individual clusters (leaf) for each item, then clusters are merged according to the distance with each other. Clusters with the shortest distance are merged into a node. Merging nodes is continued until all data points are assigned to a single cluster named root or until a distance threshold is reached for every node.

Agglomerative clustering is readily implemented in the scikit-learn [49] library. We use a distance matrix formed by pairwise RMSD values as input. Affinity is selected as "precomputed" since we have precomputed RMSD values. Affinity defines the metric used to compute the linkage between two nodes. The distance threshold is set at 1.5 with a complete linkage. Complete linkage minimizes the maximum distance between all observations when merging two nodes. Distance threshold makes sure that no two data points in a node have an RMSD value greater than 1.5.

Supplementary Table 3 shows an example list of comparison results, and Supplementary Code 3A gives the code used for sub-grouping of main groups.

After the groups are divided into sub-groups as necessary, we select the representatives for groups and sub-groups where it applies. We sum all RMSD values a given structure has with all other structures in the group to choose the representatives. The structure with the minimum sum in a given group is selected as the representative. Supplementary Code 3B shows an example of representative selection.

Final groupings and information about whether a structure is representative or not is stored in a Pandas data frame and saved as a CSV file. Table 3.9 shows an example group (Group ID: 4557) with sub-groups. IDs assigned to first groups defined only by sequential similarity is given as main_id; when a sequence group is further divided into sub-groups, the id of sub-group is kept in sub_id column; and lastly, if there is sub_id for a given dimer, it is written to the group_id column shown in the example in Table 3.9, otherwise only the main_id is written as group_id. After a representative is selected for a group, it is marked as True in the

corresponding column; if the structure is not a representative, it is left blank (NaN). If a structure is not similar to the rest of the group, its group_id is replaced with a value of NaN (blank) to show that it is not in any groups.

Table 3.9 An example group of identical interface sequences after structure comparison

	dimer_id	main_id	sub_id	group_id	representative
0	6Q5P_D_E	4557	a	4557_a	NaN
1	6Q5P_A_D	4557	c	4557_c	True
2	6Q5P_E_F	4557	a	4557_a	True
3	6Q5P_A_B	4557	a	4557_a	NaN
4	6Q5P_C_E	4557	c	4557_c	NaN
5	6Q5P_C_D	4557	None	None	NaN
6	6Q5P_C_F	4557	b	4557_b	True
7	6Q5P_B_D	4557	b	4557_b	NaN

Chapter 4:
RESULTS AND DISCUSSION

4.1 Analysis of Structures and Interfaces in PDB Through Years

Worldwide PDB (wwPDB) is the primary source of structural data for proteins. As experimental techniques to identify protein structure improve and become more available and accessible, the number of PDB structures is also growing rapidly. All data represented here is extracted from the current interface data set. Figure 4.1 shows the number of structures deposited to PDB every year and the cumulative count of available structures per year. As the graph shows, the number of structures deposited in PDB has been growing rapidly, especially in the last ten years. As of October 12, 2020, there are 169,681 structures in PDB, 166.022 of them including a protein.

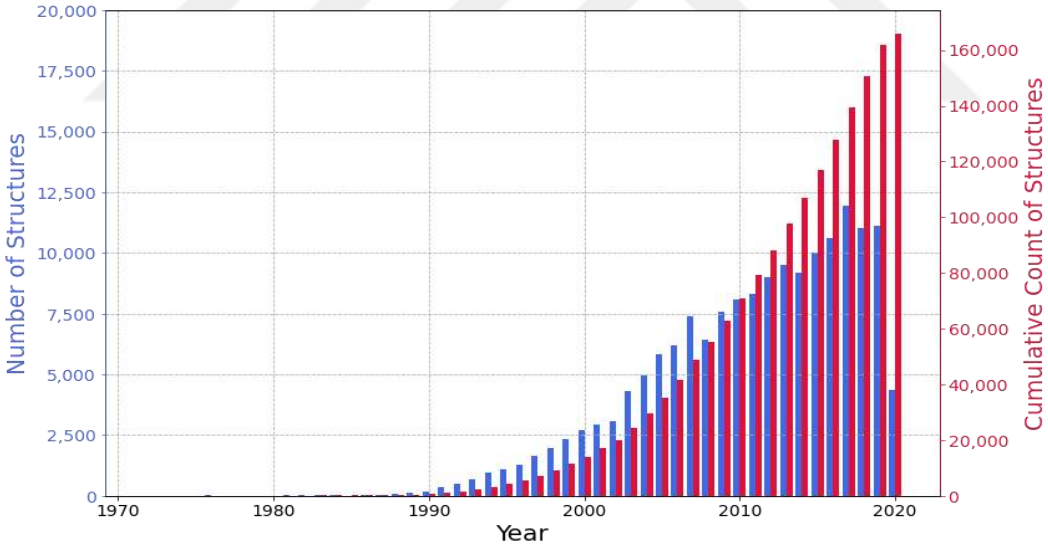


Figure 4.1 Number of structures in PDB through years

Similar and maybe a more rapid trend can also be seen with chain count. Current experimental techniques made analyzing larger structures easier. This progress also affected the chain count in PDB. Figure 4.2 shows the number of chains in PDB over the years. As of October 12, 2020, there are 555,858 chains deposited in PDB. The table that contains this data can be found in Supplementary Table 4 in Appendix A.

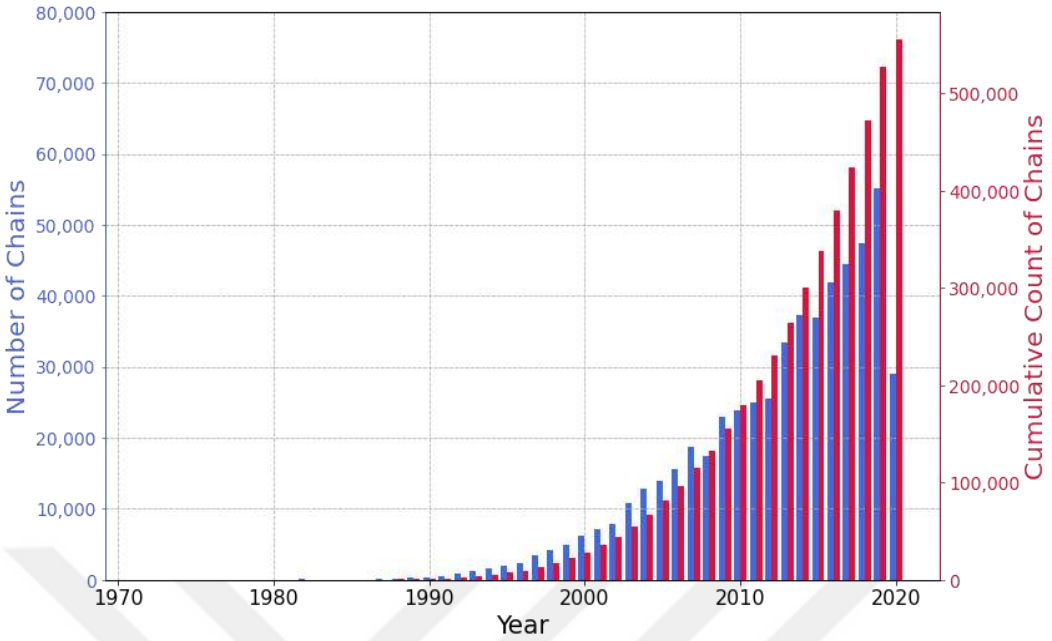


Figure 4.2 Number of chains in PDB through years

The rate of increase in chain numbers is not only a result of growth in number of structures but also the result of the increased number of chains per structure, therefore larger structure sizes. Figure 4.3 presents the average number of chain number and the average number of residues per protein structure through the years. As the graph also reveals, in 2010, there were around three chains and 400 residues per structure; in 2020, this number is more than chain number is six and residue number is about 1200; that is a two-fold increase in chain numbers and nearly three-fold increase in residue count per structure. This increase is an invaluable resource for us in terms of understanding protein interactions better. More chains in a single structure can represent more interfaces than a small structure, and in turn, this helps us provide a more inclusive data set with a greater number of interfaces. Having a more comprehensive range of structural protein-protein interface information available will hopefully make it possible for us to understand different ways proteins can interact and function.

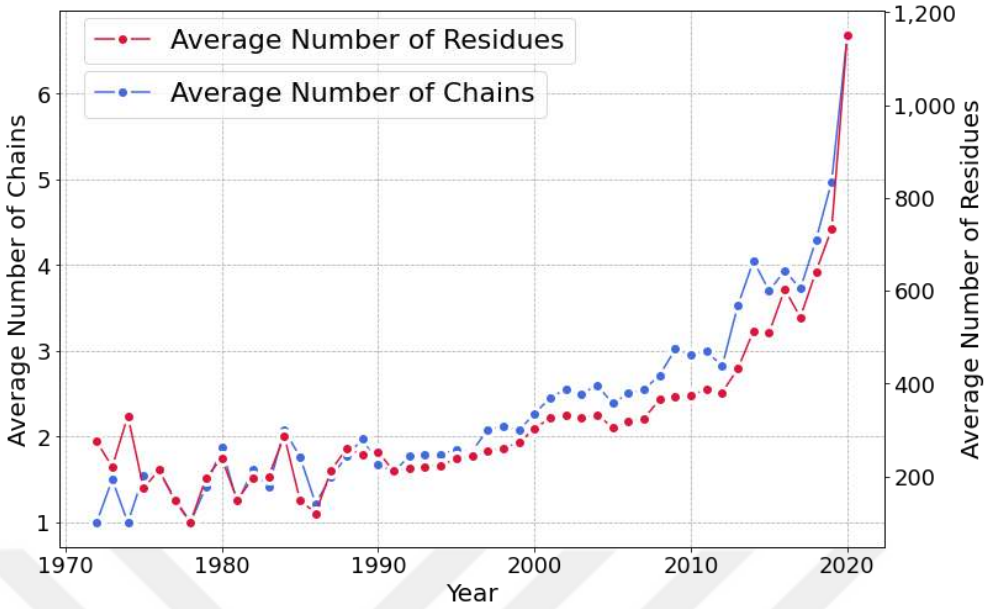


Figure 4.3 Average number of chains and residues per structure through years

The rate of increase in the average number of chains, therefore number of interfaces and average number of residues in structures, can also be explained by the use of different experimental methods. Figure 4.4 shows the count of structures determined with different structural methods each year. Although it can be seen that X-ray diffraction is still the dominant method for determining the structure of proteins, electron microscopy is becoming increasingly used. Figure 4.5 and Figure 4.6 shows the number of structures determined with electron microscopy and the frequency of structures determined with electron microscopy compared with other methods.

Structures determined with electron microscopy has a higher average residue count since electron microscopy enables the investigation of larger structures with higher resolution. The average residue count for structures determined with X-ray diffraction is 389.7; for electron microscopy, this number is 3034.5. That is an almost eight-fold increase in average residue count. This, in turn, results in a higher chain or atom count per structure as well.

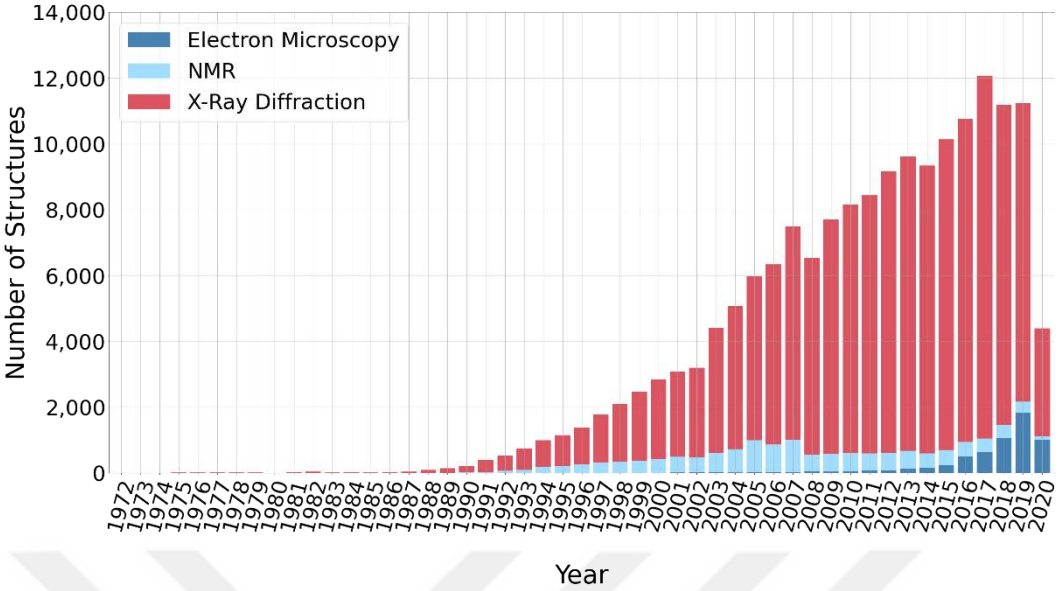


Figure 4.4 Number of structures determined with different experimental methods

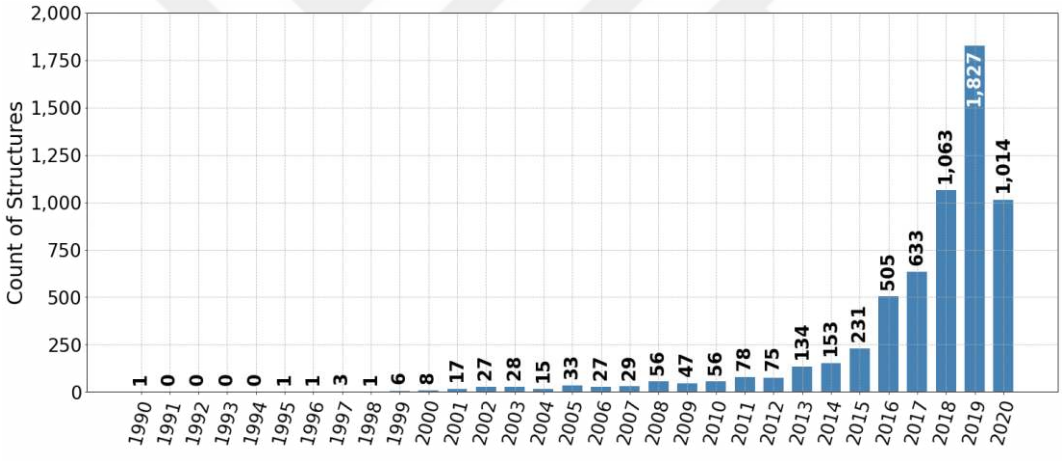


Figure 4.5 Count of structures determined with electron microscopy through years

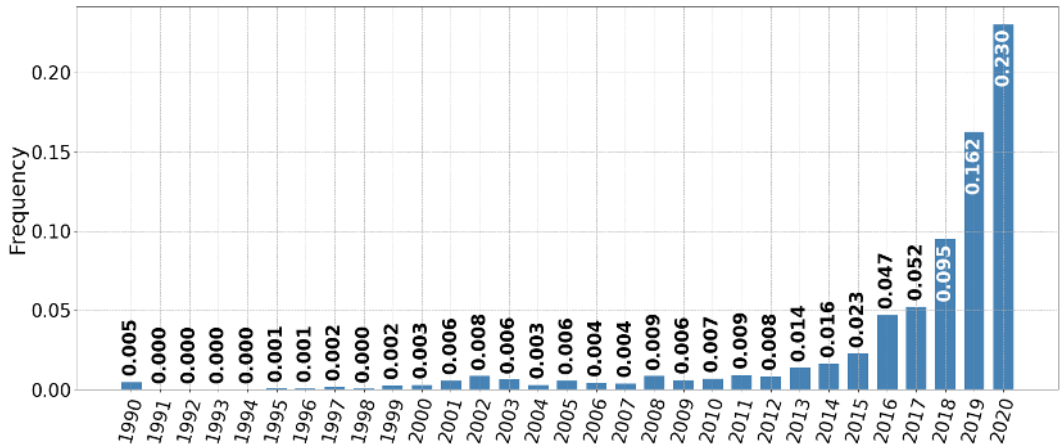


Figure 4.6 Frequency of structures determined with electron microscopy

In the context of this study, we have two different challenges regarding the size of proteins. The first of these challenges come from proteins with too many chains, which we solved using distance filtering as explained in section 3.1.2, and the second challenge comes from chains with too many atoms, which we solved by keeping distance matrices in HDF5 files as described in section 3.1.3. The rationalization for using these methods is only possible by defining what is too big; for this study, we need to specify that computationally.

