FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Lukáš Jendele

Development of web-based interface for visualization of protein-ligand interaction sites and their conservation

Department of Software Engineering

Supervisor of the bachelor thesis:	RNDr. David Hoksza, Ph.D.
Study programme:	Computer Science
Study branch:	General Computer Science

Prague 2017

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In Prague, May 16, 2017 Lukáš Jendele

This work is dedicated to my family and friends: thank you for your constant support in every move I have made.

Huge thanks to my advisor RNDr. David Hoksza, Ph.D. for his advice, guidance and comments throughout the whole project.

I would also like to thank people who helped me, namely RNDr. David Sehnal, Ph.D. for his infinite help with LiteMol, Mgr. Radoslav Krivák for his advice on P2Rank experiments, Mgr. Marian Novotný, Ph.D. for consultation on biological matter and MSc. Mayya Sedova for quick implementation of feature requests into Protael.

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme ”Projects of Large Research, Development, and Innovations Infrastructures” (CESNET LM2015042), is greatly appreciated.

Title: Development of web-based interface for visualization of protein-ligand interaction sites and their conservation

Author: Lukáš Jendele

Department: Department of Software Engineering

Supervisor: RNDr. David Hoksza, Ph.D., Department of Software Engineering

Abstract: Proteins are fundamental building blocks of all living organisms. They perform their function by binding to other molecules. This thesis deals with interactions between proteins and small molecules (so called ligands) because most of the currently used drugs are small molecules. While there are several tools that can predict these interactions, they are almost none for their visualization. Thus, we built a new visualization website by combining several protein visualizers together. Since evolutionary homology correlates with binding sites, our web interface also displays homology for comparison. We developed several ways how to calculate homology, and used it to improve detection of protein-ligand binding sites in our experiments. Here we present PrankWeb, a modern web application for structure and sequence visualization of a protein and its protein-ligand binding sites as well as evolutionary homology. We hope that it will provide a quick and convenient way for scientists to analyze proteins.

Keywords: bioinformatics protein protein-ligand interaction GUI web

Contents

Introduction

Background

Proteins are large molecules controlling virtually all processes in living organisms. A protein consists of one or multiple sequences of 20 different types of amino acids. These amino acids are linked together, forming *polypetide chains*. The order of the amino acids, in which they are linked, as well as the length of the sequence, is defined by genetic information encoded in DNA. [1]

Proteins are responsible for a wide range of functions, which are crucial for life. As enzymes, they speed up chemical reactions of the metabolism. As antibodies, they neutralize pathogens and protect the body. Proteins can also serve as building blocks of connecting tissues such as ligaments, tendons, cartilage, etc. Transport proteins deliver materials through our blood circulation to cells. In cells, proteins perform nearly every task ranging from slicing DNA sequences to reinforcing the cell's shape.

Proteins fold into a 3-dimensional structure. In biology, we distinguish primary, secondary, tertiary, and quaternary structure (see Figure 1):

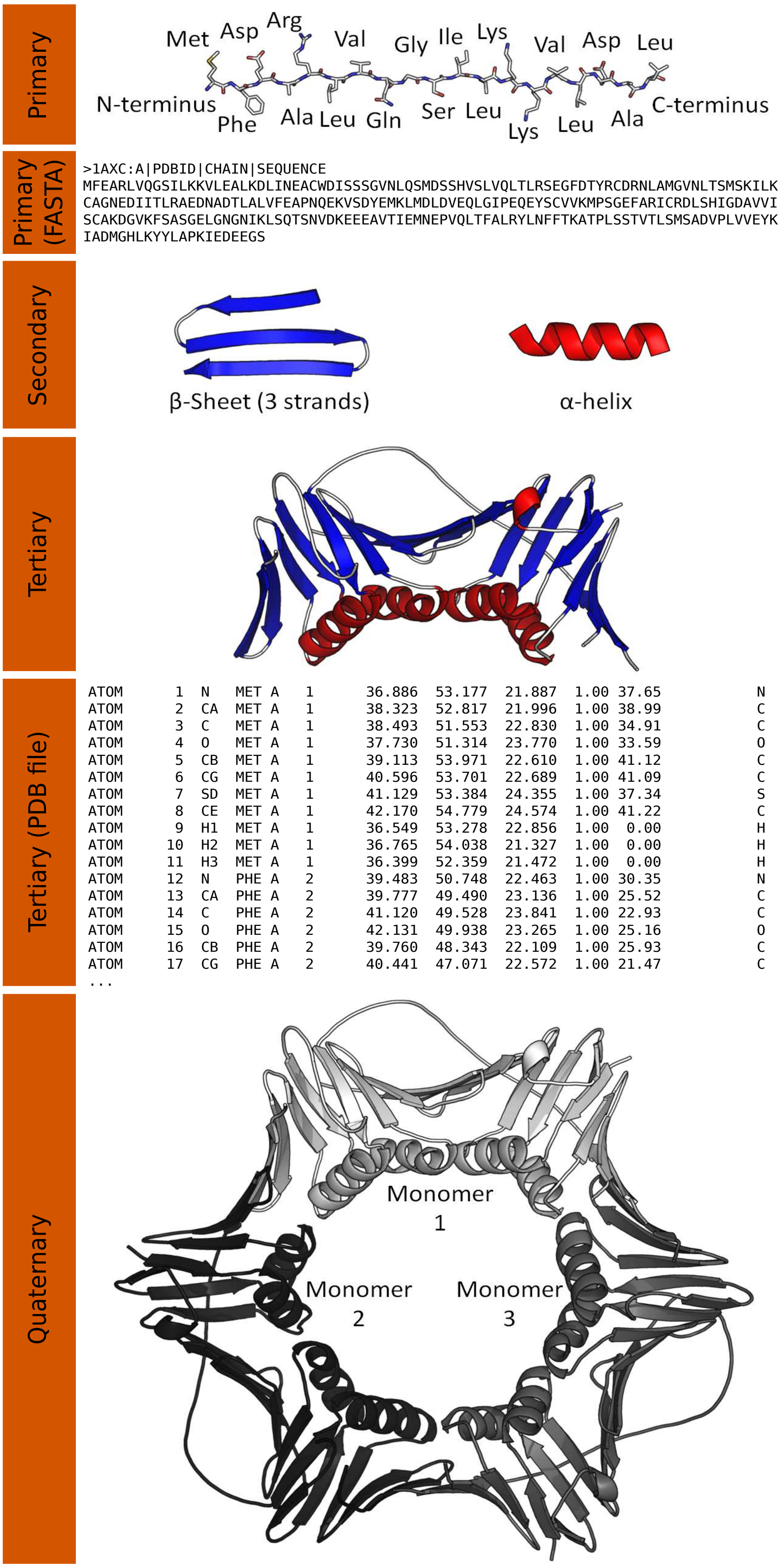
- **Primary structure** describes the sequence(s) of amino acids that form the protein. Each amino acid is assigned a letter, so a protein is represented as one or more strings of letters. Amino acids are also called *residues*.

- **Secondary structure** refers to regularly repeating local structure held together by hydrogen bonds. Secondary structure does not describe the global structure in three-dimensional space.

- **Tertiary structure** characterizes the overall shape of a single protein. Each atom is given three coordinates and together they form the whole protein. Many studies claim that protein structure plays an important role in predicting binding sites of proteins. [2, 3, 4]

- **Quaternary structure** defines the arrangement of multiple folded protein subunits in a protein complex. Complexes of two and more subunits are called *multimers*.

Proteins perform their function by binding to other molecules such as other proteins (*protein-protein interaction*), small molecules — so called ligands (*protein-ligand interactions*), DNAs or RNAs. Therefore, identification of these binding sites plays extremely important role in understanding the protein function and a mechanism of the interactions. These so-called protein-ligand interactions are of great interest because most of the currently used drugs are small molecules. More specifically, the knowledge of protein-ligand binding sites (also called *pockets*) is essential in many applications such as rational drug-design [5, 6], drug side-effects prediction [7] or elucidation of proper protein function [8]. Furthermore, protein-ligand binding site detection is a fundamental for protein-ligand docking. Protein-ligand docking is a modeling procedure that attempts to predict location and orientation of ligand when bound to protein. This process is computationally demanding because of many degrees of freedom that need to be



```
>1AXC:A|PDBID|CHAIN|SEQUENCE
MFEARLVQGSILKKVLEALKDLINEACWDISSSGVNLQSMDSSHVSLVQLTLRSEGFDTYRCDRNLAMGVNLTSMSKILK
CAGNEDIITLRAEDNADTLALVFEAPNQEKVSDYEMKLMDLDVEQLGIPEQEYSCVVKMPSGEFARICRDLSHIGDAVVI
SCAKDGVKFSASGELGNGNIKLSQTSNVDKEEEAVTIEMNEPVQLTFALRYLNFFTKATPLSSTVTLSMSADVPLVVEYK
IADMGHLKYYLAPKIEDEEGS
```

```
ATOM      1  N   MET A   1      36.886  53.177  21.887  1.00 37.65           N
ATOM      2  CA  MET A   1      38.323  52.817  21.996  1.00 38.99           C
ATOM      3  C   MET A   1      38.493  51.553  22.830  1.00 34.91           C
ATOM      4  O   MET A   1      37.730  51.314  23.770  1.00 33.59           O
ATOM      5  CB  MET A   1      39.113  53.971  22.610  1.00 41.12           C
ATOM      6  CG  MET A   1      40.596  53.701  22.689  1.00 41.09           C
ATOM      7  SD  MET A   1      41.129  53.384  24.355  1.00 37.34           S
ATOM      8  CE  MET A   1      42.170  54.779  24.574  1.00 41.22           C
ATOM      9  H1  MET A   1      36.549  53.278  22.856  1.00  0.00           H
ATOM     10  H2  MET A   1      36.765  54.038  21.327  1.00  0.00           H
ATOM     11  H3  MET A   1      36.399  52.359  21.472  1.00  0.00           H
ATOM     12  N   PHE A   2      39.483  50.748  22.463  1.00 30.35           N
ATOM     13  CA  PHE A   2      39.777  49.490  23.136  1.00 25.52           C
ATOM     14  C   PHE A   2      41.120  49.528  23.841  1.00 22.93           C
ATOM     15  O   PHE A   2      42.131  49.938  23.265  1.00 25.16           O
ATOM     16  CB  PHE A   2      39.760  48.343  22.109  1.00 25.93           C
ATOM     17  CG  PHE A   2      40.441  47.071  22.572  1.00 21.47           C
...
```

Figure 1: Illustration of protein structure and source files of protein 1AXC, source: modified image originally from wikipedia.org

considered. With the knowledge of the active sites, we can perform docking only locally and limit the number of possible combinations. Therefore, local docking is significantly less time consuming and more accurate. [9]

Protein-ligand active sites detection

Within the last 20 years, the database of protein structures [10] has grown immensely, which allowed scientists to develop many algorithms for predicting protein-ligand binding sites based on various strategies. These methods can be classified into several categories:

- **Geometrical methods** are driven by a simple idea that a ligand needs to fit to the place where it binds to. These methods use various techniques to exploit the protein's structure in order to find concave areas in the protein's surface. [11, 12, 13] Most of the tools project the protein surface onto a grid and look for most solvent accessible grid cells.

- **Energetic methods** place a probe (for example methyl) on protein surface and calculate the van der Waals energy between the protein and the probe. They predict binding sites in the energetically most favorable locus. [14, 15] Authors claim that these methods, unlike geometric methods, perform well on unbound proteins.

- **Evolutionary methods** rely on protein primary structure. Using protein sequence, we can calculate so called conservation score (homology), which correlates with protein active sites. [4, 16, 17] We describe homology in detail in Chapter 2. Evolutionary methods are usually combined with other methods.

- **Knowledge-based methods** try to extract knowledge about a protein from structural and sequential databases using statistical methods. They generalize the extracted knowledge to predict active sites on previously unseen proteins. [2, 3, 18, 19, 20]

- **Consensus methods** (also called *metapredictors*) combine results of other methods and usually achieve slightly better results. [21, 22]

P2Rank

P2Rank [3] is a knowledge-based multiplatform tool capable of predicting protein-ligand binding sites developed by Radoslav Krivák at Charles University. P2Rank utilizes a machine learning algorithm to learn protein surface features of protein-ligand binding sites on a training dataset, and then it uses this knowledge to detect the binding site on unseen data. During training, P2Rank's algorithm proceeds in the following steps:

1. *Accessible surface area* (ASA) is calculated using the rolling ball algorithm. ASA sometimes also called *Solvent accessible surface* (SAS) is the surface area that would be accessible to a solvent. [23]

2. P2Rank covers the calculated surface area of a protein with equally spaced points (we call them *SAS points*).

3. A wide range of features is extracted for each SAS point based on their local chemical neighborhood.

4. RandomForest is trained to predict the ligandibility score for each SAS point. We call the number that RandomForest outputs for each point ligandibility score. RandomForest learns to assign high ligandibility score to SAS points in binding sites.

When trained, P2Rank can detect the protein-ligand binding sites using the following procedure:

1–3. Steps 1–3 are identical to the first three steps in the training phase.

4. The trained RandomForest assigns ligandibility scores to each SAS point based on learned knowledge from the training phase.

5. SAS points are filtered and clustered in order to predict binding pockets.

6. Lastly, the predicted pockets are ranked and sorted based on the cumulative ligandibility score of SAS points forming them.

P2Rank only provides a command line interface; user specifies the path to a protein PDB file (file containing a list of atoms and the coordinates). A list of pockets and their ligandibility scores are calculated using the algorithm described above. A pocket is denoted by a list of atoms forming the binding site. P2Rank also allows the user to generate a visualization script for PyMol.

Thesis goals

Most of the protein-ligand prediction algorithms described above are implemented as web servers or freely available tools for Linux with command line interfaces. For a biologist who is not used to working in a terminal, executing prediction using a command line can present a significant challenge. In fact, authors of P2Rank were rebuked for not having graphical user interface when submitting the manuscript introducing the P2Rank tool. Another issue is results visualization. To visualize the results, a script for PyMol [24] or JMOL [25] is usually generated. Some of the prediction tools that are also implemented as web applications provide visualization using a JMOL Java Applet directly in the browser. Unfortunately, installation of PyMol as well as setting up security exceptions for JMOL is very troublesome. Neither PyMol nor JMol provide very friendly user interface. Moreover, major modern web browsers no longer support Java applets, therefore, running JMOL in certain web browsers is not possible anymore. [26] On top of that, we did not encounter a tool that is capable of visualizing not only protein tertiary structure, but also primary structure ideally with protein sequence features such as conservation scores. In fact, P2Rank does not use evolutionary information for pocket detection at all.

The goals of this work are the following:

1. Provide a modern web-based interface that allows visualization of P2Rank results as well as sequence conservation on protein's primary and tertiary structure. We named it PrankWeb.

2. Improve the P2Rank prediction algorithm by re-scoring predicted pockets using homology and by including sequence conservation in the prediction process.

We structured this work as follows: First, we describe PrankWeb design and how to use the PrankWeb application. Second, we outline what is evolutionary homology and how to calculate it. Then, we follow up with experiments for improving P2Rank prediction using homology. Finally, we evaluate the experiments and discuss their results.

1. Web application

In the introduction section, we stated that our goal is to build a web application PrankWeb, which provides a user-friendly visualization of protein-ligand active sites and conservation scores. In this chapter, we describe PrankWeb design, and at the end, we include a short user guide section explaining how to use it.

1.1 Architecture

To fulfill our goal of building a web interface for protein-ligand binding site visualization, we need three main components that communicate with each other. A user runs the web-application locally in his web-browser. We call this component the frontend. The frontend sends a request to the backend (web server running at some remote computer) and the server sends back all necessary data for protein visualization. Figure 1.1 illustrates PrankWeb's design.

1.1.1 Frontend

Users communicate with the frontend, which is a set of HTML, JavaScript and CSS files displayed in their web browser, sent to them by the web server. We based our frontend on Bootstrap.js, a popular framework for creating responsive websites. Since many sites are built with Bootstrap, people are used to Bootstrap user interface. Moreover, Boostrap makes CSS styling a lot easier. Our frontend consists of two main parts: a submit page and a visualization page, discussed later in Section 1.1.3. The submit page mediates the user request for protein visualization to the server where it is processed.

1.1.2 Server backend

The server takes care of all data requests from the frontend. PrankWeb server runs on Java EE JBoss WildFly application server. JBoss WildFly is simply a Java application that runs on the server and takes care of the HTTP communication. PrankWeb compiles into a web archive that WildFly loads and executes. We considered several other web server frameworks and assessed that Java EE suits our needs best. Here is a list of frameworks we considered and our verdict on them.

- **ASP.NET**

  Given our previous experience with ASP.NET, we wanted PrankWeb to run on the .NET platform. However, we dismissed the initial idea for several reasons. First, we want PrankWeb to be multi-platform. Second, there exists no PDB file parser implementation under .NET. Lastly, adding another programming language to the Prank project seemed superfluous. For this particular reason, we also rejected Python Django and RubyOnRails.

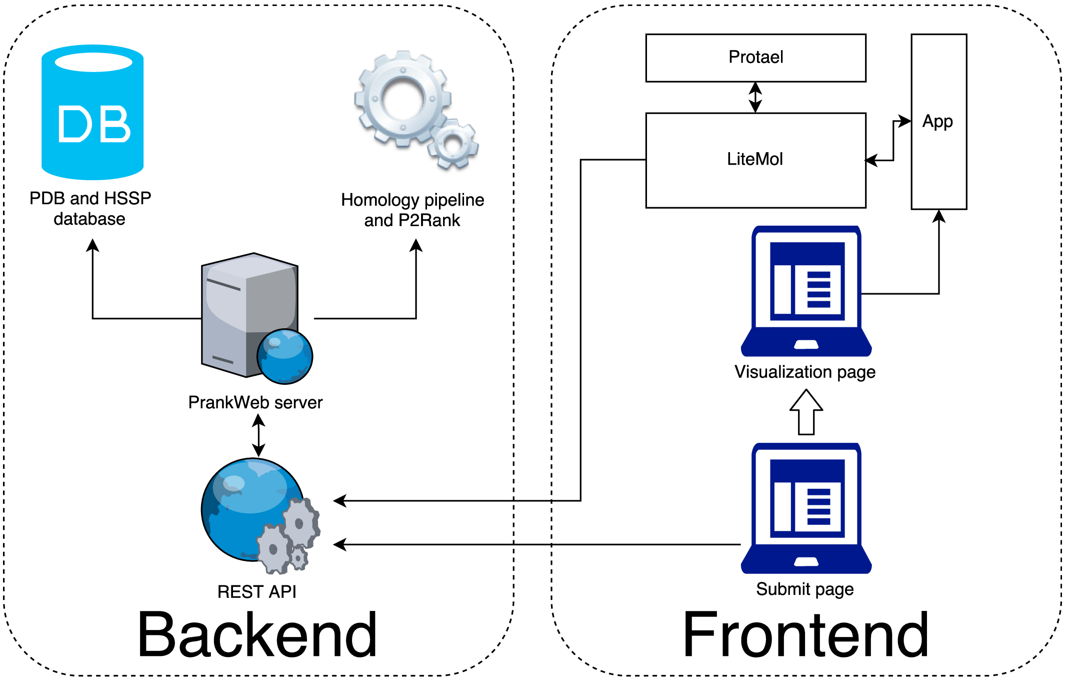


Figure 1.1: Diagram of PrankWeb architecture. Arrows illustrate calling between components.

- **NodeJs & Express**

  Given the increasing popularity of NodeJs, we built a PrankWeb prototype based on NodeJs. The main advantage was to have the frontend and backend written in the same language. Furthermore, building a web server in NodeJs using the Express framework was an extremely easy and fast process. However, as the project grew, it turned out that some preprocessing of biological data on the server side would be necessary. Since we did not find any suitable framework capable of parsing PDB files, we ported the server into Java.

- **Java & JBoss Wildfly**

  Eventually, we decided for PrankWeb to run on Java EE for the following reasons. First, P2Rank runs on the Java Virtual Machine, which allows more convenient communication between P2Rank and the server. Second, BioJava, an open-source Java library, is one of the most used tools for processing biological data. Initially, we wanted to build the server based on JBoss WildFly Swarm. WildFly Swarm allows to package all needed server components into a small standalone jar file. Unfortunately, WildFly Swarm still suffers from being in its early stages; after some struggling, we moved to the full WildFly Java Application Server and all the problems ceased.

The server handles three main tasks: it hosts the web application, it serves the precomputed data via REST API, and lastly, it executes the analysis of protein files. A manual on how to build, setup and deploy PrankWeb server is included in the attachments of this work.

Web application

Since PrankWeb is a web application, our backend has to serve HTML, JavaScript and CSS files that secures the communication with users. To avoid repetitive code, we use templating framework called JavaServer Pages (JSP). JSP files are HTML templates with Java code that compile into plain HTML files on the fly. Since all visualization is handled on the client side in JavaScript, our JSP files are almost plain static HTML files.

REST API

We store the whole PDB and HSSP database (see Section 2.1.2) locally on the server together with precomputed P2Rank prediction files and conservation scores. That way, we can handle most of the requests without high CPU server demand. We serve the stored data using *Representational State Transfer* (REST). REST provides a very convenient way of communication between the backend and the frontend. By using a REST API, we increase the modularity of the project since we can completely separate the frontend and the backend. Moreover, our REST API can be used by developers in other projects. Detailed description of the API is included in Section PrankWeb REST API in the attachments.

Analysis of proteins

Although we precomputed and stored P2Rank prediction results and conservation scores for all the proteins in PDB, PrankWeb offers visualization even for arbitrary user uploaded protein files. In such case, we run a simple data sanity check, and then execute our homology pipeline and P2Rank. Please see Chapter 2 about homology for more details about the pipeline.

P2Rank is a Java application for prediction protein-ligand binding sites of protein. PrankWeb calls P2Rank as a Java library via *PrankPredictor* API class. We described the P2Rank algorithm in the introduction chapter. We illustrate how to extend P2Rank with an additional feature in Section 3.3.

All internal classes for biological preprocessing are in a separate jar package, so that it can be used as a console application as well. Such utility comes useful, for example, when we need to convert PDB file containing protein structure into a file with protein sequence in standardized FASTA format.

FASTA is a simple text format for storing protein sequences. Each sequence in the file begins with a header line. Header line starts with the > symbol followed by an arbitrary string (usually contains details about the sequence). The following lines are the actual protein sequence, where each character represents one residue (amino acid). A sequence line should not exceed 120 characters, but usually is not longer than 80 characters. Multiple sequences can be stored in a single FASTA file; however, our conservation pipeline requires only one sequence (protein chain) on the input.

1.1.3 Visualization

When a user request is processed by the server, the user is redirected to our visualization page. Visualization is the most essential part of PrankWeb. In order to compete with JMol and PyMol, we provide the user with a graphical interface consisting of both structural and sequential visualization.

Structural visualization

To present P2Rank results, a solid protein surface visualizer based on WebGL is required. While there are a few WebGL molecular visualizers available, finding the right one presents quite a challenge. In fact, as of Summer 2016, the largest protein data bank RCSB [10] did not offer a decent JavaScript-based tool that would calculate and display protein surface even for a small protein (~ 100 residues) within several seconds. We prototyped PrankWeb with multiple visualizers and ended up with LiteMol. Here is an overview of tools we evaluated when developing PrankWeb:

- **3DMol** [27] is a tool developed at the University of Pittsburgh. While it focuses on efficient molecular visualization, it lacks any user interface controls. It presents an ideal option for embedding quick interactive snapshots of proteins in a website.

- **JSMol** [28] is JavaScript port of very popular JMol Java Applet visualizer. Nearly all biologists are used to JMol, therefore, the main advantage of JSMol lies in its user interface. Unfortunately, JSMol is extremely computationally inefficient; displaying surface of a small protein (~ 100 amino acids) freezes the web browser and even a simple rotation lags.

- **PV (Protein Viewer)** [29] is a very simple and fast visualizer. We dismissed using PV since it does not support surface visualization, which is crucial for displaying protein-ligand binding sites.

- **iCn3D** [30] is a viewer developed by the National Center for Biotechnology Information (NCBI). At the time of prototyping, iCn3D was in early stages of development. iCn3D provides a rich user interface and user scripting features. It did not fulfill our needs since features such as loading molecules from custom sources were not supported at that time. Moreover, the user interface was quite nonintuitive and the visualization was not as smooth as in other tools.

- **NGL Viewer** [31, 32] is a very fast and high quality visualizer. Currently, NGL is default molecular viewer at RCSB PDB. On the other hand, its user interface is plain and most importantly, when we started working on PrankWeb, the project did not exist yet.

- **LiteMol** [33] is a blazing fast molecular visualizer developed by David Sehnal at Masaryk University in Brno in collaboration with PDBe (Protein Data Bank in Europe). LiteMol is designed in a very general way to be capable of almost anything in terms of molecular visualization. LiteMol is written in TypeScript (extended and statically typed JavaScript) which makes such a large project more manageable and easier to extend. On the down side, LiteMol's generality makes every operation, in comparison to other tools, more complex and difficult to implement. Since LiteMol was the only tool providing a user friendly graphical interface as well as fast and smooth surface visualization, we decided to go with LiteMol.