In terms of chain count, we defined large structures as protein structures with more than 30 chains. There are 1,818 structures with more than 30 chains deposited in PDB. If we apply a SASA comparison for 31 chains, we need to make 465 comparisons to evaluate each binary interaction without filtering. An example structure with 31 chains is PDB ID: 3J7A [70]; in this case, the comparison number reduces to 286 after filtering. As the chain number increases, the difference becomes more significant, as in PDB ID:3J3Y [58]. 3J3Y has 1,176 chains and requires 690,900 comparisons without any filtering, but after distance filtering between chains, the comparison count reduces to 30,604. Less comparison without losing accuracy results in faster identification of dimers.

Definition of large chains in terms of atom count depends on the availability of RAM on the current machine in use. If we evaluate our data set for a standard 16GB of RAM, then our threshold for atom count in a single chain becomes 26,754 atoms, assuming equally sized chains for comparison. There are 794 chain structures in PDB with more than 26,754 atoms. This means that if we did not implement a method to overcome this challenge, we would not be able to extract interfaces from 278 dimer interactions. If we consider the same for an 8 GB RAM, then the atom count threshold in a single chain becomes 18,198 atoms. There are 1,433 chains in PDB with more than 18,918 atoms in a single chain. As a result, we would not be able to extract interfaces from 463 dimer interactions.

The methods implemented to overcome challenges that come from the size of proteins or chains are becoming increasingly important because the size of structures in PDB is becoming larger.

4.2 Interface Data Set

We evaluated 166.022 PDB structures that include at least one protein chain to determine whether they might include an interface or not. There is a total of 98,138 PDB structures that contains a dimer structure. From these structures, we extracted 718,610 dimer structures as a result of SASA calculations.

Evaluation of the dimer structure for interfaces resulted in 94,158 PDB structures with 499,169 interfaces. Table 4.1 gives a summary of current data. Figure 4.7 shows the number of interfaces and the cumulative count of interfaces by year. Interface counts from previous studies are also marked on the graph. Figure 4.8 shows the cumulative count of interfaces on log-scale. Table 4.2 quantifies the increase in interface counts between previous studies.

Table 4.1 Summary of the current data set

Data retrieval date:	October 12, 2020
The number of structures in PDB:	169,681
The number of structures with a protein chain:	166,022
The number of structures with at least two protein chains:	99,241
All possible binary interactions between monomers:	4,639,921
Number of dimers after SASA:	718,610
The number of interfaces:	499,169

The increase in interface numbers through the years can be attributed to the rise in the number of structures deposited each year and the use of different experimental methods for structure determination. The Pearson correlation coefficient for the comparison of the number of structures per year and the number of interfaces per year is 0.889, which suggests a strong positive correlation. Also, the correlation coefficient for the comparison of the number of structures determined with electron microscopy and the number of interfaces per year is 0.866, which also suggests a strong positive correlation.

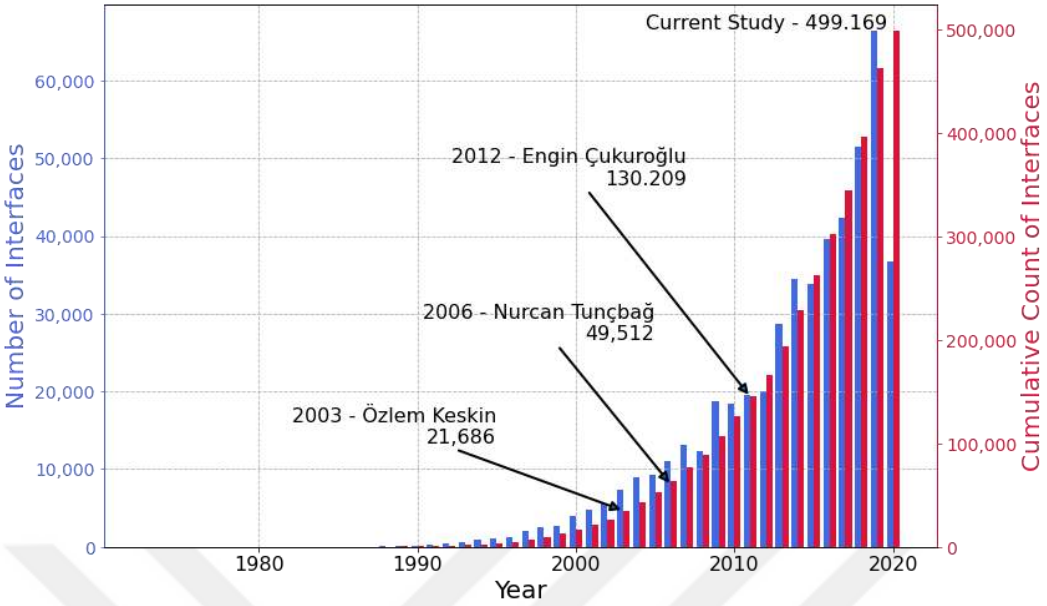


Figure 4.7 Count and cumulative count of interfaces through the years

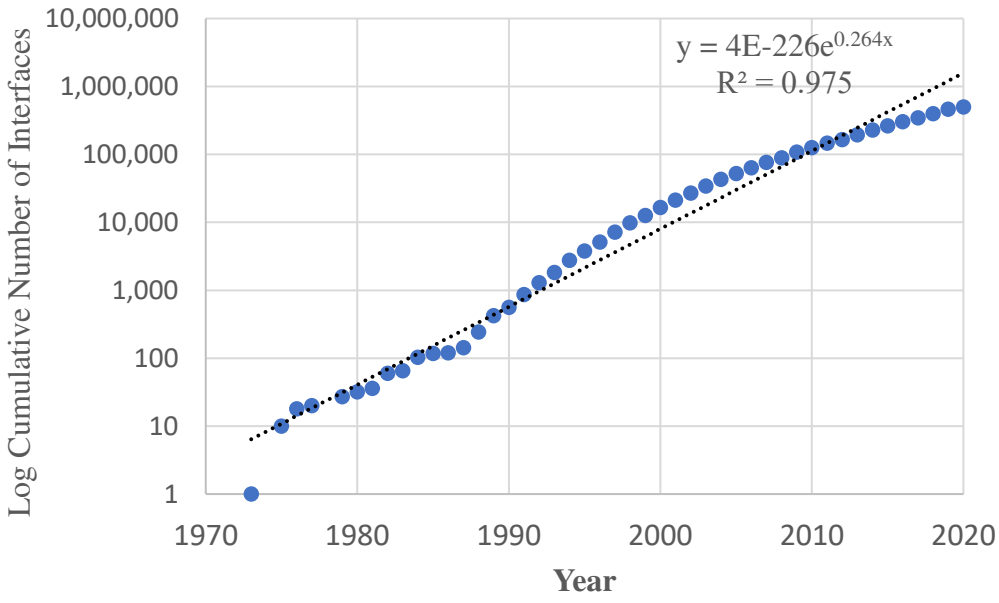


Figure 4.8 Cumulative count of interfaces through the years on log-scale

Table 4.2 Comparison of current data set with previous data sets

	Current Data Set	2012 Data Set	2006 Date Set	2003 Data Set
Structures	169,681	78,477	34,817	18,687
Structures with Interfaces	94,158	39,196	15,268	7,243
Number of Interfaces	499,169	130,209	49,512	21,686

Figure 4.8 shows the distribution of the number of residues in interfaces. Residue count is calculated as a sum of both chains of an interface. The larger plot on the outside gives a general outlook; the smaller plot with red shows a subset of data that has more than 800 residues. The average size of interfaces is 107.2 residues with a minimum of 10 and a maximum of 1574.

When we analyzed the individual chains of interfaces, it revealed that the smaller chains in interfaces have an average residue count of 48.2 with a minimum of 5 and a maximum of 771. For larger chains, the average residue count is 58.9, with a minimum of 5 and a maximum of 803.

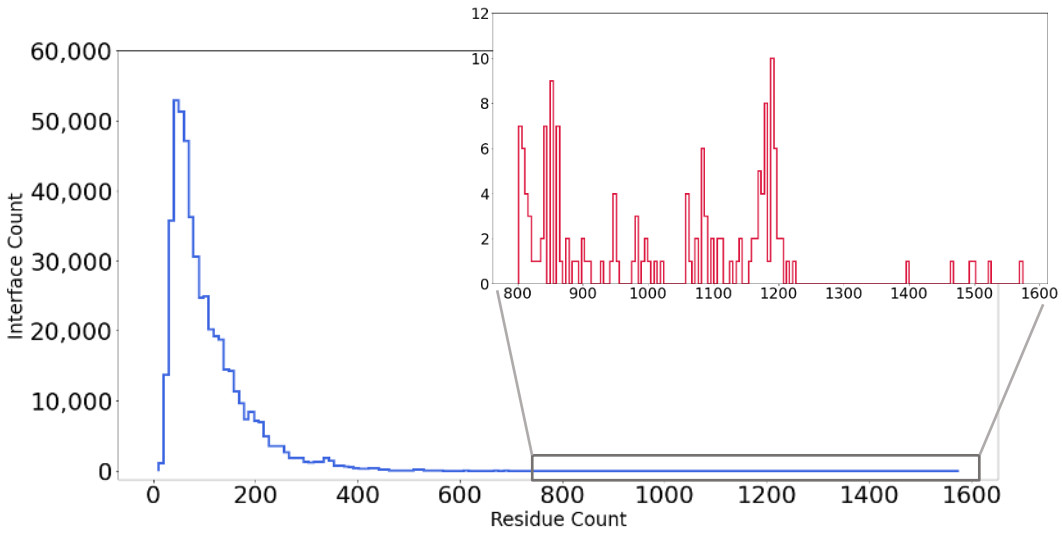


Figure 4.8 Distribution of residue count in interfaces

4.2.1 Analysis of Homodimers and Heterodimers

We investigated the subunit interactions (homodimers and heterodimers) of dimers with interfaces. From 499,169 dimer interactions, which include interfaces, 4,515 pairs have a UNK residue in one of the chains. UNK residues are which the atom coordinates are given in the structure (mmCIF) file, but the residue names are not available. Since we have no information about the residue name, it is not possible to compare these sequences. From the remaining 494,654 dimer structures, homodimers constitute 142,817 dimer structures, and heterodimers comprise 351,837 structures. Figure 4.9 shows the count of homodimers and heterodimers. We also investigated the interfaces of homodimers and heterodimers in terms of sequence identity; Figure 4.10 shows the results of interface sequence identity comparison in homodimers and heterodimers.

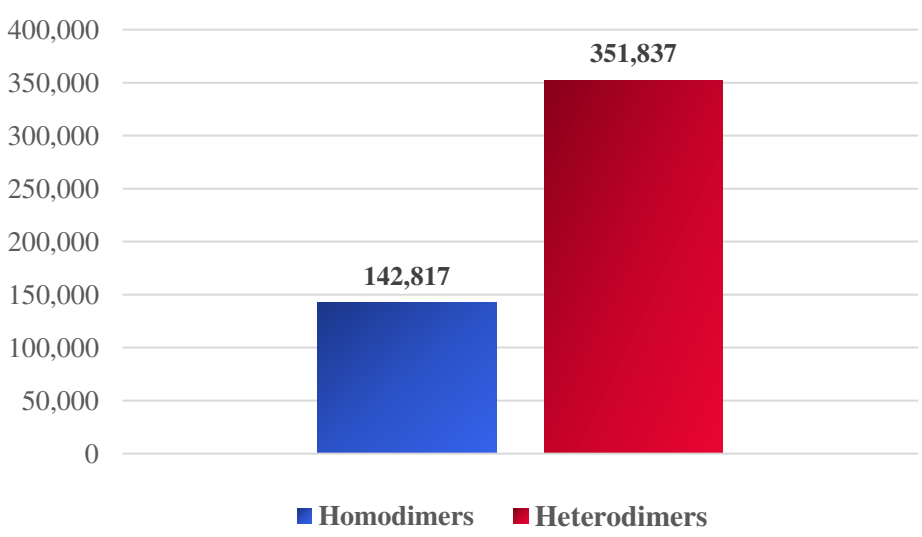


Figure 4.9 Count of homodimers and heterodimers in dimers with interfaces after structures with UNK residues excluded.

Figure 4.10 shows that both homodimers and heterodimers have interfaces with identical chain sequences. Homodimers have 24,598 interfaces with the same sequence in both chains, and this number is 13,658 for heterodimers. A heterodimer with an identical chain sequence in the interface region is given in Figure 4.11, and Table 4.3 shows the sequences for interfaces and chains for PDB ID: 4RAP [71].

An analysis of interfaces that have identical sequences between heterodimers shows that, even though the complete sequence of the chains of the heterodimers are not identical, they are usually similar and almost the same, especially around the interface region. Usually, the differences between sequences occur at both ends of the monomers. Even though the difference between two chains is usually not that much since we classify homodimers as identical sequences only, two chains with any difference are categorized as heterodimers.

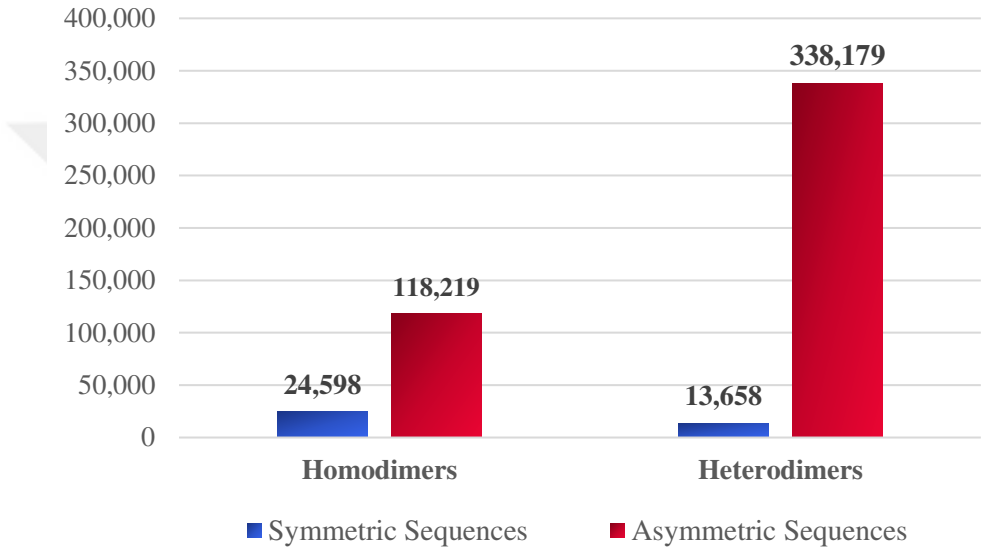


Figure 4.10 Count of sequentially identical and non-identical interface chains in homodimers and heterodimers

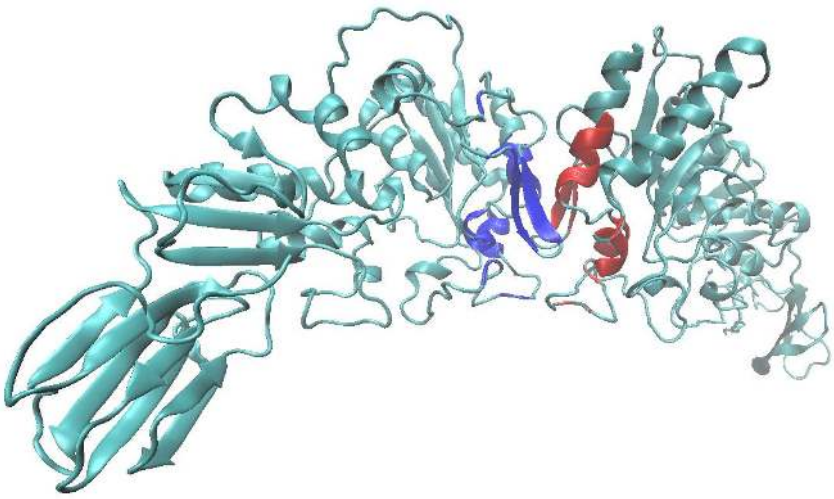


Figure 4.11 PDB ID: 4RAP with chains E and L. Interface region is shown in blue and red.

Table 4.3 PDB ID: 4RAP with chain E and L. Interface and chain sequences. Interface regions are highlighted with grey, and interface residues are highlighted in yellow in the chain sequences.

Interface Sequence Chain E
CYS,GLN,ALA,LYS,TYR,TRP,ASN,ASN,GLY,THR,GLY,TRP,SER,GLU,VAL,ILE, _ ,VAL, CYS, _ ,ASP,ALA,HIS,TYR,GLY,GLN,GLY,PHE,VAL,TRP,ASN,HIS,ILE,PRO,TRP,GLY, ALA,GLU,ASP,PHE,THR, _ ,ASP,PHE,LEU,TRP, _ ,HIS,LYS,ASN,THR,ASP,ARG,GLN, PHE,GLU,CYS,THR,ARG,LEU,ILE
Interface Sequence Chain L
CYS,GLN,ALA,LYS,TYR,TRP,ASN,ASN,GLY,THR,GLY,TRP,SER,GLU,VAL,ILE, _ ,VAL, CYS, _ ,ASP,ALA,HIS,TYR,GLY,GLN,GLY,PHE,VAL,TRP,ASN,HIS,ILE,PRO,TRP,GLY, ALA,GLU,ASP,PHE,THR, _ ,ASP,PHE,LEU,TRP, _ ,HIS,LYS,ASN,THR,ASP,ARG,GLN, PHE,GLU,CYS,THR,ARG,LEU,ILE
Monomer Sequence Chain E
ILE,THR,PRO,PRO,ASP,THR,PRO,THR,GLN,ALA,GLY,PRO,GLU,ASN,ILE,PHE,TYR,ASP, PHE,ASN,ASP,GLY,ALA,ARG,VAL,LEU,LEU,PRO,GLU,GLY,LYS,TRP,HIS,VAL,ARG,LEU, LEU,ASP,ALA,ASP,SER,GLU,ASN,ILE,LEU,PHE,CYS,CYS,ASP,VAL,ASP,LYS,GLY,TRP, VAL,THR,SER,SER,LYS,LYS,TYR,PHE,VAL,ARG,PHE,ARG,ILE,GLN,VAL,PHE,ARG,GLN, GLY,ALA,ALA,THR,PRO,LEU,LEU,ASP,GLU,THR,LEU,LYS,LEU,LYS,ASP,ARG,PRO,VAL, LEU,ILE,SER,PHE,PRO,THR,GLY,THR,LEU,GLY,ASP,LEU,LEU,GLY,TRP,PHE,PRO,TYR, ALA,GLU,ARG,PHE,GLN,SER,LEU,HIS,LYS,CYS,ARG,LEU,GLU,CYS,THR,SER,GLN,ASP, ILE,ILE,ASP,LEU,LEU,ALA,PRO,GLN,TYR,PRO,GLN,ILE,GLN,PHE,SER,THR,PRO,ASP, LYS,PRO,ARG,THR,VAL,ALA,PRO,TYR,ALA,THR,TYR,ARG,VAL,GLY,LEU,TYR,PHE,GLY, GLY,ASP,THR,ASN,ASN,GLN,PRO,VAL,ASP,PHE,ARG,LYS,VAL,GLY,PHE,HIS,ARG,SER, ALA,GLY,TYR,ILE,LEU,GLY,VAL,ASP,PRO,ARG,GLU,ALA,PRO,VAL,ARG,LEU,ASP,LEU, SER,ALA,PRO,ARG,VAL,ILE,ALA,ALA,PRO,TYR,VAL,CYS,ILE,ALA,THR,GLN,SER,THR, CYS,GLN,ALA,LYS,TYR,TRP,ASN,ASN,GLY,THR,GLY,TRP,SER,GLU,VAL,ILE,ALA,HIS, LEU,LYS,SER,LEU,GLY,TYR,ARG,VAL,CYS,ILE,ASP,ARG,ASP,ALA,HIS,TYR,GLY,GLN, GLY,PHE,VAL,TRP,ASN,HIS,ILE,PRO,TRP,GLY,ALA,GLU,ASP,PHE,THR,GLY,LYS,LEU, PRO,LEU,GLN,GLU,ARG,VAL,ASN,LEU,LEU,ARG,HIS,ALA,SER,PHE,PHE,ILE,GLY,LEU, PRO,SER,GLY,LEU,SER,TRP,LEU,ALA,TRP,ALA,THR,ARG,ILE,PRO,VAL,VAL,LEU,ILE, SER,GLY,PHE,SER,LEU,PRO,ASN,SER,GLU,PHE,TYR,THR,PRO,TRP,ARG,VAL,PHE,ASN, SER,HIS,GLY,CYS,TYR,GLY,CYS,TRP,ASP,ASP,THR,SER,LEU,ASN,PHE,ASP,HIS,HIS, ASP,PHE,LEU,TRP,CYS,PRO,ARG,HIS,LYS,ASN,THR,ASP,ARG,GLN,PHE,GLU,CYS,THR, ARG,LEU,ILE,THR,GLY,ALA,GLN,VAL,ASN,GLY,VAL,ILE,ASN,LYS,LEU,HIS,ARG,SER, LEU,THR,GLU,GLN,GLY,VAL,GLU,ALA,THR,LEU
Monomer Sequence Chain L
PRO,PRO,ASP,THR,PRO,THR,GLN,ALA,GLY,PRO,GLU,ASN,ILE,PHE,TYR,ASP,PHE,ASN, ASP,GLY,TRP,HIS,VAL,ARG,LEU,LEU,ASP,ALA,ASP,SER,GLU,ASN,ILE,LEU,PHE,CYS, CYS,GLY,TRP,VAL,THR,SER,SER,LYS,LYS,TYR,PHE,VAL,ARG,PHE,ARG,ILE,GLN,VAL, PHE,ARG,GLN,GLY,ALA,ALA,THR,PRO,LEU,LEU,ASP,GLU,THR,LEU,LYS,LEU,LYS,ASP, ARG,PRO,VAL,LEU,ILE,SER,PHE,PRO,THR,GLY,THR,LEU,GLY,ASP,LEU,LEU,GLY,TRP, PHE,PRO,TYR,ALA,GLU,ARG,PHE,GLN,SER,LEU,HIS,LYS,CYS,ARG,LEU,GLU,CYS,THR, SER,GLN,ASP,ILE,ILE,ASP,LEU,LEU,ALA,PRO,GLN,TYR,PRO,GLN,ILE,GLN,PHE,SER, THR,PRO,ASP,LYS,PRO,ARG,THR,VAL,ALA,PRO,TYR,ALA,THR,TYR,ARG,VAL,GLY,LEU, TYR,PHE,GLY,GLY,ASP,THR,ASN,ASN,GLN,PRO,VAL,ASP,PHE,ARG,LYS,VAL,GLY,PHE, HIS,ARG,SER,ALA,GLY,TYR,ILE,LEU,GLY,VAL,ASP,PRO,ARG,GLU,ALA,PRO,VAL,ARG, LEU,ASP,LEU,SER,ALA,PRO,ARG,VAL,ILE,ALA,ALA,PRO,TYR,VAL,CYS,ILE,ALA,THR, GLN,SER,THR,CYS,GLN,ALA,LYS,TYR,TRP,ASN,ASN,GLY,THR,GLY,TRP,SER,GLU,VAL, ILE,ALA,HIS,LEU,LYS,SER,LEU,GLY,TYR,ARG,VAL,CYS,ILE,ASP,ARG,ASP,ALA,HIS, TYR,GLY,GLN,GLY,PHE,VAL,TRP,ASN,HIS,ILE,PRO,TRP,GLY,ALA,GLU,ASP,PHE,THR, GLY,LYS,LEU,PRO,LEU,GLN,GLU,ARG,VAL,ASN,LEU,LEU,ARG,HIS,ALA,SER,PHE,PHE, ILE,GLY,LEU,PRO,SER,GLY,LEU,SER,TRP,LEU,ALA,TRP,ALA,THR,ARG,ILE,PRO,VAL, VAL,LEU,ILE,SER,GLY,PHE,SER,LEU,PRO,ASN,SER,GLU,PHE,TYR,THR,PRO,TRP,ARG, VAL,PHE,ASN,SER,HIS,GLY,CYS,TYR,GLY,CYS,TRP,ASP,ASP,THR,SER,LEU,ASN,PHE, ASP,HIS,HIS,ASP,PHE,LEU,TRP,CYS,PRO,ARG,HIS,LYS,ASN,THR,ASP,ARG,GLN,PHE, GLU,CYS,THR,ARG,LEU,ILE,THR,GLY,ALA,GLN,VAL,ASN,GLY,VAL,ILE,ASN,LYS,LEU, HIS,ARG,SER,LEU,THR,GLU,GLN,GLY,VAL

4.2.2 Analysis of Secondary Structure Elements

DSSP [72, 73] database is used with the help of the "DsspApp" method from the Biotite package to extract information about secondary structure elements in interfaces. DSSP can differentiate eight different types of secondary structure elements. We reduced it to three categories by assigning multiple identifiers to each group. The list of possible secondary structures available in DSSP and their assignments are given in Table 4.4.

Table 4.4 List of available SSEs in DSSP and their assignments

DSSP Notation	Description	Assigned Notation
C	loop, coil, or irregular	coil
H	α-helix	alpha
B	β-bridge	beta
E	extended strand, participation in β-ladder	beta
G	3_{10}-helix	alpha
I	π-helix	alpha
T	hydrogen-bonded turn	coil
S	bend	coil

We assigned SSEs to every residue in interfaces to analyze the distribution of different combinations. We evaluated them under five categories: α-α, β-β, α-αβ, β-αβ and αβ-αβ. α-α defines interfaces that have only α-helices as SSE in both chains, apart from coils. Likewise, β-β defines interfaces with only β-sheets as SSE in both chains. α-αβ and β-αβ define interfaces that have only α-helices or β-sheets in one chain and both in the other chain, and lastly, αβ-αβ defines interfaces with both SSEs in both chains. Figure 4.14 shows an example structure for each of these possible categories.

Out of 499,169 interfaces, 319,813 of them have SSE assignments in both chains. Interfaces that are not in one of these categories have only coils in one or both chains, which means they have no SSE assignment in one or both chains according to the DSSP database. We show the count of interfaces in different categories in Figure 4.12. The graph shows that more than half (52.5%) of the interfaces are in the α-α category, and around 10% of interfaces are in the β-β category.

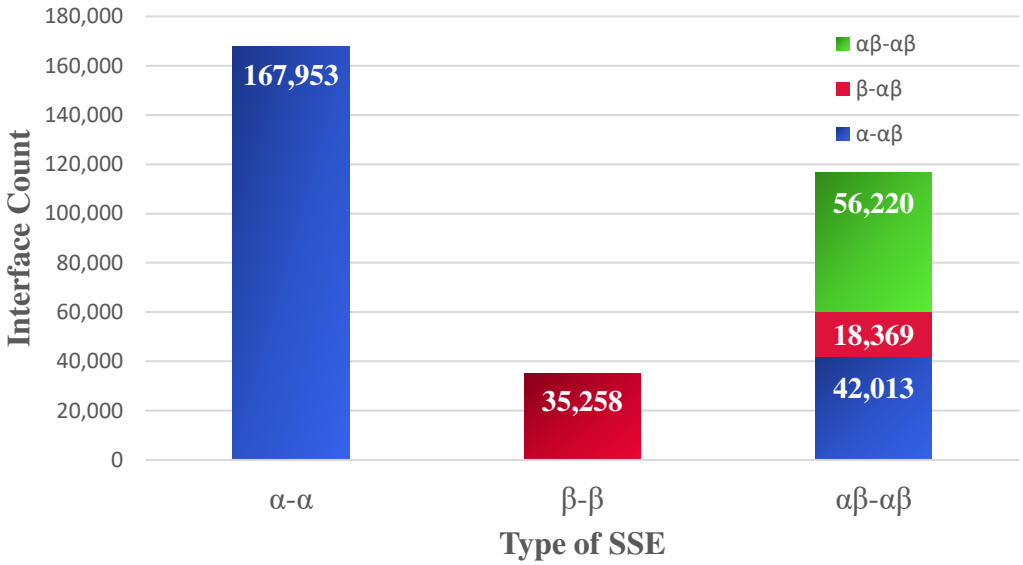


Figure 4.12 Count of interfaces in SSE categories

Figure 4.13 presents the frequency of SSEs in interfaces throughout the years. This is calculated by dividing the number of all available residue assignments to α-helices or β-sheets by the total number of interface residues each year. Even though the frequency fluctuates until 1990 due to a low number of structures deposited in PDB, there were only 635 structures deposited until then; it can be seen that after 1990 frequency of α-helices is generally around four-fold greater than that of β-sheets.