Sequence visualization

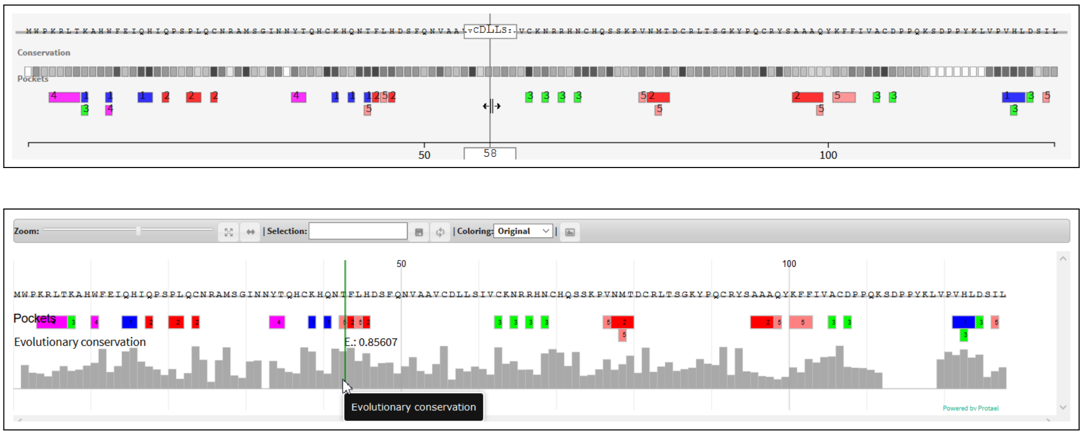
While the structural 3D model of a protein is necessary for protein-ligand active site analysis, many biologists also exploit protein primary structure (sequence). Thus, we felt the urge to offer not only the structural protein visualization, but sequence visualization as well. Moreover, to display conservation scores (see Chapter 2 about homology), we needed to display a number between 0 and 1 for each residue. Ideally, we also needed to connect both visualizations together. Particularly, as the user hovers across a residue, the residue is highlighted in the structural visualization. Again as in case of structural visualization, we considered several approaches to protein sequence visualization, and eventually, ended up with Protael. A brief summary of each approach is included below:

- **Plain HTML sequence visualizer** was a starting prototype we created to display a protein sequence and its pockets. Each residue was represented with a `span` tag, which we styled with CSS and assigned `mouseenter`, `mouseleave` and `onclick` functions. It was simple and worked flawlessly with small proteins (less than 100 residues). However, when we tried to display larger proteins, we encountered several problems. First, predicted pockets overlap. As mentioned in introduction, a residue (amino acid) consists of several atoms. If two sets of atoms, which are both part of the same amino acid, are detected as possible active sites, then we call those two pockets *overlapping*. We wanted to depict overlapping pockets; however, coloring a single letter with multiple colors was obscure. Second, as the protein sequence became longer, the user could not see the whole sequence at once. On top of that, we also wanted to incorporate conservation score into the visualization, which would mean aligning several colored lines with each other.

- **pViz** [34] is a sequence visualizer that generates SVG (Scalable vector graphics) and allows to set mouse event callback functions. Furthermore, pViz handles sequence zooming, pocket overlapping as well as displaying several features, which was the main issue with plain HTML visualizer. Sadly, it also comes with several drawbacks: pViz uses several external libraries that clash with Bootstrap.js (user interface framework) that we use to display PrankWeb menus.

- **Protael** [35] is another JavaScript library for protein sequence visualization based on generating SVG. Not only it is capable of the same functionality as pViz, but it also offers so called quantitative features.[1] Using quantitative features, we can display evolutionary conservation as a bar chart instead of a single stripe colored with multiple shades of gray. Protael suffers from similar caveats like pViz: it is based on several external libraries (jQueryUI, SnapSVG) that can possibly clash with the rest of the website. Fortunately, we did not encounter such an issue.

[1] Protael provides far more features; however, we do not find them useful for our project.



Conservation
Pockets
50
58
100
Zoom:
| Selection:
| Coloring: Original
50
100
MWPKRLTKAHWFEIQHIQPSPLQCNRAMSGINNYTQHCKHQNTFLHDSFQNVAAVCDLLSIVCKNRRHNCHQSSKPVNMTDCRLTSGKYPQCRYSAAAQYKFFIVACDPPQKSDPPYKLVPVHLDSIL
Pockets
Evolutionary conservation
E.: 0.85607
Evolutionary conservation
Powered by Protael

Figure 1.2: Comparison of pViz (top) and Protael (bottom) sequence visualizers.

After settling with the right tools, we designed the visualization website as follows. We utilized general design of LiteMol and transformed it into PrankWeb visualization page. We replaced the original Log component with a new Sequence View component and added our control panel next to LiteMol visualization window.

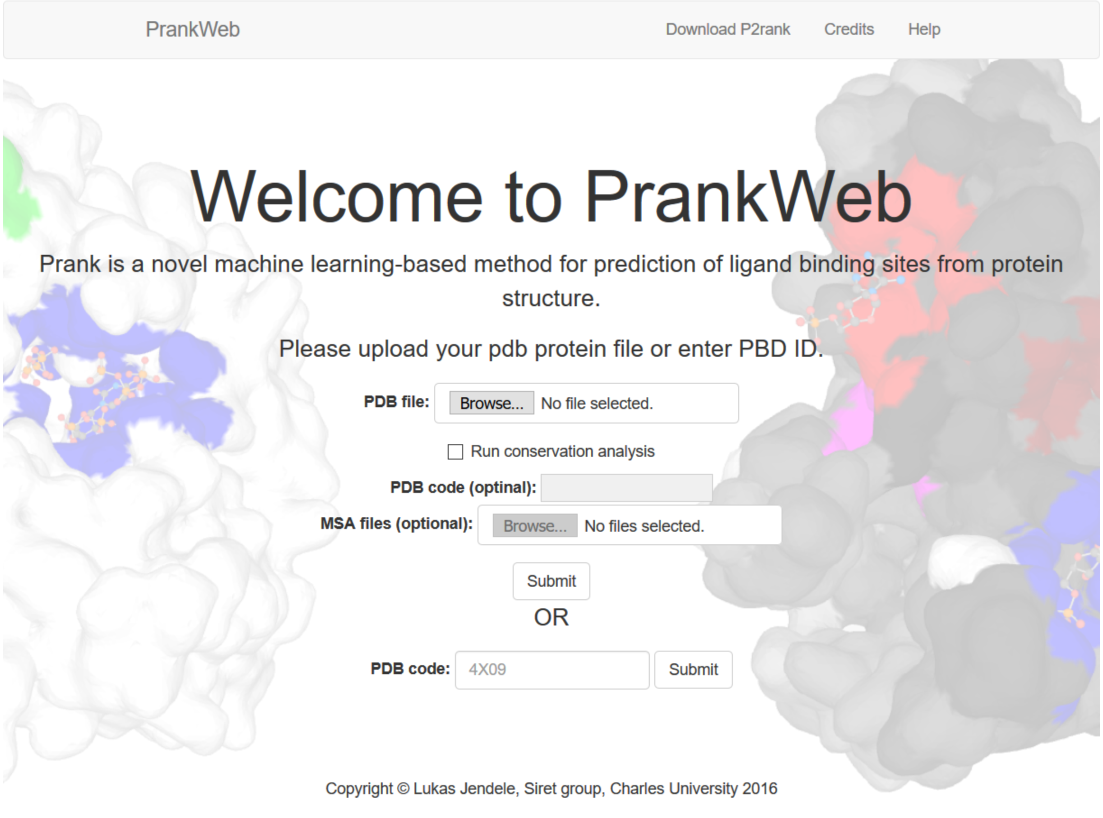
LiteMol keeps all information as entities in a tree. The source data (data entities) are represented as children nodes of the root. The children nodes of these data entities describe so called transformations such as data parsing, surface calculation, visualization and so on. All changes to the tree are broadcast via events. Any LiteMol component can subscribe to such event. Our frontend contains three main components that communicate with each other: side panel, 3D structural visualization and sequence visualization. Everything is based on React, so all views update automatically when their state changes.

We create root React component (called App) that initializes LiteMol and mounts to the right side panel. During initialization, LiteMol downloads all necessary data from backend via REST API and stores them into its tree structure. After successful loading, our App component displays a pocket list and controls LiteMol by calling commands and listening to subscribed events.

When LiteMol initializes, it creates a sequence data entity and sequence view component that resides on top of 3D visualization. The sequence view component consists of sequence view and sequence controller. While the controller takes care of storing data and communicating with other parts of LiteMol, the sequence view calculates the feature segments and renders Protael into a given region.

1.2 User guide

PrankWeb is a web-based application that allows to predict and visualize protein-ligand binding sites. Furthermore, it allows to compare the location of predicted pockets with highly conserved areas as well as actual ligand binding sites. All one needs to use PrankWeb is a device with a web-browser that supports WebGL.



Welcome to PrankWeb

Prank is a novel machine learning-based method for prediction of ligand binding sites from protein structure.

Please upload your pdb protein file or enter PBD ID.

PDB file: No file selected.

□ Run conservation analysis

PDB code (optinal):

MSA files (optional): No files selected.

OR

PDB code: 4X09

Copyright © Lukas Jendele, Siret group, Charles University 2016

Figure 1.3: Screenshot of PrankWeb home page.

1.2.1 Specify what protein to analyze

To analyze a protein from PDB database, one has to know its identification code. Go to a website, where PrankWeb server is running[2] and enter the identification code of the protein. For analysis of a custom PDB file, upload it by clicking the browse button (see Figure 1.3). Please note that it might take a while before PrankWeb analyzes the custom file. For description of the PDB format, please see its official documentation.[3]

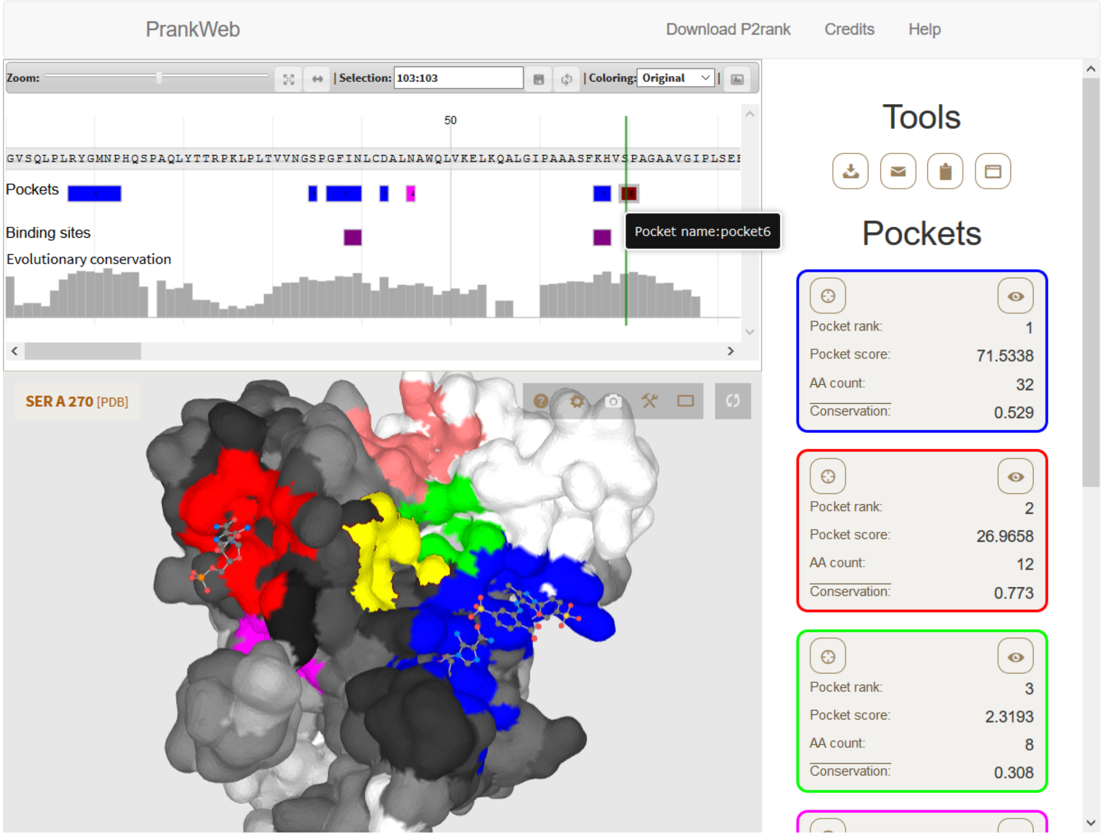
Besides selecting what protein to analyze, one can also specify if evolutionary conservation should be included in the prediction model. PrankWeb contains two pretrained models for pocket prediction. Note that calculating conservation score for user-defined protein file can significantly increase the time of analysis unless you specify its PDB identification code or upload multiple sequence alignments for homology calculation. Speed comparison of conservation scores calculation is included in Section 4.1.

1.2.2 Visualization

Once the protein visualization is loaded, three main panels appear: sequence visualization, structural visualization and the control panel (see Figure 1.4).

[2] For example: `http://prank.projekty.ms.mff.cuni.cz`.

[3] PDB Format specification available at: `http://www.wwpdb.org/documentation/file-format`.



Selection: 103:103

Coloring: Original

GVSQLPLRYGMNPHQSPAQLYTTRPKLPLTVVNGSPGFINLCDALNAWQLVKELKQALGIPAAASFKHVSPAGAAVGIPLSE

Pocket name:pocket6

Pocket rank: 1
Pocket score: 71.5338
AA count: 32
Conservation: 0.529

Pocket rank: 2
Pocket score: 26.9658
AA count: 12
Conservation: 0.773

Pocket rank: 3
Pocket score: 2.3193
AA count: 8
Conservation: 0.308

Figure 1.4: Screenshot of PrankWeb visualization page.

Structural visualization

The largest panel contains the three-dimensional visualization of the protein (see Figure 1.4). By default, the protein surface is displayed, and individual pocket areas are highlighted with different colors. If conservation is available, the protein surface is colored with 11 shades of gray according to the conservation score of each residue. Darker color depicts higher conservation score. To display the protein in cartoon style, the protein surface can be hidden completely in the advanced control panel (explained below), or one can slab the protein view by scrolling the mouse wheel.

Controls The molecule can be rotated by moving mouse while holding left mouse button. On a touch device, just slide your finger. To zoom in or out, move your mouse while holding the right mouse button or use the pinch gesture on a touch display. In order to move the protein, do the same, but this time hold the wheel button. Lastly, for slabbing the protein, scroll the mouse wheel or use the three finger gesture.