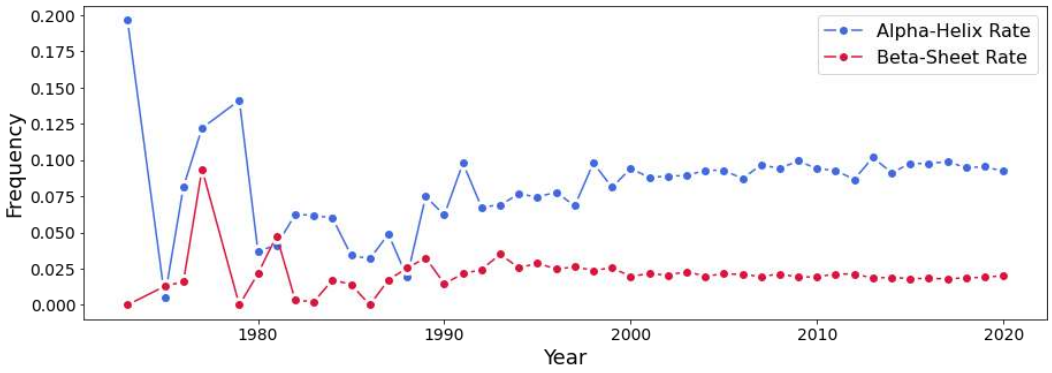


Figure 4.13 Secondary structure content of interfaces through the years

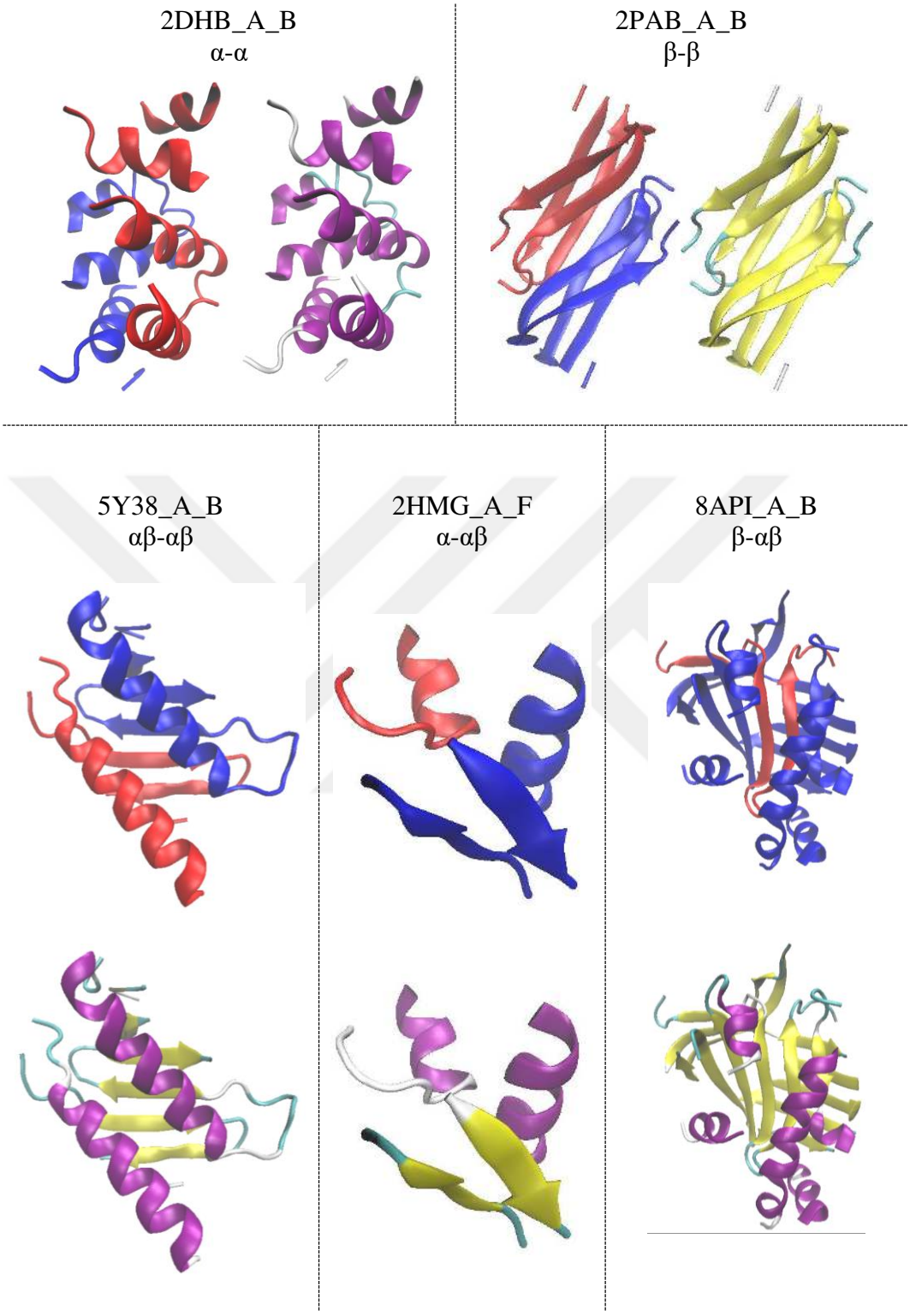


Figure 4.14 Example interface structures with different secondary structural elements in their chains. Red/blue representations are by chain ID, and purple/yellow representations are by secondary structures, α-helix, and β-sheet.

4.2.3 Analysis of Amino Acid Frequencies

We analyzed the frequencies of twenty standard amino acids in the interface region, including both contacting and nearby residues and non-interface regions of proteins. Interface region represents the unique residues on a monomer that participate in an interaction with another chain so that chains that participate in multiple interactions, which have overlapping interface regions, have only a single unique set of residues in frequency calculations.

Dai et al. (2016) [74] conducted a similar analysis on a different, less redundant data set. Even though our exact frequency values do not agree perfectly, our findings on relative frequencies between interface and non-interface regions are similar. Figure 4.15 shows the frequencies of amino acids in interface and non-interface regions of proteins available in PDB without considering redundant structures.

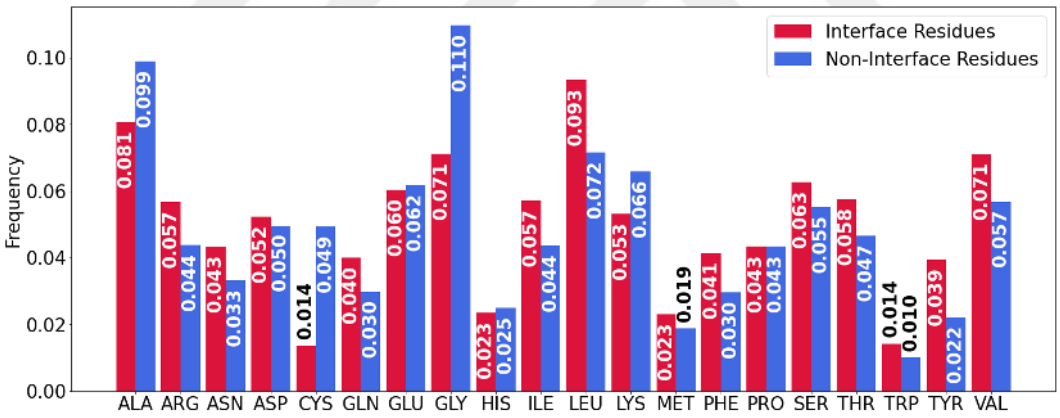


Figure 4.15 Frequency of amino acids in interfaces

Figure 4.16 shows the frequencies of amino acids in contacting residue and nearby residues of interfaces. Contacting residues constitute 44.57% of all residues in interfaces, and 55.43% of all residues in interfaces are nearby residues. Some amino acids appear to be more frequent in contacting residues than nearby residues. ARG, ILE, PHE, SER, THR, TYR, VAL are observed more frequently in contacting residues, and ALA, GLY, and LYS are observed more frequently in nearby residues.

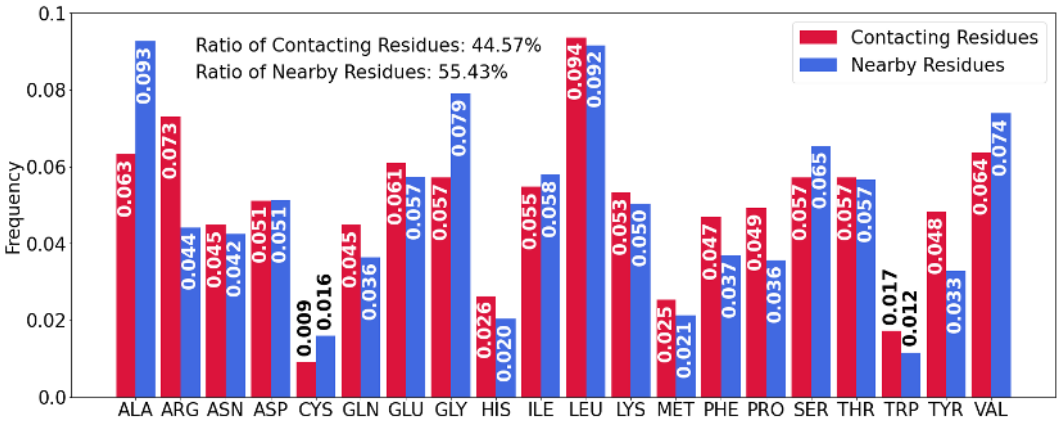


Figure 4.16 Frequency of amino acids by the interface region

4.3 Sequence-Based Interface Clustering

In this study, we grouped interfaces and chains first by identical sequences and clustered them by their structural similarities within the identical sequence groups. One of the critical steps is to decide how to represent interface sequences for sequence comparison. In section 4.3.1, we describe the reasoning behind our selection of using single-spaced interface sequence representation. Section 4.3.2 shows the results of chain-wise interface clustering and structural comparisons within a group. We also present our analysis of the interacting chains of already grouped and clustered chains. In the last section, we analyze the interface sequence groups and their structural comparison.

4.3.1 Comparison Between Different Sequence Segmentation Options

For identifying identical interface sequences, we first need to decide on an interface sequence representation. In our data set, extracted interfaces are not continuous; therefore, it is crucial to represent this fragmented structure in sequences. Detailed explanations of these representations are previously provided in section 3.2.1 in Materials and Methods.

We calculated the average pairwise RMSDs for the most populated twenty groups for all options to compare three different segmentation options. The first twenty groups have no differences between non-spaced and single-spaced segmentation. One-by-one comparison between the single-spaced and multi-spaced

groups shows that single-spaced groups can cover more interfaces without a significant increase in RMSD averages. The average RMSD for single-spaced sequences in the first twenty groups is 0.226 with 10,330 sequences, and it is 0.223 with 10,261 sequences. Considering these findings and the underlying information that non-interface regions between interfaces in a single chain may be considerably different than each other even if the interface sequences are the same, we decided to use either non-spaced or single-spaced segmentation.

Table 4.5 Average RMSDs for most populated twenty sequence groups for different segmentation options

	Non-Spaced		**Single-Spaced**		**Multi-Spaced**	
Group #	Group Size	Mean ± SD RMSD	Group Size	Mean ± SD RMSD	Group Size	Mean ± SD RMSD
1	887	0.50 ± 0.11	887	0.50 ± 0.11	887	0.50 ± 0.11
2	680	0.26 ± 0.06	680	0.26 ± 0.06	680	0.26 ± 0.06
3	603	0.96 ± 0.53	603	0.96 ± 0.53	603	0.96 ± 0.53
4	558	0.12 ± 0.09	558	0.12 ± 0.09	558	0.12 ± 0.09
5	532	0.26 ± 0.18	532	0.26 ± 0.18	525	0.12 ± 0.09
6	525	0.12 ± 0.09	525	0.12 ± 0.09	519	0.11 ± 0.12
7	519	0.11 ± 0.12	519	0.11 ± 0.12	505	0.14 ± 0.13
8	505	0.14 ± 0.13	505	0.14 ± 0.13	495	0.48 ± 0.22
9	495	0.48 ± 0.22	495	0.48 ± 0.22	495	0.12 ± 0.16
10	495	0.12 ± 0.16	495	0.12 ± 0.16	492	0.22 ± 0.16
11	491	0.14 ± 0.06	491	0.14 ± 0.06	491	0.14 ± 0.06
12	482	0.36 ± 0.09	482	0.36 ± 0.09	482	0.36 ± 0.09
13	478	0.07 ± 0.07	478	0.07 ± 0.07	478	0.07 ± 0.07
14	465	0.12 ± 0.07	465	0.12 ± 0.07	465	0.12 ± 0.07
15	461	0.16 ± 0.05	461	0.16 ± 0.05	436	0.14 ± 0.15
16	436	0.14 ± 0.15	436	0.14 ± 0.15	432	0.15 ± 0.06
17	432	0.07 ± 0.04	432	0.07 ± 0.04	432	0.07 ± 0.04
18	432	0.14 ± 0.05	432	0.14 ± 0.05	432	0.14 ± 0.05
19	429	0.17 ± 0.20	429	0.17 ± 0.20	429	0.17 ± 0.20
20	425	0.08 ± 0.07	425	0.08 ± 0.07	425	0.08 ± 0.07

To decide between non-spaced and single-spaced segmentations, we needed to evaluate a larger part of the data set, since the first twenty groups are identical.

Average RMSD values for single-spaced segmentation for the complete interface data set is 0.248, and it is 0.330 for non-spaced segmentation. Single-spaced segmentation resulted in 118,572 groups with 610,005 sequences, while non-spaced segmentation has 118,456 groups with 611,777 sequences. Even though the group number is increased with single-spaced segmentation, the structural comparison count is 11,577,582, which is less than 11,682,799. This suggests, together with the average RMSD results, single-spaced segmentation provides better segregation between similar but not identical interface sequences. These findings encouraged us to pursue the rest of the study using single-spaced sequences for interface chains and interfaces.

4.3.2 Analysis of Interface Chain Groups with Identical Sequences

In our interface data set, we have 998,338 interface chains. First, we grouped these interface chains by their sequence similarity. As a result of sequence grouping, we observed that 610,005 are identical with at least one or more chains; therefore, they can be included in sequence groups. The remaining 388,332 chains have unique sequences; consequently, they are not used for sequence groupings.

610,005 sequences are grouped into 118,572 identical sequence groups. The average member count per group is 5.14, where the most populated group has 887 member chains, and 60,729 groups have only two member chains.

Following sequence clustering, we performed a structural comparison within sequence groups. We performed 11,577,582 pairwise RMSD calculations for comparison of interface chain structures within sequence groups. We set an RMSD threshold of 1.5 Å to differentiate between chains with the same sequences but differing structures. If a sequence group has any pairwise RMSD value over the threshold, they are further divided using agglomerative clustering with a cut-off value at 1.5 Å. If the RMSD values for pairwise comparisons are all smaller than the threshold, then the members are kept as they are, and no more clustering is done. If a two-member sequence group has an RMSD greater than the threshold, the group is discarded, and members are treated as unique chains. In Figure 4.17, an example similarity matrix for a group with RMSD values greater than the threshold is given.

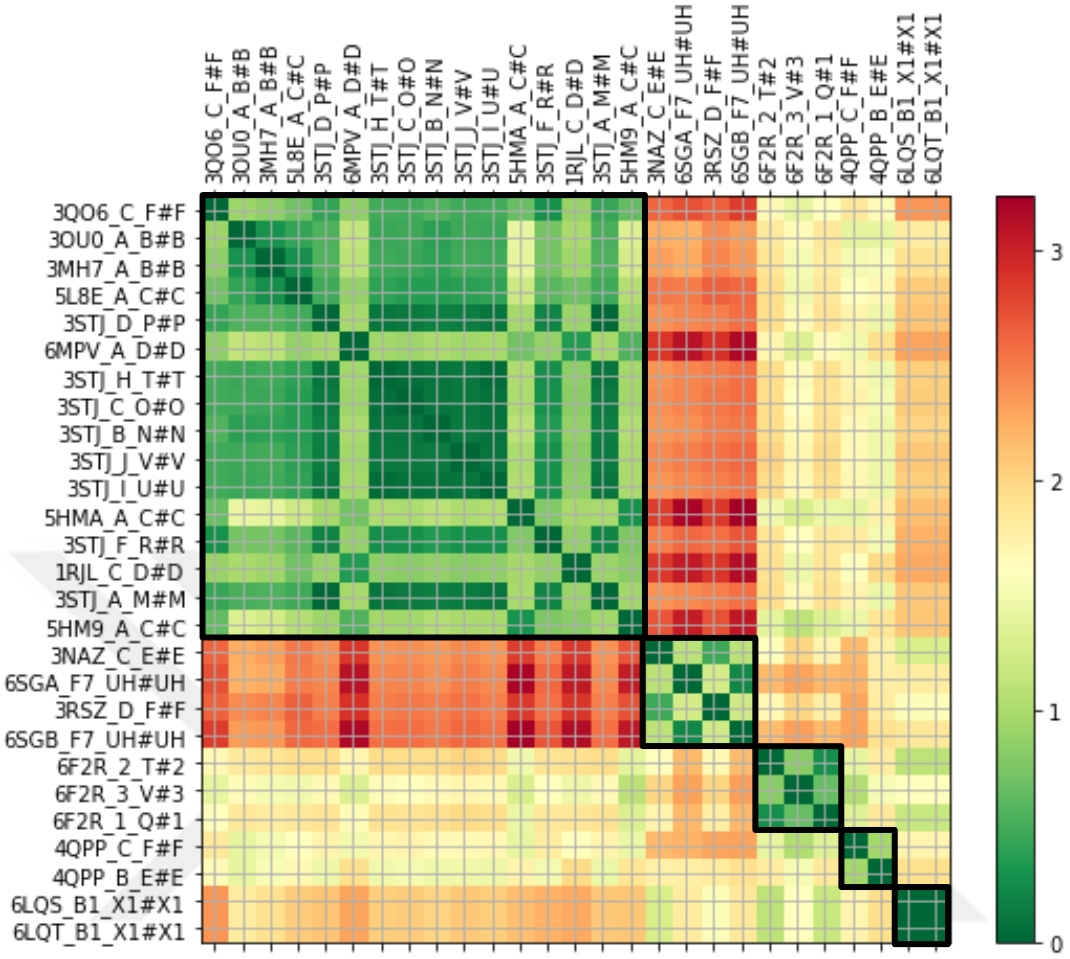


Figure 4.17 Similarity matrix with RMSD values from pairwise comparisons of chain group with main ID 2604 sorted according to final clustering. Individual clusters formed after agglomerative clustering are highlighted with a black square around them.

There are 2,418 sequence groups with a pairwise RMSD value greater than 1.5Å. 779 of those groups have only two members; therefore, these groups are discarded. 1,639 groups have more than two members; RMSD based agglomerative clustering is applied to these groups. 104 groups out of 1,639 have no members with RMSD smaller than the threshold; therefore, these groups are discarded as well. From the remaining 1,535 groups, we have 2,619 groups formed by clustering based on RMSD. As a result, we have a total of 118,773 structurally clustered sequence groups that covers 606,527 chains. Table 4.6 shows a summary of this data.

Table 4.6 Summary of chain groups in the current data set

The number of chains:	998,338
The number of chains in sequence groups:	610,005
The number of sequence groups:	118,572
The number of groups that needs RMSD clustering:	2,418 → 2,619
Number of groups after RMSD clustering:	118,773
The number of chains in groups after clustering:	606,527
Sequentially and structurally unique chains:	510,584

Sequentially and structurally unique chains include the following:

- Chains with no identical sequences in the complete data set. (388,333 chains)
- Representatives of sequence groups and subgroups formed after RMSD clustering. (118,773 chains)
- Chains that are excluded from the groups after RMSD clustering. (3,478 chains)

As a result of sequence and structure comparisons, 998,338 chains in our data set were reduced to 510,584 sequentially and structurally unique interface chains.

4.3.3 Analysis of Interface Groups with Identical Sequences

In the current interface data set, we have 499,169 interfaces. In this part of the study, we first grouped interfaces with identical sequences. We found that, out of 499,169 interfaces, 216,801 of them have at least one other interface with an identical sequence in the data set. Grouping interfaces with identical sequences resulted in 48,533 identical sequence groups. Out of 48,534 groups, 26,852 of them has only two member interfaces. The average number of members in a sequence group is 4.46, and the most populated group has 487 interfaces.

Similar to the process with interface chains, we also calculated pairwise RMSDs within each identical sequence group. Out of 48,533 groups, 238 groups has a pairwise RMSD value greater than the threshold of 1.5Å. We applied RMSD based agglomerative clustering to these groups. As a result of the clustering, we obtained 250 structurally similar sequence groups, which makes a total of 48,546 groups.

Table 4.7 Summary of interface groups current data set

The number of interfaces:	499,169
The number of chains in sequence groups:	216,801
The number of sequence groups:	48,534
The number of groups that needs RMSD clustering:	238 → 250
Number of groups after RMSD clustering:	48,546
Number of interfaces in groups after clustering:	216,484
Sequentially and structurally unique interfaces:	331,231

Similar to the clustered data set with interface chains, unique interfaces include:

- Interfaces with no identical sequences in the complete data set. (282,368 interfaces)
- Representatives of sequence groups and subgroups formed after RMSD clustering. (48,546 interfaces)
- Interfaces that are excluded from the groups after RMSD clustering. (317 interfaces)

As a result of sequential and structural comparisons, 499,169 interfaces in our data set were reduced to 331,231 sequentially and structurally unique interface chains.

One of the underlying goals of this study is to evaluate the usability of interface sequence comparison, together with in-group structural comparisons, as a filtering mechanism for structural clustering of all interface structures in PDB. Due to the increased number of interface structures in PDB, a full data set clustering with structural information is not feasible without proper filtering. 499,169 interfaces need 124,584,595,696 comparisons to evaluate the complete interface data set. This number is also increasing with the growth of PDB. Figure 4.18 shows the plot of change in comparison number with the number of interfaces. If the current increase rate for the number of interfaces in PDB continues, there will be around 720,000 interfaces in PDB by the year 2024. Comparing all interfaces that will be available would take 259,199,640,000 pairwise comparisons, which would be virtually impossible to compute in a reasonable time frame.

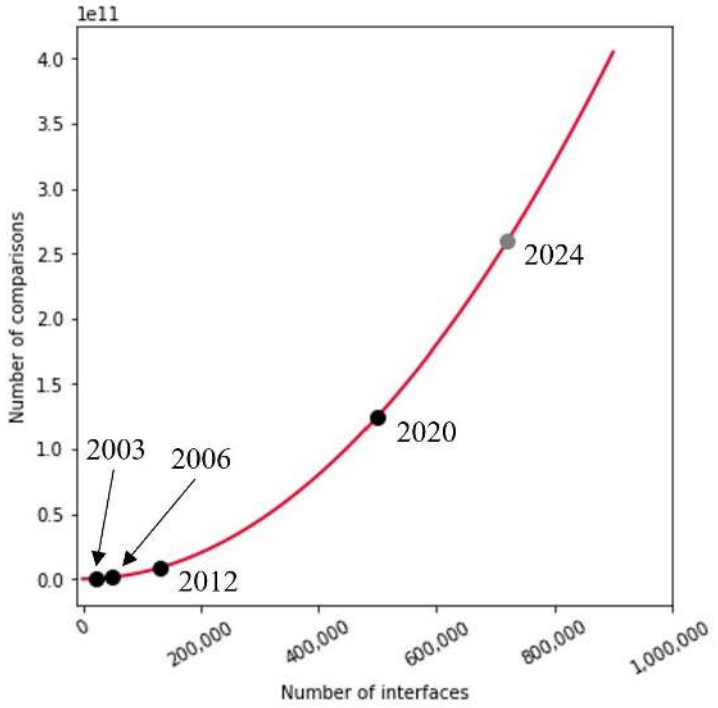


Figure 4.18 Change of the number of comparisons needed for pairwise comparison of a given number of interfaces.

In our current capabilities, we can process approximately 4,000,000 comparisons in a day with eight cores. Even under the assumably ideal circumstances in which we can utilize 40 cores uninterruptedly, this computation would take 6,229 days (~17 years). Assuming only identical sequence filtering, discussed throughout this thesis, is added comparison number of interfaces to compare reduces to 331,231, which needs 54,856,822,065 comparisons for evaluation of complete data set; this is a 2.3-fold reduction from the initial comparison operations required. We also have additional filtering criteria for comparing the complete interface data set, which is also used in a previous study by Çukuroğlu et al. (2014), which depends on the size differences between two interfaces. After both identical sequence filtering and size difference filtering are applied, the number of comparisons reduces to 8,532,297,052. This is a more manageable comparison count than the one we started with; it is almost a 15-fold reduction. This number of comparisons is expected to take around 427 days with 40 cores uninterruptedly. The addition of these filters is a significant improvement that can be used without any loss of accuracy, and they are vital to the study's success.

Chapter 5:
CONCLUSION

Proteins have crucial roles in and between cells. They function by interacting with their environments, molecules, and other proteins. Protein-protein interfaces are structural regions on the protein surface, where two proteins interact. Understanding the structural properties of protein interfaces is an essential step for understanding protein function and for using this information in life-changing fields like drug design and repositioning.

Structural information about proteins, and therefore protein interfaces are growing at an increasing rate. The introduction of more powerful experimental techniques enabled the determination of more complex structures. Also, with these techniques becoming more available, the data size on protein structures also increased. Even though gathering the data is an important step, the only way to properly utilize this data is by finding computational ways to benefit from this data. The larger structures brought computational challenges when it comes to identifying interfaces. Big data size, which comes from the number of structures in PDB, is a challenge on its own that needs to be dealt with. A complete understanding of the data available is only possible after overcoming these challenges.

In this study, we provide tools to identify interfaces from available protein structures. Different methods are implemented to include large structures as well. First of all, our choice of using mmCIF files instead of PDB files enabled us to reach the large structural files, and therefore the information contained in them. We made the analysis of large structure files possible by including methods to handle large matrices and filtering parts of the structures that do not need to be considered. These methods not only increased our algorithms' performance but also enabled us to analyze structures, which would not be possible to evaluate without implementing them.

A unique set of non-redundant interface structures is an important tool for a wide range of studies. The construction of such a data set is only possible by comparing every interface structure available. Big data size is a critical difficulty in this case. As the data size grows, comparison numbers also increase exponentially.

Efficient methods are needed to filter or select structures that are too similar to be considered different. This study proposed a sequence identity and structure similarity-based method to reduce the number of interface comparisons needed without losing information about the diverse range of interface structures. We calculated that the implementation of this filtering resulted in a more than half reduction in comparison numbers. Methods like these are necessary to make our future studies possible.

Our future work plans include creating a structurally non-redundant set of protein interfaces with the current data available. It will be an update of PIFACE data from 2014. The project is to have a web server with regular updates to data; so that the structural information of the data set is always up to date with PDB structures. Previous data sets are used extensively in protein interaction prediction studies. An updated data set can be an essential resource in further studies.

BIBLIOGRAPHY

[1] S. Jones and J. M. Thornton, "Principles of protein-protein interactions," *Proc Natl Acad Sci U S A,* vol. 93, no. 1, pp. 13-20, Jan 9 1996, doi: 10.1073/pnas.93.1.13.

[2] P. Aloy and R. B. Russell, "Ten thousand interactions for the molecular biologist," *Nat Biotechnol,* vol. 22, no. 10, pp. 1317-21, Oct 2004, doi: 10.1038/nbt1018.

[3] R. E. Hibbs *et al.*, "Structural determinants for interaction of partial agonists with acetylcholine binding protein and neuronal alpha7 nicotinic acetylcholine receptor," *EMBO J,* vol. 28, no. 19, pp. 3040-51, Oct 7 2009, doi: 10.1038/emboj.2009.227.

[4] D. E. Scott, A. R. Bayly, C. Abell, and J. Skidmore, "Small molecules, big targets: drug discovery faces the protein-protein interaction challenge," *Nat Rev Drug Discov,* vol. 15, no. 8, pp. 533-50, Aug 2016, doi: 10.1038/nrd.2016.29.

[5] E. S. Ozdemir, F. Halakou, R. Nussinov, A. Gursoy, and O. Keskin, "Methods for Discovering and Targeting Druggable Protein-Protein Interfaces and Their Application to Repurposing," *Methods Mol Biol,* vol. 1903, pp. 1-21, 2019, doi: 10.1007/978-1-4939-8955-3_1.

[6] C. J. Tsai, D. Xu, and R. Nussinov, "Protein folding via binding and vice versa," *Fold Des,* vol. 3, no. 4, pp. R71-80, 1998, doi: 10.1016/S1359-0278(98)00032-7.

[7] S. Tonddast-Navaei and J. Skolnick, "Are protein-protein interfaces special regions on a protein's surface?," *J Chem Phys,* vol. 143, no. 24, p. 243149, Dec 28 2015, doi: 10.1063/1.4937428.

[8] N. Horton and M. Lewis, "Calculation of the free energy of association for protein complexes," *Protein Sci,* vol. 1, no. 1, pp. 169-81, Jan 1992, doi: 10.1002/pro.5560010117.

[9] L. Lo Conte, C. Chothia, and J. Janin, "The atomic structure of protein-protein recognition sites," *J Mol Biol,* vol. 285, no. 5, pp. 2177-98, Feb 5 1999, doi: 10.1006/jmbi.1998.2439.

[10] Y. Mou, P. S. Huang, F. C. Hsu, S. J. Huang, and S. L. Mayo, "Computational design and experimental verification of a symmetric protein homodimer," *Proc Natl Acad Sci U S A,* vol. 112, no. 34, pp. 10714-9, Aug 25 2015, doi: 10.1073/pnas.1505072112.

[11] R. P. Bahadur, P. Chakrabarti, F. Rodier, and J. Janin, "Dissecting subunit interfaces in homodimeric proteins," *Proteins,* vol. 53, no. 3, pp. 708-19, Nov 15 2003, doi: 10.1002/prot.10461.