Using the buttons in the top-right corner, one can:

- Display a help window.
- Setup the scene such as the visualization background or the field of view.
- Create a snapshot of current visualization.
- Toggle advanced control panels.

- Toggle full-screen mode.

By toggling the advanced control panel, one has full control over the content of the visualization. First, select what part of visualization to edit in the tree control on the left, then edit its properties or create new nodes using the right panel. For example, to increase the surface probe radius, click on the surface section on the left (subsection of polymer). Now, in the update visual section, expand the visual type option and use the probe radius trackbar to update the value. Please be cautious, because as of May 2017, LiteMol does not support Undo operation. It will be added soon, though. For more help with LiteMol, please visit its wiki page.[4]

Sequence visualization

The panel above protein 3D visualization displays protein sequence. All chains are concatenated and visualized at once. Chains can be identified by regions marked on the main sequence. Colored rectangles depict areas with predicted pockets and real binding areas (if available). Real binding sites are residues within 4 Å from any ligand atom. If available, conservation scores are portrayed using a bar chart. As one hovers over the sequence with mouse, the residues are highlighted in the 3D visualization. This feature allows to analyze the protein both from the structural and sequential point of view. By default, the sequence view is zoomed out so that the whole protein is displayed. You can use the trackbar control to zoom in, or select the area with mouse and zoom to the selection. A snapshot of the sequence can be captured and exported to SVG (Scalable Vector Graphics) file using the rightmost button.

1.2.3 Pocket panel

The right panel contains several control buttons and a list of predicted pockets. Use the control buttons to download PrankWeb report, share the website or hide the sequence view. PrankWeb report is a ZIP package containing all following files:

- Original pdb file uploaded, or the protein file downloaded from RCSB PDB.
- Prediction JSON file containing a list of pockets, their scores and their location i.e. a list of atoms and residues forming the pocket (output of P2Rank).
- PyMol script for offline visualization (output of P2Rank).
- Conservation scores for each residue of the protein calculated using JSD method (see Section 2.2 for more details). The file is in TSV (Tab Separated Values) format. First column identifies the index, second contains the score and the last column contains the residues from multiple sequence alignment. The first characters in the last column are the residues of the analyzed protein file.

[4] `https://webchem.ncbr.muni.cz/Wiki/LiteMol:UserManual`

- Multiple sequence alignments in FASTA format that were used to calculate conservation score.

In the pocket list, pocket name, rank, size and average conservation score (if available) is displayed for each pocket. Moreover, one can highlight the pocket in the 3D visualization by hovering over the ⊕ button. After clicking that button, the camera will zoom in to the pocket. By clicking the 👁 button, one can toggle the pocket visibility in both structural and sequence visualizations.

2. Homology

As mentioned in the introduction, DNA sequences define protein sequence. Therefore, mutation of DNA sequence consequently leads to a change in protein sequence. When we look at the same protein sequence across several species, we can align the sequences, and notice that several positions have not changed. The same observation holds also for similar sequences which evolved from a common origin. Obviously, the more evolutionary distant the protein are, the more divergence we can observe on the sequence level. Due to various reasons, some of the positions tend to vary across the proteins and some tend to stay unchanged (we call them *conserved*). The amount of variation is captured by a number which is called *conservation score*, a real number between 0 (not conserved) and 1 (highly conserved) for each residue of the protein. Figure 2.1 illustrates example of protein sequence alignment, also called *multiple sequence alignment* (MSA), and conserved residues.

According to many studies, evolutionary conservation correlates with protein binding sites. [4, 16, 17] If a sequence mutation (insertion, deletion or change of an amino acid) damages an active site, the organism need to adapt to the sudden change.

2.1 Databases

Before we delve into calculating conservation score, we need to briefly introduce databases that we use when calculating homology.

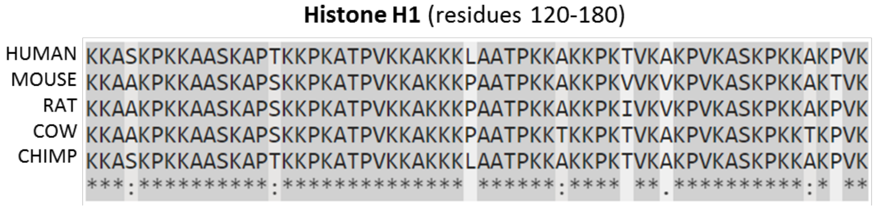
2.1.1 UniProt

SwissProt, **TrEMBL** and **UniRef** are freely accessible databases of protein sequences and annotation data.[1] They are part of UniProt (Universal Protein Resource), which is managed by consortium of several bioinformatics institutes. [36]

SwissProt is a manually annotated, non-redundant protein database consisting of over half a million sequences. It has been extracted both from scientific literature as well as computational analysis.

TrEMBL is significantly larger than SwissProt. It contains all sequences awaiting manual annotation. All sequences in TrEMBL are automatically annotated.

[1] All UniProt databases can be downloaded at `http://www.uniprot.org/downloads`



Histone H1 (residues 120-180)

```
HUMAN KKASKPKKAASKAPTKKPKATPVKKAKKKLAATPKKAKKPKTVKAKPVKASPKKAKPVK
MOUSE KKAAKPKKAASKAPSKKPKATPVKKAKKKPAATPKKAKKPKVVKVKPVKASPKKAKTVK
  RAT KKAAKPKKAASKAPSKKPKATPVKKAKKKPAATPKKAKKPKIVKVKPVKASPKKAKPVK
  COW KKAAKPKKAASKAPSKKPKATPVKKAKKKPAATPKKTKKPKTVKAKPVKASPKKTKPVK
CHIMP KKASKPKKAASKAPTKKPKATPVKKAKKKLAATPKKAKKPKTVKAKPVKASPKKAKPVK
      ***:**********:************** ******:**** **.**********:* **
```

Figure 2.1: Example of protein sequence alignment and conservation. The darkest columns depict conserved residues. Source: wikipedia.org

Database	Number of sequences
SwissProt	554 241
TrEMBL	84 827 567
UniRef100	106 078 676
UniRef90	55 192 141
UniRef50	21 859 863

Table 2.1: Number of sequences in UniProt databases at the time of writing

UniRef [37] database provides clustered sets of sequences from all UniProt databases. The clustering hides redundant sequences and obtains complete coverage of the sequence space at three resolutions:

- **UniRef100** combines identical sequences and sub-fragments with 11 or more residues from any organism into a single UniRef entry.
- **UniRef90** is built by clustering UniRef100 entries so that each cluster is composed of sequences that have at least 90% sequence identity to, and 80% overlap with, the longest sequence (a.k.a. seed sequence).
- **UniRef50** is built the same way as UniRef90, but only with threshold 50% for sequence identity.

See Figure 2.2 for an example of two sequences from a UniProt database. Table 2.1 displays number of sequences in each database at the time of writing.

2.1.2 HSSP

HSSP (Homology-derived Secondary Structure of Proteins) [38] is a database merging structural and sequence information.[2] For each protein with structural information in Protein Data Bank, it contains a multiple sequence alignment of all available homologues (similar proteins). The homologues have been extracted from SwissProt database (described in Section 2.1.1). The database is updated frequently. MSAs are stored in a special format and we use a Perl script by Yossi Rosenberg and Fabian Glaser to convert it to standardized FASTA format. The script is attached to this thesis.

2.2 Calculating conservation score

While precomputed databases with conservation scores and MSAs are a great source of information, they do not always contain the information for an arbitrary user uploaded protein. Thus, we need to be able to calculate conservation scores ourselves. As explained at the beginning of the chapter, for each residue (amino acid) of each chain in a protein, we need to calculate a real number between 0 and 1, which we call conservation score. Conservation scores are usually calculated from multiple sequence alignment (MSA) of similar species. MSA contains a set of sequences that are aligned so that each column consists of the same residues.

[2] HSSP database is freely available at: `http://swift.cmbi.ru.nl/gv/hssp/`

ConSurfDB [39] is precomputed database of conservation scores of nearly 81 thousand proteins. Inspired by ConSurfDB, we built a pipeline of bioinformatics tools that calculates conservation scores for arbitrary protein sequence.[3]

2.2.1 Acquiring multiple sequence alignment

MSA for a particular sequence can be acquired from a HSSP database (see Section 2.1.2) or calculated from a set of sequences using bioinformatics tools such as MUSCLE. [40] Moreover, PrankWeb also allows the user to upload own MSA for each chain, using which we calculate the conservation scores.

If HSSP database contains the protein and we cannot identify the particular chain (no chain for particular ID was found), we take the chain with the longest common subsequence (see Section 2.2.4 for more details). In case the protein is not present in the HSSP database and the user did not provide us with the MSA for that protein, we need to calculate the MSA ourselves using homology pipeline. The main idea of the pipeline is based on querying several databases for sequences similar to the sequence on input. The pipeline is described in detail in Section 2.2.3. We illustrate the decision making process for calculating conservation scores in Figure 2.4.

2.2.2 Calculating homology from MSA

Once we have a multiple sequence alignment, we can calculate the conservation scores for each residue. There exist multiple methods for calculating conservation score from MSA. We decided to use method based on *Jensen-Shannon divergence* since it proved to be effective in finding protein functional sites and is significantly less computationally demanding than other methods such as Rate4Site (a method used in ConSurfDB). [4, 16] Moreover, the combination of HSSP database and the JSD method has already proven to be successful in ConCavity. [4]

Jensen-Shannon divergence (JSD) measures the similarity between two probability distributions. JSD is based on the famous *Kullback-Leibler divergence* also known as relative entropy or information gain. Unlike other divergence measures, JSD is symmetric and always finite. When we calculate conservation score from MSA, we need to calculate a score for each column. The idea behind JSD method is to compare the distribution of residues in the column with an appropriate background nonconserved distribution. The distribution of conserved column differs significantly from the background distribution.

2.2.3 Conservation pipeline

With the help of Marian Novotný, a biologist at Charles University with a research interest in protein homology, and inspired by ConSurfDB [39], we developed a Bash script running several bioinformatics tools calculating evolutionary conservation for any protein chain. It takes a protein sequence in FASTA format as in input and outputs a tab separated file with conservation scores, which is

[3] We tried to download, build and run our own instance of ConSurfDB pipeline, and compare its results with our pipeline; unfortunately, we did not manage to get ConSurfDB to work.

```
>sp|Q6GZX4|001R_FRG3G Putative transcription factor 001R OS=Frog...
MAFSAEDVLKEYDRRRRMEALLLSLYYPNDRKLLDYKEWSPPRVQVECPKAPVEWNNPPS
EKGLIVGHFSGIKYKGEKAQASEVDVNKMCCWVSKFKDAMRRYQGIQTCKIPGKVLSDLD
...
>sp|Q6GZX3|002L_FRG3G Uncharacterized protein 002L OS=Frog virus 3...
MSIIGATRLQNDKSDTYSAGPCYAGGCSAFTPRGTCGKDWDLGEQTCASGFCTSQPLCAR
IKKTQVCGLRYSSKGKDPLVSAEWDSRGAPYVRCTYDADLIDTQAQVDQFVSMFGESPSL
...
```

Figure 2.2: Example of first two protein sequences from SwissProt database in FASTA format.

the result of Jensen-Shannon Divergence method for calculating the conservation score from multiple sequence alignment. [16] The pipeline proceeds as follows:

1. We query SwissProt for similar protein sequences using PSI-BLAST [41] with e-value=10^{-5}. This way we specify that we want to output only very significant matches. ConSurfDB uses the same value. The output of this step is a subset of protein sequences from SwissProt and their properties.

2. We then filter out sequences that are too similar or too different than our query sequence.

3. Then CD-HIT [42] is run with default parameters to cluster the sequences and receive a non-redundant representative sequence list.

4. If less than 50 sequences are left, we repeat the steps 1–3 on a larger database UniRef90 or TrEMBL depending on the settings of the pipeline.

5. Sequences are aligned using MUSCLE [40], so that the same residues are underneath each other. This is done by inserting so called *gap* symbols (usually represented using a dash symbol) into the sequence strings (Figure 2.3).

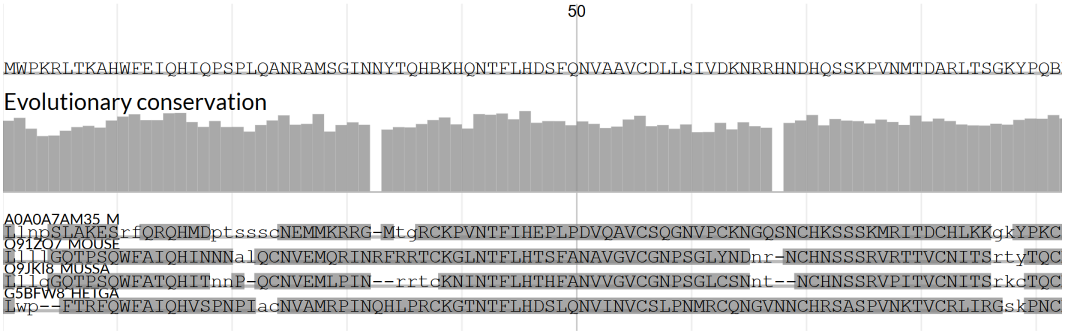
6. At this point, we have a multiple sequence alignment and can calculate conservation score using the Jensen-Shannon divergence method [16].