[12] Y. Ofran and B. Rost, "Analysing six types of protein-protein interfaces," *J Mol Biol,* vol. 325, no. 2, pp. 377-87, Jan 10 2003, doi: 10.1016/s0022-2836(02)01223-8.

[13] D. Baker and A. Sali, "Protein structure prediction and structural genomics," *Science,* vol. 294, no. 5540, pp. 93-6, Oct 5 2001, doi: 10.1126/science.1065659.

[14] B. Rost, "Twilight zone of protein sequence alignments," *Protein Eng,* vol. 12, no. 2, pp. 85-94, Feb 1999, doi: 10.1093/protein/12.2.85.

[15] A. R. Kinjo and K. Nishikawa, "Eigenvalue analysis of amino acid substitution matrices reveals a sharp transition of the mode of sequence

conservation in proteins," *Bioinformatics,* vol. 20, no. 16, pp. 2504-8, Nov 1 2004, doi: 10.1093/bioinformatics/bth297.

[16] S. Jones and J. M. Thornton, "Analysis of protein-protein interaction sites using surface patches," *J Mol Biol,* vol. 272, no. 1, pp. 121-32, Sep 12 1997, doi: 10.1006/jmbi.1997.1234.

[17] M. Guharoy and P. Chakrabarti, "Secondary structure based analysis and classification of biological interfaces: identification of binding motifs in protein-protein interactions," *Bioinformatics,* vol. 23, no. 15, pp. 1909-18, Aug 1 2007, doi: 10.1093/bioinformatics/btm274.

[18] L. C. Xue, D. Dobbs, and V. Honavar, "HomPPI: a class of sequence homology based protein-protein interface prediction methods," *BMC Bioinformatics,* vol. 12, p. 244, Jun 17 2011, doi: 10.1186/1471-2105-12-244.

[19] F. Minhas, B. J. Geiss, and A. Ben-Hur, "PAIRpred: partner-specific prediction of interacting residues from sequence and structure," *Proteins,* vol. 82, no. 7, pp. 1142-55, Jul 2014, doi: 10.1002/prot.24479.

[20] C. Yan, F. Wu, R. L. Jernigan, D. Dobbs, and V. Honavar, "Characterization of protein-protein interfaces," *Protein J,* vol. 27, no. 1, pp. 59-70, Jan 2008, doi: 10.1007/s10930-007-9108-x.

[21] L. C. Xue, D. Dobbs, A. M. Bonvin, and V. Honavar, "Computational prediction of protein interfaces: A review of data driven methods," *FEBS Lett,* vol. 589, no. 23, pp. 3516-26, Nov 30 2015, doi: 10.1016/j.febslet.2015.10.003.

[22] E. Cukuroglu, A. Gursoy, R. Nussinov, and O. Keskin, "Non-redundant unique interface structures as templates for modeling protein interactions," *PLoS One,* vol. 9, no. 1, p. e86738, 2014, doi: 10.1371/journal.pone.0086738.

[23] O. Keskin, C. J. Tsai, H. Wolfson, and R. Nussinov, "A new, structurally nonredundant, diverse data set of protein-protein interfaces and its implications," *Protein Sci,* vol. 13, no. 4, pp. 1043-55, Apr 2004, doi: 10.1110/ps.03484604.

[24] N. Tuncbag, A. Gursoy, R. Nussinov, and O. Keskin, "Predicting protein-protein interactions on a proteome scale by matching evolutionary and structural similarities at interfaces using PRISM," *Nat Protoc,* vol. 6, no. 9, pp. 1341-54, Aug 11 2011, doi: 10.1038/nprot.2011.367.

[25] R. A. Jordan, Y. El-Manzalawy, D. Dobbs, and V. Honavar, "Predicting protein-protein interface residues using local surface structural similarity," (in eng), *BMC bioinformatics,* vol. 13, pp. 41-41, 2012, doi: 10.1186/1471-2105-13-41.

[26] C. J. Tsai, S. L. Lin, H. J. Wolfson, and R. Nussinov, "A dataset of protein-protein interfaces generated with a sequence-order-independent comparison technique," *J Mol Biol,* vol. 260, no. 4, pp. 604-20, Jul 26 1996, doi: 10.1006/jmbi.1996.0424.

[27] J. J. Headd, Y. E. Ban, P. Brown, H. Edelsbrunner, M. Vaidya, and J. Rudolph, "Protein-protein interfaces: properties, preferences, and projections," *J Proteome Res,* vol. 6, no. 7, pp. 2576-86, Jul 2007, doi: 10.1021/pr070018+.

[28] H. M. Berman *et al.*, "The Protein Data Bank," *Nucleic Acids Res,* vol. 28, no. 1, pp. 235-42, Jan 1 2000, doi: 10.1093/nar/28.1.235.

[29] S. K. Burley *et al.*, "RCSB Protein Data Bank: biological macromolecular structures enabling research and education in fundamental biology, biomedicine, biotechnology and energy," *Nucleic Acids Res,* vol. 47, no. D1, pp. D464-D474, Jan 8 2019, doi: 10.1093/nar/gky1004.

[30] A. Gutmanas *et al.*, "PDBe: Protein Data Bank in Europe," *Nucleic Acids Res,* vol. 42, no. Database issue, pp. D285-91, Jan 2014, doi: 10.1093/nar/gkt1180.

[31] A. R. Kinjo *et al.*, "Protein Data Bank Japan (PDBj): updated user interfaces, resource description framework, analysis tools for large structures," *Nucleic Acids Res,* vol. 45, no. D1, pp. D282-D288, Jan 4 2017, doi: 10.1093/nar/gkw962.

[32] E. L. Ulrich *et al.*, "BioMagResBank," *Nucleic Acids Res,* vol. 36, no. Database issue, pp. D402-8, Jan 2008, doi: 10.1093/nar/gkm957.

[33] T. UniProt Consortium, "UniProt: the universal protein knowledgebase," *Nucleic Acids Res,* vol. 46, no. 5, p. 2699, Mar 16 2018, doi: 10.1093/nar/gky092.

[34] A. Andreeva, D. Howorth, C. Chothia, E. Kulesha, and A. G. Murzin, "SCOP2 prototype: a new approach to protein structure mining," *Nucleic Acids Res,* vol. 42, no. Database issue, pp. D310-4, Jan 2014, doi: 10.1093/nar/gkt1242.

[35] A. Andreeva, E. Kulesha, J. Gough, and A. G. Murzin, "The SCOP database in 2020: expanded classification of representative family and superfamily domains of known protein structures," *Nucleic Acids Res,* vol. 48, no. D1, pp. D376-D382, Jan 8 2020, doi: 10.1093/nar/gkz1064.

[36] S. El-Gebali *et al.*, "The Pfam protein families database in 2019," *Nucleic Acids Res,* vol. 47, no. D1, pp. D427-D432, Jan 8 2019, doi: 10.1093/nar/gky995.

[37] R. A. Laskowski, E. G. Hutchinson, A. D. Michie, A. C. Wallace, M. L. Jones, and J. M. Thornton, "PDBsum: a Web-based database of summaries and analyses of all PDB structures," *Trends Biochem Sci,* vol. 22, no. 12, pp. 488-90, Dec 1997, doi: 10.1016/s0968-0004(97)01140-7.

[38] R. A. Laskowski, J. Jablonska, L. Pravda, R. S. Varekova, and J. M. Thornton, "PDBsum: Structural summaries of PDB entries," *Protein Sci,* vol. 27, no. 1, pp. 129-134, Jan 2018, doi: 10.1002/pro.3289.

[39] A. Baspinar, E. Cukuroglu, R. Nussinov, O. Keskin, and A. Gursoy, "PRISM: a web server and repository for prediction of protein-protein interactions and modeling their 3D complexes," *Nucleic Acids Res,* vol. 42, no. Web Server issue, pp. W285-9, Jul 2014, doi: 10.1093/nar/gku397.

[40] Q. Xu and R. L. Dunbrack, Jr., "ProtCID: a data resource for structural information on protein interactions," *Nat Commun,* vol. 11, no. 1, p. 711, Feb 5 2020, doi: 10.1038/s41467-020-14301-4.

[41] Q. Xu and R. L. Dunbrack, Jr., "The protein common interface database (ProtCID)--a comprehensive database of interactions of homologous proteins in multiple crystal forms," *Nucleic Acids Res,* vol. 39, no. Database issue, pp. D761-70, Jan 2011, doi: 10.1093/nar/gkq1059.

[42] R. Mosca, A. Ceol, A. Stein, R. Olivella, and P. Aloy, "3did: a catalog of domain-based interactions of known three-dimensional structure," *Nucleic Acids Res,* vol. 42, no. Database issue, pp. D374-9, Jan 2014, doi: 10.1093/nar/gkt887.

[43] M. Su *et al.*, "Comparative Assessment of Scoring Functions: The CASF-2016 Update," *J Chem Inf Model,* vol. 59, no. 2, pp. 895-913, Feb 25 2019, doi: 10.1021/acs.jcim.8b00545.
[44] G. a. D. Van Rossum, Fred L., *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.
[45] *pandas-dev/pandas: Pandas 1.1.3*. (2020). Zenodo. [Online]. Available: https://doi.org/10.5281/zenodo.4067057
[46] C. R. Harris *et al.*, "Array programming with NumPy," *Nature,* vol. 585, no. 7825, pp. 357-362, Sep 2020, doi: 10.1038/s41586-020-2649-2.
[47] P. Virtanen *et al.*, "SciPy 1.0: fundamental algorithms for scientific computing in Python," *Nat Methods,* vol. 17, no. 3, pp. 261-272, Mar 2020, doi: 10.1038/s41592-019-0686-2.
[48] P. Kunzmann and K. Hamacher, "Biotite: a unifying open source computational biology framework in Python," *BMC Bioinformatics,* vol. 19, no. 1, p. 346, Oct 1 2018, doi: 10.1186/s12859-018-2367-z.
[49] F. Pedregosa, Varoquaux, G.,;Gramfort, A. ,Michel, V., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.
[50] *SQLite*. (2020). [Online]. Available: https://www.sqlite.org/index.html
[51] N. Tuncbag, A. Gursoy, E. Guney, R. Nussinov, and O. Keskin, "Architectures and functional coverage of protein-protein interfaces," *J Mol Biol,* vol. 381, no. 3, pp. 785-802, Sep 5 2008, doi: 10.1016/j.jmb.2008.04.071.
[52] W. Humphrey, A. Dalke, and K. Schulten, "VMD: visual molecular dynamics," *J Mol Graph,* vol. 14, no. 1, pp. 33-8, 27-8, Feb 1996, doi: 10.1016/0263-7855(96)00018-5.
[53] Schrodinger, LLC, "The PyMOL Molecular Graphics System, Version 1.8," November, 2015.
[54] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in science & engineering,* vol. 9, 3, pp. 90-95, 2007.
[55] H. Berman, K. Henrick, and H. Nakamura, "Announcing the worldwide Protein Data Bank," *Nat Struct Biol,* vol. 10, no. 12, p. 980, Dec 2003, doi: 10.1038/nsb1203-980.
[56] S. Miller, J. Janin, A. M. Lesk, and C. Chothia, "Interior and surface of monomeric proteins," *J Mol Biol,* vol. 196, no. 3, pp. 641-56, Aug 5 1987, doi: 10.1016/0022-2836(87)90038-6.
[57] A. Shrake and J. A. Rupley, "Environment and exposure to solvent of protein atoms. Lysozyme and insulin," *J Mol Biol,* vol. 79, no. 2, pp. 351-71, Sep 15 1973, doi: 10.1016/0022-2836(73)90011-9.
[58] G. Zhao *et al.*, "Mature HIV-1 capsid structure by cryo-electron microscopy and all-atom molecular dynamics," *Nature,* vol. 497, no. 7451, pp. 643-6, May 30 2013, doi: 10.1038/nature12162.
[59] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, "CHARMM: A program for macromolecular energy, minimization, and dynamics calculations," *Journal of Computational Chemistry,* vol. 4, no. 2, pp. 187-217, 1983, doi: https://doi.org/10.1002/jcc.540040211.
[60] K. A. Iyer, Y. Hu, A. R. Nayak, N. Kurebayashi, T. Murayama, and M. Samso, "Structural mechanism of two gain-of-function cardiac and

skeletal RyR mutations at an equivalent site by cryo-EM," *Sci Adv,* vol. 6, no. 31, p. eabb2964, Jul 2020, doi: 10.1126/sciadv.abb2964.

[61] M. Kosloff and R. Kolodny, "Sequence-similar, structure-dissimilar protein pairs in the PDB," *Proteins,* vol. 71, no. 2, pp. 891-902, May 1 2008, doi: 10.1002/prot.21770.

[62] S. Mukherjee and Y. Zhang, "MM-align: a quick algorithm for aligning multiple-chain protein complex structures using iterative dynamic programming," *Nucleic Acids Res,* vol. 37, no. 11, p. e83, Jun 2009, doi: 10.1093/nar/gkp318.

[63] Y. Zhang and J. Skolnick, "TM-align: a protein structure alignment algorithm based on the TM-score," *Nucleic Acids Res,* vol. 33, no. 7, pp. 2302-9, 2005, doi: 10.1093/nar/gki524.

[64] R. L. Owen *et al.*, "Low-dose fixed-target serial synchrotron crystallography," *Acta Crystallogr D Struct Biol,* vol. 73, no. Pt 4, pp. 373-378, Apr 1 2017, doi: 10.1107/S2059798317002996.

[65] A. G. Gabdulkhakov, O. S. Kostareva, I. A. Kolyadenko, A. O. Mikhaylina, L. I. Trubitsina, and S. V. Tishchenko, "[Incorporation of Copper Ions into T2/T3 Centers of Two-Domain Laccases]," *Mol Biol (Mosk),* vol. 52, no. 1, pp. 29-35, Jan-Feb 2018, doi: 10.7868/S0026898418010056.

[66] A. Gabdulkhakov *et al.*, "Investigations of Accessibility of T2/T3 Copper Center of Two-Domain Laccase from Streptomyces griseoflavus Ac-993," *Int J Mol Sci,* vol. 20, no. 13, Jun 28 2019, doi: 10.3390/ijms20133184.

[67] O. Tange, *GNU Parallel 2018*. Ole Tange, 2018.

[68] M. Bayer, "SQLAlchemy," in *The Architecture of Open Source Applications Volume II: Structure, Scale*, A. a. W. Brown, Greg Ed. Mountain View: aosabook.org, 2012.

[69] Y. Zhang and J. Skolnick, "Scoring function for automated assessment of protein structure template quality," *Proteins,* vol. 57, no. 4, pp. 702-10, Dec 1 2004, doi: 10.1002/prot.20264.

[70] W. Wong *et al.*, "Cryo-EM structure of the Plasmodium falciparum 80S ribosome bound to the anti-protozoan drug emetine," *Elife,* vol. 3, Jun 9 2014, doi: 10.7554/eLife.03080.

[71] Q. Yao *et al.*, "A structural mechanism for bacterial autotransporter glycosylation by a dodecameric heptosyltransferase family," *Elife,* vol. 3, Oct 13 2014, doi: 10.7554/eLife.03714.

[72] W. Kabsch and C. Sander, "Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features," *Biopolymers,* vol. 22, no. 12, pp. 2577-637, Dec 1983, doi: 10.1002/bip.360221211.

[73] W. G. Touw *et al.*, "A series of PDB-related databanks for everyday needs," *Nucleic Acids Res,* vol. 43, no. Database issue, pp. D364-8, Jan 2015, doi: 10.1093/nar/gku1028.

[74] W. Dai, A. Wu, L. Ma, Y. X. Li, T. Jiang, and Y. Y. Li, "A novel index of protein-protein interface propensity improves interface residue recognition," *BMC Syst Biol,* vol. 10, no. Suppl 4, p. 112, Dec 23 2016, doi: 10.1186/s12918-016-0351-7.

Appendix A:

Supplementary Tables

Supplementary Table 1: CHARMM Van der Waals radii definitions

Amino Acid Code	Atom	VDW radii
ALA	N	1.83
	H	0.8
	CA	2.265
	CB	2.165
	C	1.87
	O	1.55
ARG	N	1.83
	H	0.8
	CA	2.265
	CB	2.235
	CG	2.235
	CD	2.235
	NE	1.83
	HE	0.8
	CZ	1.87
	NH1	1.83
	HH11	0.6
	HH12	0.6
	NH2	1.83
	HH21	0.6
	HH22	0.6
	C	1.87
	O	1.55
ARGN	N	1.83
	H	0.8
	CA	2.265
	CB	2.235
	CG	2.235
	CD	2.235
	NE	1.83
	HE	0.8
	CZ	1.87
	NH1	1.83
	HH11	0.6
	HH12	0.6
	NH2	1.83
	HH21	0.6
	C	1.87
	O	1.55
ASN	N	1.83
	H	0.8
	CA	2.265
	CB	2.235
	CG	1.87
	OD1	1.55
	ND2	1.83
	HD21	0.8
	HD22	0.8
	C	1.87
	O	1.55
	AD1	1.55
	AD2	1.83
ASP	N	1.83
	H	0.8
	CA	2.265
	CB	2.235
	CG	1.87
	OD1	1.55
	OD2	1.66
	C	1.87
	O	1.55
ASPH	N	1.83
	H	0.8
	CA	2.265
	CB	2.235
	CG	1.87
	OD1	1.52
	OD2	1.55
	HD	0.8
	C	1.87
	O	1.55
CYS	N	1.83
	H	0.8
	CA	2.265
	CB	2.235
	SG	1.89
	C	1.87
	O	1.55
GLN	N	1.83
	H	0.8
	CA	2.265
	CB	2.235
	CG	2.235
	CD	1.87
	OE1	1.55
	NE2	1.83
	HE21	0.8
	HE22	0.8
	C	1.87
	O	1.55
	AE1	1.55
	AE2	1.83
GLU	N	1.83
	H	0.8
	CA	2.265
	CB	2.235
	CG	2.235
	CD	1.87
	OE1	1.66
	OE2	1.66
	C	1.87
	O	1.55

Amino Acid Code	Atom	VDW radii
GLUH	N	1.83
	H	0.8
	CA	2.265
	CB	2.235
	CG	2.235
	CD	1.87
	OE1	1.52
	OE2	1.55
	HE	0.8
	C	1.87
	O	1.55
GLY	N	1.83
	H	0.8
	CA	2.265
	C	1.87
	O	1.55
HIS	N	1.83
	H	0.8
	CA	2.265
	CB	2.235
	CG	2.04
	ND1	1.72
	HD1	0.8
	CD2	2.1
	NE2	1.72
	CE1	2.1
	C	1.87
	O	1.55
	AE1	2.1
	AE2	1.72
ILE	N	1.83
	H	0.8
	CA	2.265
	CB	2.265
	CG2	2.165
	CG1	2.235
	CD1	2.165
	C	1.87
	O	1.55
	CD	2.165
LEU	N	1.83
	H	0.8
	CA	2.265
	CB	2.235
	CG	2.265
	CD1	2.165
	CD2	2.165
	C	1.87
	O	1.55
LYS	N	1.83
	H	0.8
	CA	2.265
	CB	2.235
	CG	2.235
	CD	2.235
	CE	2.235
	NZ	1.65
	HZ	0.6
	HZ1	0.6
	HZ2	0.6
	HZ3	0.6
	C	1.87
	O	1.55
LYSN	N	1.83
	H	0.8
	CA	2.265
	CB	2.235
	CG	2.235
	CD	2.235
	CE	2.235
	NZ	1.65
	HZ1	0.8
	HZ2	0.8
	C	1.87
	O	1.55
MET	N	1.83
	H	0.8
	CA	2.265
	CB	2.235
	CG	2.235
	SD	1.97
	CE	2.165
	C	1.87
	O	1.55
PHE	N	1.83
	H	0.8
	CA	2.265
	CB	2.235
	CG	2.235
	CD1	1.99
	CD2	1.99
	CE1	1.99
	CE2	1.99
	CZ	1.99
	C	1.87
	O	1.55
PRO	N	1.83
	CD	2.235
	CA	2.265
	CB	2.235
	CG	2.235
	C	1.87
	O	1.55
SER	N	1.83
	H	0.8
	CA	2.265
	CB	2.235
	OG	1.55
	HG	0.76
	C	1.87
	O	1.55
THR	N	1.83
	H	0.8
	CA	2.265
	CB	2.265
	OG1	1.55
	HG1	0.76
	CG2	2.165
	C	1.87
	O	1.55
TRP	N	1.83
	H	0.8
	CA	2.265
	CB	2.235
	CG	2.04
	CD2	2.04

Amino Acid Code	Atom	VDW radii
	CE2	2.04
	CE3	1.99
	CD1	2.1
	NE1	1.72
	HE1	0.8
	CZ2	1.99
	CZ3	1.99
	CH2	1.99
	C	1.87
	O	1.55
TYR	N	1.83
	H	0.8
	CA	2.265
	CB	2.235
	CG	2.04
	CD1	1.99
	CE1	1.99
	CD2	1.99
	CE2	1.99
	CZ	2.04
	OH	1.55
	HH	0.76
	C	1.87
	O	1.55
VAL	N	1.83
	H	0.8
	CA	2.265
	CB	2.265
	CG1	2.165
	CG2	2.165
	C	1.87
	O	1.55
HSC	N	1.83
	H	0.8
	CA	2.265
	CB	2.2.35
	CG	2.04
	CD2	2.1
	ND1	1.72
	HD1	0.8
	CE1	2.1
	NE2	1.72
	HE2	0.8
	C	1.87
	O	1.55
HSD	N	1.83
	H	0.8
	CA	2.265
	CB	2.235
	CG	2.04
	ND1	1.72
	CE1	2.1
	CD2	2.1
	NE2	1.72
	HE2	0.8
ACE	C	1.87
	O	1.55
	C	1.87
	O	1.55
	CH3	2.165

Supplementary Table 2: Default Van der Waals radii definitions with digits

Atom	VDW radii
AD1	1.55
AD2	1.83
AE1	2.1
AE2	1.83
C	1.87
CA	2.265
CB	2.265
CD	2.235
CD1	2.165
CD2	2.165
CE	2.235
CE1	2.1
CE2	2.04
CE3	1.99
CG	2.265
CG1	2.235
CG2	2.165
CH2	1.99
CH3	2.165
CZ	2.04
CZ2	1.99
CZ3	1.99
H	0.8
HD	0.8
HD1	0.8
HD21	0.8
HD22	0.8
HE	0.8
HE1	0.8
HE2	0.8
HE21	0.8

Atom	VDW radii
HE22	0.8
HG	0.76
HG1	0.76
HH	0.76
HH11	0.6
HH12	0.6
HH21	0.6
HH22	0.6
HZ	0.6
HZ1	0.8
HZ2	0.8
HZ3	0.6
N	1.83
ND1	1.72
ND2	1.83
NE	1.83
NE1	1.72
NE2	1.83
NH1	1.83
NH2	1.83
NZ	1.65
O	1.55
OD1	1.66
OD2	1.66
OE1	1.66
OE2	1.66
OG	1.55
OG1	1.55
OH	1.55
SD	1.97

Supplementary Table 3: Default Van der Waals radii definitions without digits

Atom	VDW radii
AD	1.83
AE	2.1
C	1.87
CA	2.265
CB	2.265
CD	2.235
CE	2.235
CG	2.265
CH	2.165
CZ	2.04
H	0.8
HD	0.8
HE	0.8
HG	0.76
HH	0.76
HZ	0.8
N	1.83
ND	1.83
NE	1.83
NH	1.83
NZ	1.65
OD	1.66
OE	1.66
OG	1.55
OH	1.55
SD	1.97

Supplementary Table 3: List of results for comparison as an example for the group with main ID 20374 with relevant information

cid	interface_1	interface_2	rmsd	tmscore_int1	tmscore_int2
2578870	6Q5P_D_E	6Q5P_A_D	1.93	0.72012	0.72012
2578871	6Q5P_D_E	6Q5P_E_F	0.31	0.98655	0.98655
2578872	6Q5P_D_E	6Q5P_A_B	0.48	0.96865	0.96865
2578873	6Q5P_D_E	6Q5P_C_E	2.03	0.71015	0.71015
2578874	6Q5P_D_E	6Q5P_C_D	2.16	0.64218	0.64218
2578875	6Q5P_D_E	6Q5P_C_F	1.47	0.71589	0.71589
2578876	6Q5P_D_E	6Q5P_B_D	1.40	0.70623	0.70623
2578877	6Q5P_A_D	6Q5P_E_F	1.88	0.72542	0.72542
2578878	6Q5P_A_D	6Q5P_A_B	1.77	0.74842	0.74842
2578879	6Q5P_A_D	6Q5P_C_E	0.77	0.94258	0.94258
2578880	6Q5P_A_D	6Q5P_C_D	1.69	0.72815	0.72815
2578881	6Q5P_A_D	6Q5P_C_F	2.03	0.70788	0.70788
2578882	6Q5P_A_D	6Q5P_B_D	2.21	0.67607	0.67607
2578883	6Q5P_E_F	6Q5P_A_B	0.38	0.97956	0.97956
2578884	6Q5P_E_F	6Q5P_C_E	2.01	0.70645	0.70645
2578885	6Q5P_E_F	6Q5P_C_D	2.40	0.56625	0.56625
2578886	6Q5P_E_F	6Q5P_C_F	1.50	0.73108	0.73108
2578887	6Q5P_E_F	6Q5P_B_D	1.45	0.71246	0.71246
2578888	6Q5P_A_B	6Q5P_C_E	1.80	0.72758	0.72758
2578889	6Q5P_A_B	6Q5P_C_D	2.16	0.60240	0.60240
2578890	6Q5P_A_B	6Q5P_C_F	1.46	0.71229	0.71229
2578891	6Q5P_A_B	6Q5P_B_D	1.43	0.70951	0.70951
2578892	6Q5P_C_E	6Q5P_C_D	1.82	0.73055	0.73055
2578893	6Q5P_C_E	6Q5P_C_F	1.94	0.70786	0.70786
2578894	6Q5P_C_E	6Q5P_B_D	2.35	0.65587	0.65587
2578895	6Q5P_C_D	6Q5P_C_F	2.08	0.62160	0.62160
2578896	6Q5P_C_D	6Q5P_B_D	1.82	0.61117	0.61117
2578897	6Q5P_C_F	6Q5P_B_D	0.90	0.91940	0.91940

Supplementary Table 4: Number of structure counts and chain counts by years

Year	Struc. Count	Cum. Struc. Count	Chain Count	Cum. Chain Count
1972	1	1	1	1
1973	2	3	3	4
1974	1	4	1	5
1975	11	15	17	22
1976	17	32	27	49
1977	7	39	9	58
1978	2	41	2	60
1979	12	53	17	77
1980	8	61	15	92
1981	22	83	27	119
1982	37	120	60	179
1983	17	137	24	203
1984	26	163	54	257
1985	13	176	23	280
1986	18	194	22	302
1987	41	235	63	365
1988	75	310	133	498
1989	131	441	259	757
1990	194	635	325	1.082
1991	357	992	569	1.651
1992	480	1.472	851	2.502
1993	696	2.168	1.243	3.745
1994	941	3.109	1.688	5.433
1995	1.084	4.193	2.004	7.437
1996	1.282	5.475	2.342	9.779
1997	1.637	7.112	3.409	13.188
1998	1.960	9.072	4.161	17.349
1999	2.346	11.418	4.862	22.211
2000	2.699	14.117	6.127	28.338
2001	2.948	17.065	7.229	35.567
2002	3.085	20.150	7.892	43.459
2003	4.298	24.448	10.751	54.210
2004	4.969	29.407	12.899	67.109
2005	5.827	35.234	13.918	81.027
2006	6.196	41.430	15.531	96.558
2007	7.378	48.808	18.798	115.356
2008	6.424	55.232	17.456	132.812
2009	7.598	62.830	22.951	155.763
2010	8.067	70.897	23.855	179.618
2011	8.329	79.226	24.922	205.450
2012	9.022	88.248	25.482	230.022
2013	9.496	97.744	33.503	263.525
2014	9.211	106.955	37.317	300.842
2015	10.007	116.962	37.028	337.870
2016	10.633	127.595	41.854	379.724
2017	11.956	139.551	44.545	424.269
2018	11.031	150.582	47.370	471.639
2019	11.103	161.685	55.229	526.868
2020	4.337	166.022	28.990	555.858

Appendix B:

Supplementary Codes and Algorithms

Supplementary Code 1: Example set of SQL queries for PIFace DB

```
INSERT INTO
    fasta (pdb_id, entity_id, chain_id, info, source_organism,
    ncbi_taxonomy, sequence, insert_time)
VALUES
    ({pdb_id}, {entity_id, {chain_id}, {info}, {source_organism},
    {ncbi_taxonomy}, {sequence}, {insert_time});
```

```
INSERT INTO
     cif (pdb_id, insert_time)
VALUES
    ({pdb_id}, {insert_time});
```

```
UPDATE
    cif
SET
    chain_count = {chain_count},
    combination_count = {combination_count} WHERE pdb_id = {pdb_id};
```

```
INSERT INTO
    interfaces (dimer_id, pdb_id, chain_1, chain_2,
    chain_1_contacting, chain_2_contacting, chain_1_nearby,
    chain_2_nearby, insert_time)
VALUES
    ({dimer_id}, {pdb_id}, {chain_1}, {chain_2}, {chain_1_contacting},
    {chain_2_contacting}, {chain_1_nearby}, {chain_2_nearby},
    {insert_time});
```

```
INSERT INTO
    interfaces_cif (dimer_id, cif_file, insert_time)
VALUES
    ({dimer_id}, {cif_file}, {insert_time});
```