The pipeline process also depicted in Figure 2.4. For instructions on how to setup the pipeline please follow the pipeline setup manual in the attachments.

2.2.4 Matching sequence conservation with protein structure

Protein structural data are usually obtained using X-ray crystallography, NMR spectroscopy, or cryo-electron microscopy. On the other hand, protein sequencing is done using mass spectrometry or Edman degradation, which is much easier process, and therefore, protein sequence databases contain somewhat more entries than structural protein data banks. Since we rely on information from several databases, the sequence extracted from the structural PDB file does not

```
>4x09A
MWPKRLTKAHWFEIQHIQPSPLQaNRAMSGINNYTQHbKHQNTFLHDSFQNVAAVcDLLSIVdKNRRHN
dHQSSKPVNMTDaRLTSGKYPQbRYSAAAQYKFFIVAcDPPQKSDPPYKLVPVHLDSIL
>A0A0A7AM35_M
LlnpSLAKESrfQRQHMDptssscNEMMKRRG-MtgRCKPVNTFIHEPLPDVQAVCSQGNVPCKNGQSN
CHKSSSKMRITDCHLKKgkYPKCDYETKQLQKSIIMACECVQY-------VPVHFDGSV
>Q91ZQ7_MOUSE
LlllGQTPSQWFAIQHINNNalQCNVEMQRINRFRRTCKGLNTFLHTSFANAVGVCGNPSGLYNDnr-N
CHNSSSRVRTTVCNITSrtyTQCRYQPRRSLEYYTVACNPRTPQDSpyPVVPVHLDGTF
>Q9JKI8_MUSSA
LllqGQTPSQWFATQHITnnP-QCNVEMLPIN--rrtcKNINTFLHTHFANVVGVCGNPSGLCSNnt--
NCHNSSRVPITVCNITSrkcTQCRYQTTRSMEYYKVACNPRTPQDSsyPVVPVHLDGTF
>G5BFW8_HETGA
Lwp--FTRFQWFAIQHVSPNPIacNVAMRPINQHLPRCKGTNTFLHDSLQNVINVCSLPNMRCQNGVNN
CHRSASPVNKTVCRLIRGskPNCRYTARAMVANFVVACSPPRAGDPP-dlVPVHLD---
```



```
MWPKRLTKAHWFEIQHIQPSPLQANRAMSGINNYTQHBKHQNTFLHDSFQNVAAVCDLLSIVDKNRRHNDHQSSKPVNMTDARLTSGKYPQB
A0A0A7AM35_M
LlnpSLAKESrfQRQHMDptssscNEMMKRRG-MtgRCKPVNTFIHEPLPDVQAVCSQGNVPCKNGQSNCHKSSSKMRITDCHLKKgkYPKC
Q91ZQ7_MOUSE
LlllGQTPSQWFAIQHINNNalQCNVEMQRINRFRRTCKGLNTFLHTSFANAVGVCGNPSGLYNDnr-NCHNSSSRVRTTVCNITSrtyTQC
Q9JKI8_MUSSA
LllqGQTPSQWFATQHITnnP-QCNVEMLPIN--rrtcKNINTFLHTHFANVVGVCGNPSGLCSNnt--NCHNSSRVPITVCNITSrkcTQC
G5BFW8_HETGA
Lwp--FTRFQWFAIQHVSPNPIacNVAMRPINQHLPRCKGTNTFLHDSLQNVINVCSLPNMRCQNGVNNCHRSASPVNKTVCRLIRGskPNC
```

Figure 2.3: Example of multiple sequence alignment for protein 4X09 (chain A) in FASTA format and its visualization using Protel. Please note that the conservation scores are so high because 5 sequences is too few.

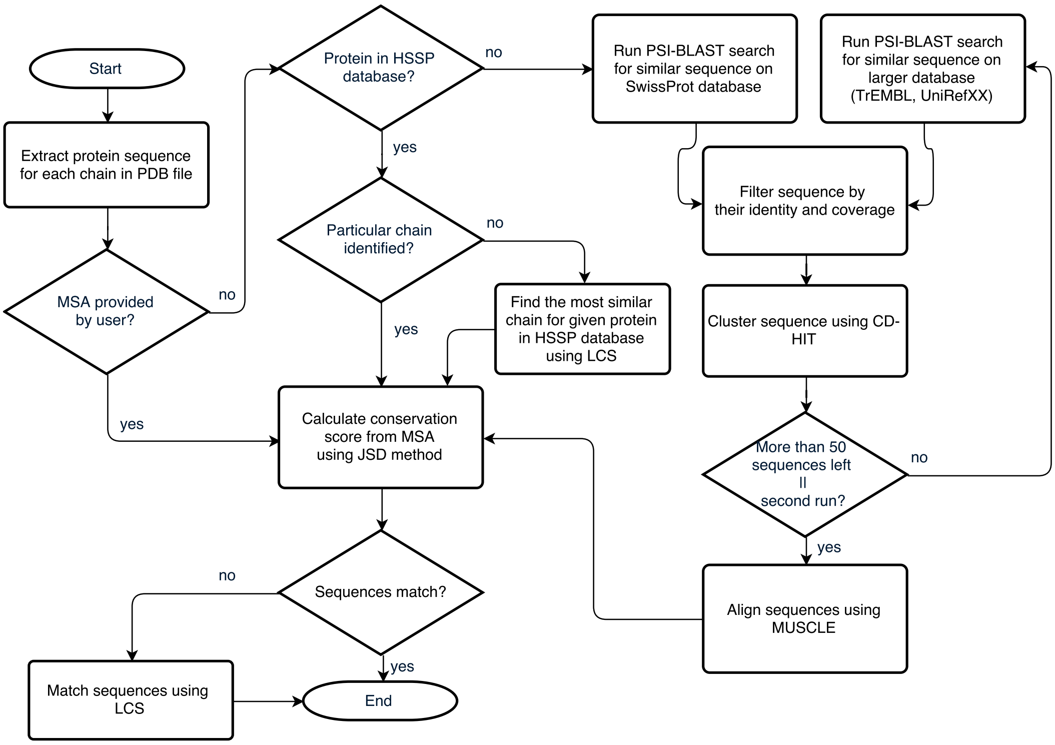

Start
Extract protein sequence for each chain in PDB file
MSA provided by user?
no
yes
Protein in HSSP database?
no
yes
Particular chain identified?
no
yes
Find the most similar chain for given protein in HSSP database using LCS
Run PSI-BLAST search for similar sequence on SwissProt database
Run PSI-BLAST search for similar sequence on larger database (TrEMBL, UniRefXX)
Filter sequence by their identity and coverage
Cluster sequence using CD-HIT
More than 50 sequences left || second run?
no
yes
Align sequences using MUSCLE
Calculate conservation score from MSA using JSD method
Sequences match?
no
yes
Match sequences using LCS
End

Figure 2.4: Diagram of conservation loading workflow and conservation pipeline.

always match with the homology annotated sequence. Usually, the annotated sequence contains some additional prefix or suffix. Unfortunately, sometimes the strings differ a bit more possibly due to measuring errors. To overcome the issue of misaligned sequences, we calculate the *longest common subsequence* using *dynamic programming*. The longest common subsequence can be determined using the following formula: Let denote the two sequences $x = x_1x_2x_3 \ldots x_m$ and $y = y_1y_2y_3 \ldots y_n$. Let denote the prefixes of sequences x: $X_1, X_2, X_3, \ldots, X_m$ and y: $Y_1, Y_2, Y_3, \ldots, Y_n$. We calculate the longest common subsequence (LCS) using this recursive formula:

$$LCS(X_i, X_j) = \begin{cases} \emptyset, & \text{if } i = 0 \vee j = 0, \\ LCS(X_{i-1}, Y_{j-1}) \oplus x_j, & \text{if } x_i = y_j, \\ lng(LCS(X_{i-1}, Yj), LCS(X_i, Y_{j-1})), & \text{otherwise,} \end{cases}$$

where $\oplus$ is string concatenation and the function lng returns the longer sequence. Time and space complexity of this algorithm is $O(mn)$, where n and m are the lengths of the sequences (usually hundreds of characters). Unfortunately, we need the longest common subsequence, not only its length, thus, the memory consumption cannot be reduced. In comparison with the conservation calculation or binding site prediction, this adds negligible overhead both in terms of time and memory usage.

3. Experiments

In the previous chapter, we explained what homology is and how to calculate conservation scores. According to many studies, functional protein sites correlate to highly conserved sites. [4, 16] We verify this statement, and rescore (reorder) P2Rank predicted pockets using calculated conservation score. Finally we implement conservation score into P2Rank as a native feature.

3.1 Correlation of conservation score and binding sites

Before we begin improving protein-ligand detection using homology, we must verify whether protein binding sites have higher conservation scores. This hypothesis has already been confirmed in many studies. [4, 16, 43] By verifying the hypothesis ourselves, we check that our calculation of homology is correct. Therefore, we calculate these following three statistics for each protein that we later average over the whole dataset in evaluation Section 4.2:

- Average protein conservation score

  We simply take the average score of all protein residues.

- Average conservation score of binding sites

  We take all protein atoms that are within 4 Å from any ligand atom. Then we average the conservation scores for parent residues (amino acids) of those atoms.

- Average conservation score of non-binding sites

  We take the complement residues for the binding residues and average their conservation scores.

3.2 Pocket rescoring

After we verify that highly conserved sites correlate with protein binding sites, we want to use this information to improve P2Rank prediction results. Most tools that detect protein functional sites output many possible pockets and only a few of them are true binding sites. In fact, the total coverage (% of identified pockets) of pocket detection tools is very high. For example, for all tested datasets FPocket achieved pocket coverage over 70%. [2] Thus, it is important not only to predict the possible binding sites, but also to rank them according to the probability of being a true pocket. Prank [2] (predecessor of P2Rank) is a knowledge-based algorithm that takes predicted pockets as input and rescores them. We adopt the idea of pocket rescoring and enhance P2Rank results by rescoring the pockets using their conservation score.

We call a predicted pocket *true* if the center of the pocket resides within 4 Å from any ligand atom. Every pocket that does not meet this condition is called a *false pocket*. We define the conservation score of a pocket as the average of conservation scores of all residues forming the pocket.

We compare the conservation scores of true and false pockets. Furthermore, we rescore (reorder) the predicted pockets according to

1. Original P2Rank ligandibility score
2. Conservation score
3. Combination of the two previous scores. Since conservation score is a number between 0 and 1, it seems natural to multiply the scores together.

3.3 Including homology as a native P2Rank feature

So far, we described how to run P2Rank to predict possible binding sites and used calculated conservation scores to rescored them. It makes sense to include conservation score as a native P2Rank feature.

As we already mentioned, P2Rank calculates various features for every SAS point (a point at the protein solvent accessible surface). Based on these features, a Random Forest is trained to predict so called ligandibility score.

P2Rank offers a nice extensible interface for implementing new features. All one needs to do, is to extend the *AtomFeatureCalculator* if you want to add an atom feature or *SasFeatureCalculator* if you can calculate your feature directly for a SAS point. They both provide a function for any protein preprocessing (for example for loading conservation score) and a function for calculating the feature value for a SAS point or a solvent accessible atom. Conservation score is a real number for each residue, thus we have to somehow assign these scores to the SAS points on protein surface. We used both *AtomFeatureCalculator* and *SasFeatureCalculator* for implementing conservation feature using the following mapping techniques:

- For each solvent accessible protein atom, we say that its conservation score is the conservation score of its parent residue i.e. the amino acid that the atom is part of. Formally:

  $$\forall a \in \{\text{solvent accessible atoms of } P\} : cscore(a) = cscore(parent(a)),$$

  where P is a protein, a is an atom and $parent(a)$ is the parent residue of atom a. Then we use P2Rank default solvent accessible atom to SAS point feature mapping. Basically, all atom features in the neighborhood of a SAS point are aggregated (we refer to the original paper for specific details [3, Equations (1),(2)]) We will call this technique simply *conservation* (C).

- Let A_d be all the protein atoms within distance d from a SAS point c. Now $\forall c \in P$, we calculate its conservation score as

  $$cscore(c) = \sum_{a \in A_d} cscore(a).$$

We will call this mapping technique *conservation cloud* (CC). Note that by summing the conservation scores, we basically calculate weighted protrusion. Protrusion exploits the protein geometrical structure and is calculated as

$$\forall c \in P : protrusion(c) = |A_d|.$$

Thanks to Random Forests, we can calculate feature importance, and it turns out that protrusion is the most important feature in the original P2Rank model. In the next chapter, we examine the feature importances after including the conservation feature.

- Let A_d be all protein atoms within distance d from a SAS point c. We calculate the conservation score the same way as in conservation cloud, but this time, we weight the score by reciprocal distance between the SAS point and the atom. Formally:

  $$\forall c \in P : cscore(c) = \sum_{a \in A_d} \frac{1}{dist(a, c)} cscore(a),$$

  where $dist(a, b)$ is the Euclidean distance between the points a and b. We call this mapping method *conservation cloud scaled* (CCS)

4. Experimental results

In this chapter, we will evaluate the experiments described in the previous chapter to improve P2Rank prediction using homology. We will also attempt to interpret the reasoning for the results.

4.1 Datasets

For our experiments, we adopted the datasets from other studies pursuing protein pocket detection. This way, our results can be directly compared with the previous studies. The model can always be both trained and evaluated on other data. See Table 4.1 for quantitative information about the datasets we used.

- CHEN11

  This dataset was designed for a comparative study [44] of protein-ligand binding site predictors. Despite its size (251 proteins), is supposed to cover wide range of non-homologous proteins. A protein representative chain has been selected for each SCOP (Structural Classification of Protein [45]) protein family (a group of evolutionary related proteins). Positions of binding sites of similar chains has also been transfered on this representative chain. Default P2Rank prediction model is trained on this dataset.

- JOINED

  Small datasets from several studies [21, 22, 43, 46] have been joined together to create this dataset. JOINED dataset served as development test data for P2Rank. That is, P2Rank hyperparameters were optimized so that P2Rank achieved highest results on JOINED dataset.

- HOLO4K

  Large dataset of protein-ligand complexes downloaded from PDB used in large scale comparative study. [47] P2Rank paper used this dataset for final evaluation. HOLO4K contains on average two ligands, that is pockets, per protein and protein structures are larger than in the other datasets (see Table 4.1).

Since we calculate the conservation score from multiple sources, we want to compare its properties among the multiple origins. By default, we calculate the homology from HSSP database if possible, otherwise, we run a pipeline that calculates the MSA and conservation score from SwissProt and TrEMBL/UniRef

Dataset	$\#proteins$	$\#ligands$	$\overline{\#ligands}$	$\overline{\#atoms_{lig}}$	$\overline{\#atoms_{prot}}$
CHEN11	251	476	1.90	25.9	1844.3
JOINED	584	680	1.16	22.3	2392.3
HOLO4K	3905	7836	2.01	26.9	3916.9

Table 4.1: Quantitative details about datasets.

Dataset	$\overline{\text{protein } cs}$	$\overline{\text{binding sites } cs}$	$\overline{\text{non-binding sites } cs}$
CHEN11$_{HSSP}$	0.503	0.621	0.503
JOINED$_{HSSP}$	0.516	0.573	0.516
HOLO4K$_{HSSP}$	0.515	0.635	0.515
CHEN11$_{SP/T}$	0.575	0.675	0.575
JOINED$_{SP/T}$	0.516	0.573	0.516
HOLO4K$_{SP/T}$	0.585	0.685	0.585
CHEN11$_{SP/U90}$	0.515	0.620	0.515
JOINED$_{SP/U90}$	0.516	0.573	0.516
HOLO4K$_{SP/U90}$	0.545	0.659	0.544

Table 4.2: Average conservation scores of binding and non-binding sites.

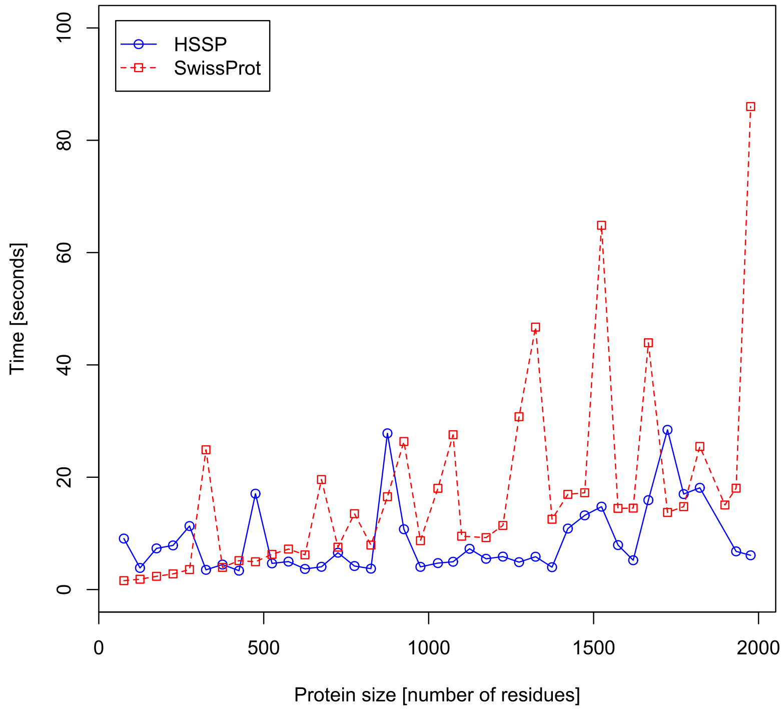
databases. The flow is illustrated in Figure 2.4. See Chapter 2 about homology for more details. Conservation can be calculated within several minutes when we use HSSP database or our pipeline with SwissProt; however, homology calculation using UniRef90 or TrEMBL fallback databases takes more than ten times longer (see Figure 4.1). In this chapter, we will denote the method of calculation conservation scores using subscripts by dataset names, and also abbreviate database names. For example, CHEN11$_{SP/U90}$ indicates dataset CHEN11 with conservation scores calculated using homology pipeline with UniRef90 as the fallback database.

4.2 Verifying correlation between homology and functional sites

In this section, we will evaluate experiment described in Section 3.1, i.e. we will calculate average conservation scores for binding and non-binding protein sites and compare them. We trained 10 P2Rank models with default parameters and evaluated them on all datasets and averaged the results.

As presented in Table 4.2 and Figure 4.2, protein-ligand binding sites indeed have higher average conservation score than their complementary residues. Conservation scores calculated using our homology pipeline with TrEBML as the fallback database achieve worse results than conservation scores calculated using HSSP database or the pipeline with UniRef90 as the second database. This is probably caused by including too many sequences in the MSA, which implies higher conservation scores for all residues. The results also indicate that the correlation between protein active sites and homology is the lowest for JOINED dataset. Thus, we expect that the improvement of prediction using conservation on this dataset will be also lowest.

Unfortunately, not all binding sites are necessarily conserved (see histogram in Figure 4.2). Moreover, there exists no strict threshold that would separate binding and non-binding sites with relatively high precision.



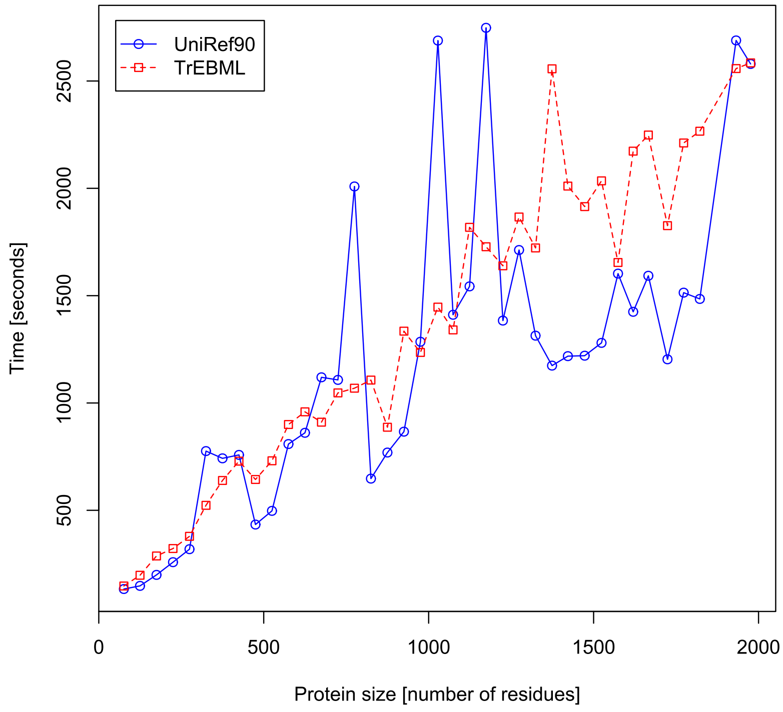


Figure 4.1: Plot of time needed for homology calculation vs. protein size.

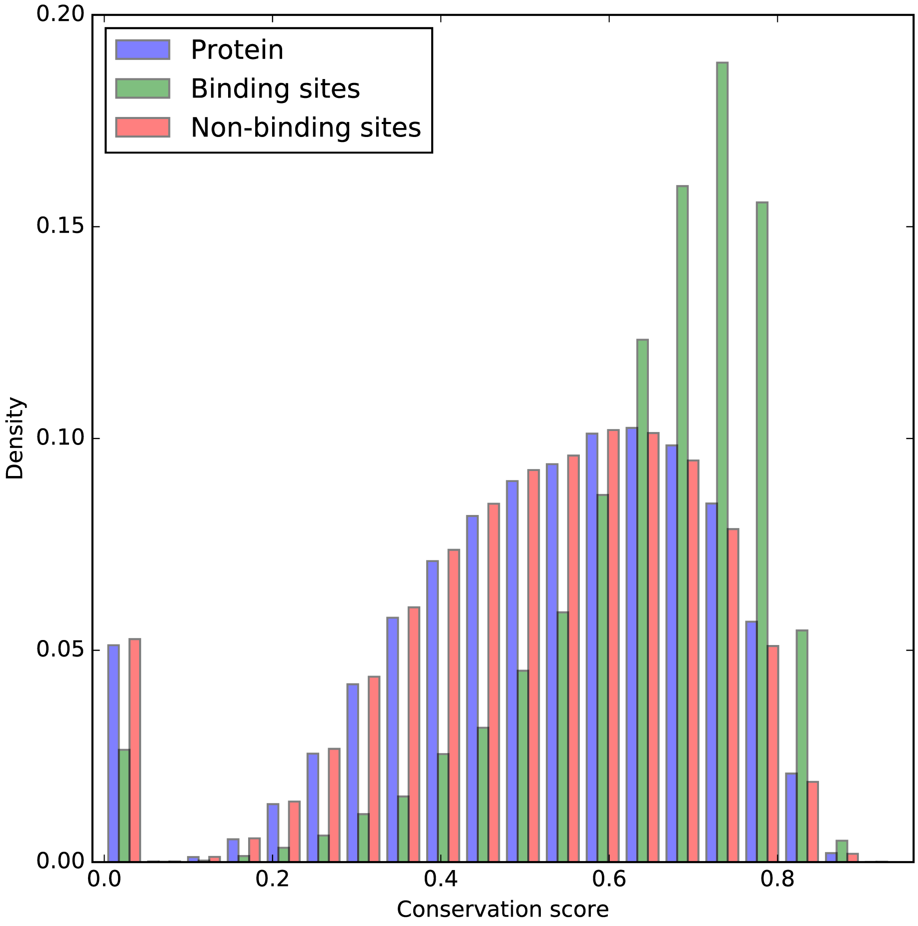


Figure 4.2: Distribution of conservation scores on HOLO4K$_{HSSP}$ dataset.

4.3 P2Rank success metrics

Before we attempt to improve the prediction results, we will describe how the prediction is evaluated. Since we focus on improving P2Rank prediction, we will adopt its evaluation system, and thus, we can easily compare the results.

Pocket is defined as a set of indices of the original protein atoms. For our evaluation, we use D_{CA} metric — the distance between the center of a pocket and any ligand atom. A binding site is then considered correctly predicted (true pocket) if D_{CA} is not greater than an arbitrary threshold, which is usually 4 Å. It is the most commonly used detection criterion that has been used in virtually all previous studies.

When evaluating a dataset, we adopted ligand-centric counting i.e. in order to achieve 100%, each ligand must a have corresponding predicted binding site that meets the D_{CA} 4 criterion. Furthermore, to account for the pocket order (pocket ranking) we use Top-(n+m) success rate. Let us assume that a protein has n ligands. Then, for every ligand of this protein, its binding site must be correctly predicted within the first $n + m$ positions. For brevity we will refer to

Top-(n+0) as to Top-n. Most studies use Top-1 or Top-3; however, for datasets where proteins have more than one or three ligands such as HOLO4K, this metric cannot even achieve 100% and that's why we use the Top-(n+m) metric.

To achieve our goal and verify whether conservation score and protein-ligand binding sites correlate, we implemented several statistics into P2Rank evaluation routine (Evaluation class). After a Random Forest is trained on a protein, any information about the protein, its features and predicted pockets can be saved before the allocated memory is released and a new protein is loaded. At that moment, we calculate average conservation scores of the protein, its binding sites, predicted pockets etc. Finally, we aggregate the collected entries when the whole dataset is processed. All results we present in this work are direct output of P2Rank.

For experiments on JOINED and HOLO4K datasets, we trained 10 models on CHEN11, evaluated them on particular dataset, and then we averaged the results. Results on CHEN11 dataset are average of 10 5-fold cross-validations. A spreadsheet with all statistics collected during our experiments is included in the attachments.

4.4 Pocket rescoring

We already verified that protein-ligand binding sites have higher conservation scores. In this section, we want to verify whether homology is capable of improving P2Rank prediction by reordering predicted pockets. True pockets have higher average conservation than false pockets (see Table 4.3). The difference is more notable when using homology from HSSP database. We tried to exploit the evolutionary information and reorder the pockets using a combination of P2Rank ligandibility score and average pocket conservation score. We compute the combined score as $cs^{ex} \cdot s$, where s is P2Rank ligandibility score, cs is average pocket conservation score, and ex is an arbitrary parameter. Using paramater search on JOINED dataset, we discovered that $ex = 1$ achieves the best results.

Homology alone performs a lot worse as ranking method than the P2Rank ligandibility score calculated by RandomForest. On the other hand, the combined score ($s \cdot cs$) achieves better results on all datasets except for JOINED dataset (see Table 4.4). We believe that P2Rank ranking algorithm works almost perfectly on JOINED because there are not as many ligands per protein and because the correlation between homology and binding sites is significantly lower on JOINED dataset as noted in the previous experiment. We also see that the differences between the methods of calculating conservation scores are subtle.