Supplementary Code 2A: Codes for structural comparisons of chains

```
import sqlite3
from sqlalchemy import create_engine, and_, update, Column, Integer, MetaData, String, Boolean,
                       Table, DateTime, Float
from sqlalchemy.sql.expression import bindparam
import pandas as pd
import os
import random
import string
import argparse
from datetime import datetime
import subprocess
from itertools import combinations

def read_mmalign(mm_file):
  with open(mm_file, 'r') as mmalign_result:
    result_list = mmalign_result.readlines()

  int_1 = result_list[7].split("/")[-1].split(":")[0].split(".")[0]
  int_2 = result_list[8].split("/")[-1].split(":")[0].split(".")[0]
  al = int(result_list[12].split(",")[0].split("=")[1])
  rd = float(result_list[12].split(",")[1].split("=")[1])
  si = float(result_list[12].split(",")[2].split("=")[2])
  tm_1 = float(result_list[13].split("=")[1].split("(")[0])
  tm_2 = float(result_list[14].split("=")[1].split("(")[0])
  ar = ''.join(result_list[18:21])

  tr_file = mm_file.replace(".mmalign", ".trans")

  with open(tr_file, 'r') as trans_result:
    trans_list = trans_result.readlines()

  data = [list(filter(None, i.strip().split(" "))) for i in trans_list if i[0] in ["0", "1", "2"]]

  t0_v, t1_v, t2_v = float(data[0][1]), float(data[1][1]), float(data[2][1])
  u00_v, u01_v, u02_v = float(data[0][2]), float(data[0][3]), float(data[0][4])
  u10_v, u11_v, u12_v = float(data[1][2]), float(data[1][3]), float(data[1][4])
  u20_v, u21_v, u22_v = float(data[2][2]), float(data[2][3]), float(data[2][4])

  return int_1, int_2, al, rd, si, tm_1, tm_2, ar, t0_v, t1_v, t2_v, u00_v, u01_v, u02_v, u10_v, u11_v, u12_v,
u20_v, u21_v, u22_v

def write_results(file_names, group_id):
  chunk_size = 10000
  mmalign_files = file_names[:]
  mmalign_results = [read_mmalign(mmalign_file) for mmalign_file in mmalign_files]

  with open('comparisons.log', 'a') as f:
    print(f"--- Number of comparisons for MySQL DB: {len(mmalign_results)} - {datetime.now()} ---",
file=f)

  chunks = [mmalign_results[i * chunk_size:(i + 1) * chunk_size] for i in
       range((len(mmalign_results) + chunk_size - 1) // chunk_size)]
```

```
 for chunk in chunks:
     with engine.begin() as conn:
       stmt = comparisons.insert().values({"interface_1": bindparam("interface_1"),
                "interface_2":   bindparam("interface_2"), "group_id": group_id, "aligned_length":
               bindparam("aligned_length"), "rmsd": bindparam("rmsd"),
               sequence_identity":bindparam("sequence_identity"),
               "tmscore_int1": bindparam("tmscore_int1"), "tmscore_int2": bindparam("tmscore_int2"),
                "aligned_residues": bindparam("aligned_residues"), "t0": bindparam("t0"),
               "t1": bindparam("t1"), "t2": bindparam("t2"), "u00": bindparam("u00"), "u01":
bindparam("u01"),        "u02": bindparam("u02"), "u10": bindparam("u10"), "u11": bindparam("u11"),
               "u12": bindparam("u12"), "u20": bindparam("u20"), "u21": bindparam("u21"),
                "u22": bindparam("u22"), "processed": 0, "insert_time": datetime.now()})

       conn.execute(stmt, [{"interface_1": result[0], "interface_2": result[1], "aligned_length": result[2],
               "rmsd": result[3], "sequence_identity": result[4], "tmscore_int1": result[5],
               "tmscore_int2": result[6], "aligned_residues": result[7], "t0": result[8], "t1": result[9],
               "t2": result[10], "u00": result[11], "u01": result[12], "u02": result[13], "u10": result[14],
               "u11": result[15], "u12": result[16], "u20": result[17], "u21": result[18], "u22": result[19]}
                for result in chunk])

def get_random_string(length):
   letters_and_digits = string.ascii_lowercase + string.digits
   return "".join([random.choice(letters_and_digits) for i in range(length)])

engine =
create_engine('mysql://interfaceUser:interfacePass20@interactome.ku.edu.tr:3306/interfaceDB')
metadata = MetaData()
comparisons = Table('upd_chain2_comparison_seq', metadata, autoload=True,
autoload_with=engine)

with open('upd_chain2_unique_groups.list', "r") as f:
   unique_all_groups = [i.strip(" \n").split(", ") for i in f.readlines()]

group_id = 0
if os.path.exists('upd_comparisons2.log'):
   with engine.connect() as conn:
   # Identical sequences in group comparisons
     comp_seq_df = pd.read_sql_query("SELECT * FROM upd_chain2_comparison_seq", conn)

   last_index = list(comp_seq_df.group_id)[-1]
   unique_all_groups = unique_all_groups[last_index+1:]
   group_id = last_index + 1

for group in unique_all_groups:
   all_combs = list(combinations(group, 2))

   with open('upd_comparisons2.log', 'a') as f:
     print(f"--- Total number of current combinations: {len(all_combs)} - {datetime.now()} ---", file=f)
   with open('upd_comparisons2.log', 'a') as f:
     print(f"--- Writing the new comparison codes. - {datetime.now()} ---", file=f)

   rand_str = f"{get_random_string(10)}"
   results_file = rand_str + ".mmalignccommands"
   mysql_file = rand_str + ".mysql"
```

```
 lines = []

   with open(results_file, 'a') as f:
      f.write("".join([f'TMalign
/kuacc/users/zabali16/IDSeq2Struc/ChainComparisons/cif2Files/{comb[0]}.cif
                /kuacc/users/zabali16/IDSeq2Struc/ChainComparisons/cif2Files/{comb[1]}.cif -m
mmalignDir/{comb[0]}-{comb[1]}.trans >          mmalignDir/{comb[0]}-{comb[1]}.mmalign\n' for comb
in all_combs]))

   with open(mysql_file, 'a') as f:
      f.write("".join([f'mmalignDir/{comb[0]}-{comb[1]}.mmalign\n' for comb in all_combs]))

   if not os.path.exists("mmalignDir/"):
      os.mkdir("mmalignDir/")

   with open('upd_comparisons2.log', 'a') as f:
      print(f"--- Starting comparisons. - {datetime.now()} ---", file=f)

   os.system(f"parallel --jobs 12 --joblog log.file < {results_file}")

   with open('log.file', 'r') as f:
      log_len = len([i.strip("\n") for i in f.readlines()]) - 1

   # Check if all processes are completed
   while True:
      if log_len == len(all_combs):

         with open('upd_comparisons2.log', 'a') as f:
            print(f"--- Writing to MySQL Database. - {datetime.now()} ---", file=f)
         with open(mysql_file, 'r') as f:
            mm_files = [i.strip("\n") for i in f.readlines()]

         write_results(mm_files, group_id)
         os.remove(f"{mysql_file}")
         os.remove('log.file')

         break

      else:
         time.sleep(0.1)
         continue

   os.remove(f"{results_file}")
   os.system("rm -r mmalignDir/")
   group_id += 1

os.rmdir('mmalignDir/')
```

Supplementary Code 2B: Part of the input file for structural comparison

```
Line 3  6MSK_q_r#q, 6MSK_Q_R#Q

Line 4  6Q3G_QQ_Y7#Y7, 6Q3G_KE_Y7#KE, 6Q3G_KE_QQ#QQ, Q3G_C6_gG#C6,
        6Q3G_IR_X1#IR, 6Q3G_Y8_bQ#bQ, 6Q3G_I1_SE#I1, Q3G_CR_X4#X4,
        6Q3G_QG_f1#QG, 6Q3G_IQ_Q4#IQ, 6Q3G_Q3_X4#Q3, Q3G_CG_aE#CG,
        6Q3G_I3_I4#I4, 6Q3G_I7_gR#I7, 6Q3G_Q7_QR#QR, Q3G_AE_g7#g7,
        6Q3G_C6_f4#f4, 6Q3G_CQ_I7#CQ, 6Q3G_I8_IR#I8, Q3G_CQ_gR#gR,
        6Q3G_IG_Q1#Q1, 6Q3G_I6_YR#I6, 6Q3G_IL_IQ#IL, Q3G_SE_Y6#SE,
        6Q3G_C1_bQ#C1, 6Q3G_I8_X1#X1, 6Q3G_f4_gG#gG, Q3G_C4_CG#C4,
        6Q3G_C3_QG#C3, 6Q3G_C4_aE#aE, 6Q3G_AE_C8#AE, Q3G_QR_YL#YL,
        6Q3G_C1_Y8#Y8, 6Q3G_I3_QL#I3, 6Q3G_C3_f1#f1, Q3G_Q7_YL#Q7,
        6Q3G_C8_g7#C8, 6Q3G_I4_QL#QL, 6Q3G_CL_YR#YR, Q3G_CR_Q3#CR,
        6Q3G_IL_Q4#Q4, 6Q3G_IG_Q8#IG, 6Q3G_CL_I6#CL, Q3G_I1_Y6#Y6,
        6Q3G_Q1_Q8#Q8

Line 5  6TZ4_K_ZA#K, 6TZ4_LA_M#LA, 6TZ4_BB_DA#BB, 6E8G_AB_R#AB,
        6E8G_MA_UB#MA, 6E8G_F_W#F, 6TZ4_VA_X#VA, 6E8G_KA_UA#KA,
        6E8G_EA_QB#QB, 6TZ4_M_NB#M, 6E8G_GB_SA#SA, 6E8G_N_WA#N,
        6TZ4_FB_HA#FB, 6TZ4_JB_X#X, 6E8G_MA_T#T, 6E8G_MB_W#W,
        6E8G_IA_P#IA, 6TZ4_JA_K#JA, 6TZ4_DB_FA#DB, 6TZ4_O_PB#O,
        6TZ4_XA_Z#XA, 6E8G_CB_KA#CB, 6TZ4_NA_O#NA, 6E8G_L_YA#L,
        6E8G_J_KB#J, 6TZ4_RB_S#S, 6TZ4_A_Z#Z, 6E8G_B_J#B,
        TZ4_02_XA#02, 6TZ4_FA_G#FA, 6E8G_EA_SB#EA, 6E8G_B_EB#EB,
        6E8G_D_H#H, 6E8G_QA_Y#Y, 6TZ4_02_V#V, 6TZ4_FB_Q#Q,
        6TZ4_E_TA#E, 6TZ4_PB_RA#PB, 6E8G_N_R#R, 6TZ4_BA_C#BA,
        6TZ4_I_LB#I, 6E8G_AB_CA#CA, 6TZ4_LA_LB#LB, 6E8G_L_OB#OB,
        6TZ4_DA_E#DA, 6TZ4_PA_Q#PA, 6E8G_F_UA#UA, 6E8G_GA_QB#GA,
        6TZ4_TA_V#TA, 6E8G_OA_T#OA, 6E8G_GB_Y#GB, 6E8G_AA_D#D,
        6E8G_IA_YA#YA, 6E8G_H_MB#MB, 6E8G_CA_SB#SB, 6TZ4_C_DB#C,
        6E8G_IB_KB#KB, 6E8G_OB_QA#QA, 6TZ4_G_HB#G, 6TZ4_JB_NA#JB,
        6TZ4_HA_I#HA, 6E8G_EB_UB#UB, 6TZ4_BA_ZA#ZA, 6TZ4_RA_S#RA
        4Z27_A_B#A, 4Z2P_A_B#A, 4Z28_A_B#B, 4Z2P_A_B#B, Z2V_A_B#B,
        4Z6J_A_B#B, 4Z27_A_B#B, 4Z2V_A_B#A

Line 6  5OGK_A_B#A, 5OGK_A_B#B

Line 7  2VSS_B_D#D, 2VSU_C_F#F

Line 8  4NHH_O_Q#O, 4NHH_E_G#E, 4NHH_H_L#H
```

Supplementary Code 3A: Code for sub-grouping of sequence groups by structural comparison results

```
import pandas as pd
import numpy as np
from sklearn.cluster import AgglomerativeClustering

id_groups_df = pd.DataFrame(columns=['chain_id', 'main_id', 'sub_id', 'group_id', 'representative'])

group_ids = list(chain_rmsd_df.group_id.unique())
rmsd_threshold = 1.5

for group_id in group_ids:

  group_rmsd = chain_rmsd_df[chain_rmsd_df.group_id == group_id]

  if len(group_rmsd[group_rmsd.rmsd > rmsd_threshold]) == 0:

    interface_ids = list(group_rmsd.interface_1.unique()) + list(group_rmsd.interface_2[-1:])

    id_groups_df = id_groups_df.append(
      [{'chain_id': item, 'main_id': group_id, 'sub_id': None, 'group_id': group_id} for item in
                                                                                    interface_ids])
  elif len(group_rmsd[group_rmsd.rmsd > rmsd_threshold]) > 0:

    interface_ids = list(group_rmsd.interface_1.unique()) + list(group_rmsd.interface_2[-1:])

    if len(interface_ids) == 2:

      id_groups_df = id_groups_df.append(
        [{'chain_id': item, 'main_id': group_id, 'sub_id': None, 'group_id': None} for item in
                                                                                    interface_ids])
    else:
      rmsd_matrix = np.zeros((len(interface_ids), len(interface_ids)))
      triu = np.triu_indices(len(interface_ids), +1)
      tril = np.tril_indices(len(interface_ids), -1)
      rmsd_matrix[triu] = list(group_rmsd.rmsd)
      rmsd_matrix[tril] = rmsd_matrix.T[tril]

      clustering = AgglomerativeClustering(linkage='complete', affinity='precomputed',
                           distance_threshold=rmsd_threshold, n_clusters=None).fit(rmsd_matrix)
      labeled = list(zip(clustering.labels_, interface_ids))

      np_counts = list(zip(list(np.unique(clustering.labels_, return_counts=True)[0]),
                 list(np.unique(clustering.labels_, return_counts=True)[1])))
      np_sorted = sorted(np_counts, key=lambda x: x[1], reverse=True)
      np_assigned = list(enumerate([i[0] for i in np_sorted if i[1] > 1], start=97))
      trans_dict = {i[1]: chr(i[0]) for i in np_assigned}

      sub_ids = [(trans_dict[i], str(group_id) + "_" + trans_dict[i]) if i in trans_dict.keys() else (None,
                                                                    None) for i in clustering.labels_]

      id_groups_df = id_groups_df.append([{'chain_id': item[0], 'main_id': group_id, 'sub_id':
                           item[1][0], 'group_id': item[1][1]} for item in list(zip(interface_ids, sub_ids))])
```

Supplementary Code 3B: Representative selection example for group with group ID 4557_a

1. Select comparisons for the interface IDs in the group.

cid	interface_1	interface_2	rmsd	tmscore_int1	tmscore_int2
2578871	6Q5P_D_E	6Q5P_E_F	0.31	0.98655	0.98655
2578872	6Q5P_D_E	6Q5P_A_B	0.48	0.96865	0.96865
2578883	6Q5P_E_F	6Q5P_A_B	0.38	0.97956	0.97956

2. Sum RMSD values for each of the unique interface IDs.

Interface ID	Sum of RMSD Values
6Q5P_A_B	0,86
6Q5P_E_F	0.69
6Q5P_D_E	0.79

3. Choose the interface ID with the smallest sum of RMSD value.

Interface ID	Sum of RMSD Values
6Q5P_E_F	0.69
6Q5P_D_E	0.79
6Q5P_A_B	0.86

6Q5P_E_F is selected as the representative for group with ID 4557_a.