4.5 Including homology as native P2Rank feature

Since homology proved to be effective in pocket rescoring, we try to exploit conservation scores for localizing protein-ligand binding sites. We implement homology as three native P2Rank features (C,CS and CCS), which we describe in Section 3.3 in the previous chapter. We train a new P2Rank model with the same feature

Dataset	$\overline{\text{pocket } cs}$	$\overline{\text{true pocket } cs}$	$\overline{\text{false pocket } cs}$
CHEN11$_{HSSP}$	0.574	0.617	0.554
JOINED$_{HSSP}$	0.558	0.609	0.543
HOLO4K$_{HSSP}$	0.557	0.622	0.537
CHEN11$_{SP/T}$	0.623	0.662	0.605
JOINED$_{SP/T}$	0.558	0.609	0.543
HOLO4K$_{SP/T}$	0.623	0.675	0.608
CHEN11$_{SP/U90}$	0.554	0.609	0.528
JOINED$_{SP/U90}$	0.558	0.608	0.543
HOLO4K$_{SP/U90}$	0.578	0.646	0.558

Table 4.3: Average conservation scores of P2Rank predicted pockets.

Dataset	Top-n			Top-(n+2)			$\overline{\text{true pocket rank}}$		
	s	cs	$cs \cdot s$	s	cs	$cs \cdot s$	s	cs	$cs \cdot s$
CHEN11$_{HSSP}$	54.0	50.0	**54.1**	57.4	56.5	57.4	1.74	1.88	**1.71**
JOINED$_{HSSP}$	**73.7**	57.1	73.0	**79.7**	73.3	78.9	**1.54**	2.14	1.62
HOLO4K$_{HSSP}$	76.5	61.7	**76.9**	83.4	76.1	**83.5**	2.89	3.54	**2.85**
CHEN11$_{SP/T}$	53.6	50.2	**53.8**	57.0	56.3	**57.2**	1.76	1.85	**1.71**
JOINED$_{SP/T}$	**73.7**	57.4	72.8	**79.5**	73.7	78.9	**1.54**	2.13	1.62
HOLO4K$_{SP/T}$	76.5	60.0	**77.1**	83.5	75.0	**83.7**	2.90	3.66	**2.82**
CHEN11$_{SP/U90}$	53.7	50.2	**54.0**	57.3	56.2	**57.6**	**1.72**	1.89	1.74
JOINED$_{SP/U90}$	**73.7**	57.1	72.8	**79.7**	73.4	78.8	**1.54**	2.12	1.62
HOLO4K$_{SP/U90}$	76.5	62.1	**77.2**	83.5	76.0	**83.7**	2.90	3.49	**2.80**

Table 4.4: Results of pocket rescoring using conservation scores. The best results are in bold. We want the success metric to be the highest and the average true pocket rank to be the lowest.

Dataset	Top-n				Top-(n+2)			
	None	C	CC	CCS	None	C	CC	CCS
CHEN11$_{HSSP}$	**54.0**	53.7	51.5	53.8	57.4	**57.6**	55.5	57.3
JOINED$_{HSSP}$	73.7	**74.5**	73.1	73.9	**79.7**	79.6	78.2	78.9
HOLO4K$_{HSSP}$	76.5	**77.9**	77.2	77.0	83.4	**84.0**	83.2	83.3
CHEN11$_{SP/T}$	53.6	**53.9**	51.9	53.6	57.4	**57.8**	55.8	57.4
JOINED$_{SP/T}$	73.7	**74.4**	73.1	73.9	**79.7**	79.5	78.0	79.2
HOLO4K$_{SP/T}$	76.5	**77.5**	76.8	76.5	83.4	83.4	82.8	82.5
CHEN11$_{SP/U90}$	53.7	**53.9**	51.9	54.4	57.3	**57.8**	55.9	57.8
JOINED$_{SP/U90}$	73.7	**74.5**	72.9	74.3	79.7	79.7	77.9	79.4
HOLO4K$_{SP/U90}$	76.5	**77.9**	77.3	77.0	83.4	**83.7**	83.0	82.7

Table 4.5: Results of P2Rank prediction with various implementations of conservation feature.

set as in the previous section, but this time we include the newly implemented conservation feature.

Table 4.5 contains the results of this experiment. Adding conservation feature improved the pocket prediction in almost all cases. The table clearly shows that our Conservation (C) implementation of homology scores better than the other methods probably because of its higher complementarity to protrusion. The differences between different sources of conservation scores are very small, which verifies that our proposed homology pipeline works correctly.

Since we use RandomForest as predicting classifier, we can look at the feature importances. Feature importance indicates how much a feature contributes to the prediction. Homology plays important role in predicting protein-ligand binding sites. In fact, conservation feature was ranked as second most important feature after protrusion (see Table 4.6). Although implementation CCS (Conservation Cloud Scaled) achieved the highest importance, the feature is not as complementary to other features as other conservation implementations. Therefore, it achieved worse results. We also tried to add all conservation feature implementations; however, the improvement was not as high for the same reason as with CCS.

When we train a model solely using conservation feature, we achieve Top-n score 34.7 on HOLO4K$_{HSSP}$. This implies that homology contains a lot of information about protein-lignad binding sites; however, other important features such as protrusion share most of the information. Thus, the improvement when combined together is not as significant as expected.

Homology not only improves the pocket detection in terms of P2Rank success metrics, but it also improves the quality of the prediction. After adding conservation feature (C), the average number of predicted pockets is lower and the ligand coverage higher, while achieving the same or better success score (see Table 4.7). We calculate the ligand coverage as the number of SAS points within 2 Å to any ligand atom that were predicted as binding by RandomForest.

None		C	
protrusion:	0.0197	protrusion:	0.0151
apRawInvalids:	0.0016	conservation:	0.0042
apRawValids:	0.0013	apRawInvalids:	0.0015

CC		CCS	
protrusion:	0.0157	protrusion:	0.0089
conservationcloud:	0.0035	conservationcloudscaled:	0.0062
apRawInvalids:	0.0017	apRawInvalids:	0.0015

Table 4.6: Top 3 feature importances on HOLO4K$_{HSSP}$ dataset.

Conservation method	$\overline{\text{\#predicted pockets}}$	Ligand coverage
No conservation	7.814	0.541
Conservation (C)	7.243	0.572
Conservation Cloud (CC)	**7.200**	0.565
Conservation Cloud Scaled (CCS)	7.507	**0.583**

Table 4.7: Average number of predicted pockets and ligand coverage on HOLO4K$_{HSSP}$ dataset.

Conclusion

The goal of this work was to develop a web application capable of visualization of P2Rank results as well as conservation scores. Furthermore, we wanted to improve P2Rank prediction using evolutionary homology.

In this thesis, we have proposed PrankWeb, a modern web application for convenient structure and sequence visualization of a protein and its protein-ligand binding sites as well as evolutionary homology.

We elaborated PrankWeb components with its design, and described its functions and how to use them. We compared several molecular and sequence visualizers and explained why LiteMol together with Protael represent the best choice. To guarantee immediate results and reduce CPU server demand, we stored precomputed results for the whole PDB database locally.

We constructed a pipeline capable of calculating conservation scores, and verified that evolutionary homology correlates with protein active sites. Moreover, we showed that P2Rank prediction can be improved by rescoring predicted pockets using conservation score, and also by including homology as native P2Rank feature. PrankWeb allows to display both predicted pockets as well as the computed conservation scores to see how they correlate with each other.

We hope that PrankWeb will provide a quick and convenient way for scientists to analyze particular protein. Alternatively, PrankWeb REST API can be used by other developers to build on our work.

Bibliography

[1] Rene F Kratz. *Biology For Dummies.* Wiley, June 2010.

[2] Radoslav Krivák and David Hoksza. Improving protein-ligand binding site prediction accuracy by classification of inner pocket points using local features. *Journal of cheminformatics*, 7(1):12, 2015.

[3] Radoslav Krivák and David Hoksza. P2RANK: Knowledge-Based Ligand Binding Site Prediction Using Aggregated Local Features. *Algorithms for Computational Biology*, January 2015.

[4] John A Capra, Roman A Laskowski, Janet M Thornton, Mona Singh, and Thomas A Funkhouser. Predicting protein ligand binding sites by combining evolutionary sequence conservation and 3D structure. *PLoS Computational Biology*, 5(12):e1000585, 2009.

[5] Stéphanie Pérot, Olivier Sperandio, Maria A Miteva, Anne-Claude Camproux, and Bruno O Villoutreix. Druggable pockets and binding site centric chemical space: a paradigm shift in drug discovery. *Drug discovery today*, 15(15):656–667, 2010.

[6] Xiliang Zheng, LinFeng Gan, Erkang Wang, and Jin Wang. Pocket-based drug design: exploring pocket space. *The AAPS journal*, 15(1):228–241, 2013.

[7] Lei Xie, Li Xie, and Philip E Bourne. Structure-based systems biology for analyzing off-target binding. *Current opinion in structural biology*, 21(2):189–199, 2011.

[8] Janez Konc and Dušanka Janežič. Binding site comparison for function prediction and pharmaceutical discovery. *Current opinion in structural biology*, 25:34–39, 2014.

[9] Didier Rognan. Docking methods for virtual screening: Principles and recent advances. *Virtual Screening: Principles, Challenges, and Practical Guidelines*, pages 153–176, 2011.

[10] Helen M Berman, John Westbrook, Zukang Feng, Gary Gilliland, Talapady N Bhat, Helge Weissig, Ilya N Shindyalov, and Philip E Bourne. The protein data bank. *Nucleic acids research*, 28(1):235–242, 2000.

[11] Martin Weisel, Ewgenij Proschak, and Gisbert Schneider. PocketPicker: analysis of ligand binding-sites with shape descriptors. *Chemistry Central Journal*, 1(1):7, 2007.

[12] Manfred Hendlich, Friedrich Rippmann, and Gerhard Barnickel. LIGSITE: automatic and efficient detection of potential small molecule-binding sites in proteins. *Journal of Molecular Graphics and Modelling*, 15(6):359–363, 1997.

[13] Roman A Laskowski. SURFNET: a program for visualizing molecular surfaces, cavities, and intermolecular interactions. *Journal of molecular graphics*, 13(5):323–330, 1995.

[14] Alasdair TR Laurie and Richard M Jackson. Q-SiteFinder: an energy-based method for the prediction of protein–ligand binding sites. *Bioinformatics*, 21(9):1908–1916, 2005.

[15] Mizuki Morita, Shugo Nakamura, and Kentaro Shimizu. Highly accurate method for ligand-binding site prediction in unbound state (apo) protein structures. *Proteins: Structure, Function, and Bioinformatics*, 73(2):468–479, 2008.

[16] John A Capra and Mona Singh. Predicting functionally important residues from sequence conservation. *Bioinformatics*, 23(15):1875–1882, 2007.

[17] Jeffrey Skolnick and Michal Brylinski. FINDSITE: a combined evolution/structure-based approach to protein function prediction. *Briefings in Bioinformatics*, 10(4):378, 2009.

[18] Peng Chen, Jianhua Z. Huang, and Xin Gao. LigandRFs: random forest ensemble to identify ligand-binding residues from sequence information alone. *BMC Bioinformatics*, 15(15):S4, 2014.

[19] Chris Kauffman and George Karypis. LIBRUS: combined machine learning and homology information for sequence-based ligand-binding residue prediction. *Bioinformatics*, 25(23):3099, 2009.

[20] Zhijun Qiu and Xicheng Wang. Improved prediction of protein ligand-binding sites using random forests. *Protein and peptide letters*, 18(12):1212–1218, 2011.

[21] Bingding Huang. MetaPocket: a meta approach to improve protein ligand binding site prediction. *OMICS A Journal of Integrative Biology*, 13(4):325–330, 2009.

[22] Zengming Zhang, Yu Li, Biaoyang Lin, Michael Schroeder, and Bingding Huang. Identification of cavities on protein surface using multiple computational approaches for drug binding site prediction. *Bioinformatics*, 27(15):2083, 2011.

[23] A. Shrake and J.A. Rupley. Environment and exposure to solvent of protein atoms. Lysozyme and insulin. *Journal of Molecular Biology*, 79(2):351–371, September 1973.

[24] Warren L DeLano. The PyMOL Molecular Graphics System. De-Lano Scientific, San Carlos, CA, USA. *http://www.pymol.org*, 2002.

[25] Shane P. Tully, Thomas M. Stitt, Robert D. Caldwell, Brian J. Hardock, Robert M. Hanson, and Przemyslaw Maslak. Interactive Web-Based Pointillist Visualization of Hydrogenic Orbitals Using Jmol. *Journal of Chemical Education*, 90(1):129–131, January 2013.

[26] Oracle. Java and Google Chrome Browser. https://java.com/en/download/faq/chrome.xml, April 2016. Accessed: 11 April 2017.

[27] N. Rego and D. Koes. 3Dmol.js: molecular visualization with WebGL. *Bioinformatics*, 31(8):1322–1324, December 2014.

[28] Robert M. Hanson, Jaime Prilusky, Zhou Renjian, Takanori Nakane, and Joel L. Sussman. JSmol and the Next-Generation Web-Based Representation of 3D Molecular Structure as Applied toProteopedia. *Israel Journal of Chemistry*, 53(3-4):207–216, April 2013.

[29] Marco Biasini. PV-WebGL-based protein viewer. *Zenodo*, 2014.

[30] National Center for Biotechnology Information. iCn3D Overview: web-based 3D structure viewer. https://www.ncbi.nlm.nih.gov/Structure/icn3d/docs/icn3d_about.html, 2017. Accessed: 11 April, 2017.

[31] Alexander S. Rose and Peter W. Hildebrand. NGL Viewer: a web application for molecular visualization. *Nucleic Acids Research*, 43(W1):W576, 2015.

[32] Alexander S. Rose, Anthony R. Bradley, Yana Valasatava, Jose M. Duarte, Andreas Prlić, and Peter W. Rose. Web-based Molecular Graphics for Large Complexes. In *Proceedings of the 21st International Conference on Web3D Technology*, Web3D '16, pages 185–186, New York, NY, USA, 2016. ACM.

[33] David Sehnal. LiteMol / 3D macromolecular data in the browser. http://webchemdev.ncbr.muni.cz/Litemol/, 2017. Accessed: 11 April, 2017.

[34] Kiran Mukhyala and Alexandre Masselot. Visualization of protein sequence features using JavaScript and SVG with pviz.js. *Bioinformatics*, 30(23):3408–3409, 2014.

[35] Mayya Sedova, Lukasz Jaroszewski, and Adam Godzik. Protael: protein data visualization library for the web. *Bioinformatics*, 32(4):602–604, 2016.

[36] Rolf Apweiler, Amos Bairoch, Cathy H Wu, Winona C Barker, Brigitte Boeckmann, Serenella Ferro, Elisabeth Gasteiger, Hongzhan Huang, Rodrigo Lopez, Michele Magrane, et al. UniProt: the universal protein knowledgebase. *Nucleic acids research*, 32(suppl 1):D115–D119, 2004.

[37] Baris E Suzek, Hongzhan Huang, Peter McGarvey, Raja Mazumder, and Cathy H Wu. UniRef: comprehensive and non-redundant UniProt reference clusters. *Bioinformatics*, 23(10):1282–1288, 2007.

[38] Robbie P. Joosten, Tim A.H. te Beek, Elmar Krieger, Maarten L. Hekkelman, Rob W.W. Hooft, Reinhard Schneider, Chris Sander, and Gert Vriend. A series of PDB related databases for everyday needs. *Nucleic Acids Research*, 39(suppl_1):D411, November 2011.

[39] H. Ashkenazy, E. Erez, E. Martz, T. Pupko, and N. Ben-Tal. ConSurf 2010: calculating evolutionary conservation in sequence and structure of proteins and nucleic acids. *Nucleic Acids Research*, 38(Web Server):W529–W533, May 2010.

[40] Robert C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792, 2004.

[41] Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic acids research*, 25:3389–3402, September 1997.

[42] Weizhong Li and Adam Godzik. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22(13):1658, 2006.

[43] Bingding Huang and Michael Schroeder. LIGSITE csc: predicting ligand binding sites using the Connolly surface and degree of conservation. *BMC structural biology*, 6(1):19, 2006.

[44] Ke Chen, Marcin J Mizianty, Jianzhao Gao, and Lukasz Kurgan. A critical comparative assessment of predictions of protein-binding sites for biologically relevant organic compounds. *Structure*, 19(5):613–621, 2011.

[45] Alexey G Murzin, Steven E Brenner, Tim Hubbard, and Cyrus Chothia. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *Journal of molecular biology*, 247(4):536–540, 1995.

[46] Michael J Hartshorn, Marcel L Verdonk, Gianni Chessari, Suzanne C Brewerton, Wijnand TM Mooij, Paul N Mortenson, and Christopher W Murray. Diverse, high-quality test set for the validation of protein- ligand docking performance. *Journal of medicinal chemistry*, 50(4):726–741, 2007.

[47] Peter Schmidtke, Catherine Souaille, Frédéric Estienne, Nicolas Baurin, and Romano T Kroemer. Large-scale comparison of four binding site detection algorithms. *Journal of chemical information and modeling*, 50(12):2191–2200, 2010.

List of Figures

List of Tables

List of Abbreviations

ASA	Accessible Surface Area
C	Conservation
CC	Conservation Cloud
CCS	Conservation Cloud Scaled
DNA	Deoxyribonucleic Acid
HSSP	Homology derived Secondary Structure of Proteins
JSD	Jensen-Shannon Divergence
MSA	Multiple Sequence Alignment
PDB	Protein Data Bank
RCSB	Research Collaboratory for Structural Bioinformatics
RNA	Ribonucleic Acid
SAS	Solvent Accessible Surface
SVG	Scalable Vector Graphics

Attachments

We attach a compact disk (CD) with this thesis, that contains:

- PrankWeb source code (including LiteMol).
- P2Rank source code and default pretrained models.
- LaTeX source code of this document and its figures.
- Collected results for all experiments.
- Manuals, which are also included below.

PrankWeb setup

Building PrankWeb

1. Install Node.js and Java.
2. Clone this repository including all submodules:

   ```
   git clone --recursive \
   https://github.com/jendelel/PrankWebApp.git
   ```

   Alternatively, copy the `source` directory from the attached CD.
3. Now, build PrankWeb using command `./gradlew war`

   This will compile all submodules and create ROOT.war file in build/libs directory.

Configuring PrankWeb

To actually, run PrankWeb server, you will need to download and unpack JBoss WildFly application server[1] and setup environment variable JBOSS_HOME to WildFly path.

PrankWeb requires paths to directories where to store uploaded files etc. Everything is stored in a directory, which we call `PrankData`. We recommend to store all the data in `PrankData` directory; nevertheless, the directory must contain `prankweb.properties` file, that specifies all necessary paths and configurations. The file contains pairs of keys and values. Setup the following properties:

- **prank.installdir** — Path to P2Rank distro directory (needed for running P2Rank).
- **database.pdb** — Path to local PDB database[2]

[1] See `http://wildfly.org/downloads/`.

[2] See `http://www.rcsb.org/pdb/static.do?p=download/ftp/index.html` how to get a local copy.

- **database.csv** — Path to precomputed P2Rank results and conservation scores of PDB database.
- **uploads.pdb** — Path to directory where to store uploaded user PDB files.
- **uploads.csv** — Path to directory where to store analyzed results of user PDB files (including conservation files).
- **conservation.hssp** — Path to local HSSP database[3]
- **conservation.script** — Path to an executable that computes conservation score from protein sequence FASTA file.[4]

The following properties can be setup optionally:

- **queue.size** — Size of the queue of analyses (both P2rank and conservation pipeline).
- **pool.coresize** — Number of analyses that can run simultaneously.
- **pool.maxsize** — Number of analyses that can run simultaneously if the queue is full.

Create a symbolic link in JBOSS_HOME/standalone/data/PrankWeb pointing to PrankData directory using these commands:
Windows

```
cd /d %JBOSS_HOME%\standalone\data
mklink /D PrankWeb {path to PrankData directory}
```

Linux:

```
cd $JBOSS_HOME/standalone/data
ln -s -d {path to PrankData directory} PrankWeb
```

Since the server runs P2Rank internally, please increase the memory limit in `JBOSS_HOME/bin/` in file `standalone.conf` on Linux or `standalone.bat.conf` on Windows. To change the limit, change the `-Xmx` option in `JAVA_OPTS` statement to at least `1024m`. For example:

```
JAVA_OPTS="-Xms64m -Xmx1024m -XX:MetaspaceSize=96M
   -XX:MaxMetaspaceSize=256m -Djava.net.preferIPv4Stack=true"
```

Enabling GZIP compression (optional)

To enable GZIP compression of the web server, add the two lines marked with `**` in Figure below to `standalone.xml` in `JBOSS_HOME/standalone/configuration` directory.

[3] See `http://swift.cmbi.ru.nl/gv/hssp/`.

[4] See `https://github.com/jendelel/calc_protein_conservation`.

```
...
<subsystem xmlns="urn:jboss:domain:undertow:3.1">
    <buffer-cache name="default"/>
    <server name="default-server">
        <http-listener name="default" socket-binding="http"
            redirect-socket="https" enable-http2="true"/>
        <https-listener name="https" socket-binding="https"
            security-realm="ApplicationRealm" enable-http2="true"/>
        <host name="default-host" alias="localhost">
            <location name="/" handler="welcome-content"/>
            <filter-ref name="server-header"/>
            <filter-ref name="x-powered-by-header"/>
            **<filter-ref name="gzipFilter"
                predicate="regex[pattern='(?:application/javascript|
                text/css|text/html|text/plain)(;.*)?',
                value=%{o,Content-Type}, full-match=true]"/>**
        </host>
    </server>
    <servlet-container name="default">
        <jsp-config/>
        <websockets/>
    </servlet-container>
    <handlers>
        <file name="welcome-content"
            path="${jboss.home.dir}/welcome-content"/>
    </handlers>
    <filters>
        <response-header name="server-header"
            header-name="Server" header-value="WildFly/10"/>
        <response-header name="x-powered-by-header"
            header-name="X-Powered-By" header-value="Undertow/1"/>
        **<gzip name="gzipFilter"/>**
    </filters>
</subsystem>
...
```

Excerpt from `standalone.xml` WildFly configuration illustrating how to enable GZIP compression. Added lines are enclosed between `**`

Running PrankWeb

After you setup JBoss WildFly, just run `standalone.sh` (Linux), or alternatively, `standalone.bat` (Windows) in `JBOSS_HOME/bin` directory and copy the `ROOT.war` file to `$JBOSS_HOME/standalone/deployments`, or run `./gradlew deploy` command from the project directory.

PrankWeb REST API

In Section 1.1.2, we explained why we use REST for communication between the frontend and the backend. One of the main advantages is that other developers can use the REST API and build upon our work. The URIs (Uniform Resource Identifiers) follow pattern `/api/origin/type/id`, where `origin` is either `upload` or `id` identifying whether the original PDB file was uploaded by user or downloaded from data bank. The `type` indicates type of file requested by user. Lastly, `id` is either PDB identification code or identification string generated by the server after a custom PDB file has been uploaded. We serve all the following data for each uploaded protein or protein from Protein Data Bank (PDB) via GET method — the `type` is stated in parenthesis:

- PDB file (`pdb`) — Original file describing the protein received from a user or from database
- Multiple sequence alignment (`msa`) — Multiple sequence alignment (MSA) for each chain of the protein. See Chapter 2 about homology.
- Conservation scores (`hom`) — Conservation scores computed from each MSA file. See Chapter 2 about homology.
- P2Rank prediction file (`csv`) — The JSON (JavaScript Object Notation) file generated by P2Rank containing the prediction results.
- Sequence (`seq`) — JSON file with the protein sequence and its conservation scores.
- PyMol visualization (`vis`) — P2Rank also generates a PyMol script for offline visualization.
- Package file (`all`) — A ZIP file containing all files listed above.

Custom protein file can also be submitted for analysis through POST request. The URI is: `/analyze/file_upload`. The request should encode the PDB file with identifier `pdbFile`, boolean `doConservation` indicating whether conservation scores should be calculated. Optionally, `pdbId` or MSA files ending with `.fasta` can be included to speed up the analysis. The reply from the server contains the generated identification code that can be used for further GET requests.

Homology pipeline setup

Installing all necessary tools

1. Download BLAST+ suite.[5]
2. Install/unpack the BLAST+ archive and make sure the directory is in `PATH`.
3. Download CD-HIT.[6]
4. Unpack CD-HIT archive, run `make` command and make sure cd-hit is in `PATH`.
5. Download MUSCLE tool.[7]
6. Unpack MUSCLE archive, and again add the directory to `PATH`.
7. Download Python script for Jensen-Shannon divergence method.[8]
8. Finally, unpack the pipeline archive. The archive is attached to this thesis and also available at GitHub.[9]
9. Copy the archive contents to the directory where Python script for Jensen-Shannon divergence (`score_conservation.py`) is located.

Installing databases

1. Create a directory where you want to store the databases.
2. Add environmental variable `BLASTDB={path to the directory}`
3. Change directory to BLASTDB directory and run these commands. Each command will download and parse one database. We recommend to use SwissProt and UniRef90, but using TrEBML instead of UniRef90 is also possible.[10] Note that it will take a lot of time.

```
curl {UniRef90 URI} | gunzip | makeblastdb -out uniref90 \
-dbtype prot -title UniRef90 -parse_seqids
curl {SwissProt URI} | gunzip | makeblastdb -out swissprot \
-dbtype prot -title SwissProt -parse_seqids
curl {TrEMBL URI} | gunzip | makeblastdb -out trembl \
-dbtype prot -title TrEMBL -parse_seqids
```

[5] Available at ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/.

[6] Available at https://github.com/weizhongli/cdhit/releases.

[7] Available at http://www.drive5.com/muscle/downloads.htm.

[8] Available at http://compbio.cs.princeton.edu/conservation/index.html.

[9] See https://github.com/jendelel/calc_protein_conservation

[10] Available at http://www.uniprot.org/downloads.

Running the script

1. Change directory (`cd`) to the directory where you copied the contents of pipeline archive including `calc_conservation.sh`.

2. Run command: `./calc_conservation.sh {path to your fasta file}`

When using from a Java Application such as PrankWeb, please edit the paths at the beginning of `calc_conservation_incl_vars.sh` and specify path to this file in the PrankWeb configuration file (`prankweb.properties`).