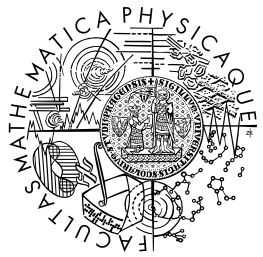

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Lukáš Polák

Extension of web-based interface for protein binding sites prediction

Department of Software Engineering

Supervisor of the bachelor thesis:	doc. RNDr. David Hoksza, Ph.D.
Study programme:	Programming and software development Bc.
Study branch:	IPP2

Prague 2023

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

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 ..... date ....

Author's signature

Dedication. It is nice to say thanks to supervisors, friends, family, book authors and food providers.

Title: Extension of web-based interface for protein binding sites prediction

Author: Lukáš Polák

Department: Department of Software Engineering

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

Abstract: Abstracts are an abstract form of art. Use the most precise, shortest sentences that state what problem the thesis addresses, how it is approached, pinpoint the exact result achieved, and describe the applications and significance of the results. Highlight anything novel that was discovered or improved by the thesis. Maximum length is 200 words, but try to fit into 120. Abstracts are often used for deciding if a reviewer will be suitable for the thesis; a well-written abstract thus increases the probability of getting a reviewer who will like the thesis.

Keywords: bioinformatika web software protein

Contents

Introduction

Protein ligand-binding sites are a vital aspect of nowaday's drug discovery and development. Identification of the potential binding sites allows understanding of various molecule interactions, which may lead to interesting conclusions. Recognizing the binding sites is thus very important for further bioinformatic research and studies [1].

P2Rank is a machine-learning based Java tool developed at MFF UK which allows users to predict the ligand-binding sites for a given protein. P2Rank works standalone and outperforms most of the existing binding sites prediction tools [2]. PrankWeb is a web-based tool which is providing an user-friendly interface for P2Rank. A significant difference between PrankWeb and other web-based tools is that PrankWeb does not employ either JMol nor JSMol for online visualization of the results [3]. Both JMol and JSMol are Java based. Java applets are rather old-fashioned and in most cases even deprecated. The original PrankWeb auhtors decided to use the LiteMol and Protael libraries for the online visualization. Although these libraries are still working, they are not actively developed anymore.

There are two main goals of this thesis. The first goal is to update the PrankWeb interface to use different libraries for the visualization, namely MolStar and RCSB Saguaro 1D Feature Viewer. This will improve not only the visual appearance of the results, but also the performance. Furthermore, the updated libraries are actively developed and thus potentially more reliable in the future. The second goal is to update the PrankWeb architecture, so that more computing may be done on the predicted binding sites both on the client-side and the server-side. This introduces a potential to create custom plug-ins for the web interface.

The thesis should present a working version of the updated PrankWeb tool. The updated tool should keep the current functionality of the website and visualize the results via a more user-friendly interface. The tool should also be able to perform computations on the predicted binding sites.

In the first chapter, the reader will be introduced to the basics of protein and ligand-binding sites problematic, later in this chapter will we cover the

current PrankWeb architecture and the P2Rank tool itself. The reader will also be briefly introduced to similar web-tools. The second chapter will cover the programming part of the work. Firstly, we will introduce the usage of the updated libraries and other frontend design decisions. Secondly, we will cover the plug-in architecture including both the client-side and the server-side computations in the respective subchapters. In the third chapter, we will cover the tool usage from two perspectives - the user and the developer. The conclusion will cover the results and the potential extensions to the tool.

Chapter 1

Introduction and background

In this chapter, we will discuss the basics of protein and ligand-binding sites problematic, later in this chapter will we cover the current PrankWeb architecture and the P2Rank tool itself. The reader will also be briefly introduced to similar web-tools.

1.1 Introduction to molecular biology

TODO

1.2 P2Rank tool

P2Rank allows its users to predict the ligand-binding sites for a given protein. In contrast to other projects, P2Rank was one of the first tools to employ machine learning for predicting the pockets. P2Rank outperforms most of the existing binding sites prediction tools [2]. Most of the other tools include geometry-based, energetic-based, or template-based methods that are not as efficient and rather outdated. Moreover, P2Rank works as a standalone application and is fully automated, which makes the tool very intuitive and easy to use.

P2Rank works with specific file formats such as PDB and PDBx/mmCIF. maybe insert some detailed explanation of the formats? After running the tool on a specific protein structure file, the tool will provide a CSV output file with the prediction and residue-level scores. The output file includes predicted pockets, their ranks, center coordinates, adjacent residues, related surface atoms and a probability score.

The following command would run P2Rank on the protein structure file `1fbl.pdb`:

```
prank predict -f test_data/1fbl.pdb
```

The tool was written in Java and requires only the JRE to run. Additionally, the source codes are publicly available at GitHub[1]. This allows the users to potentially modify the tool to their respective needs.

1.3 PrankWeb architecture

PrankWeb consists of several components that cooperate together. Currently, the application is deployed via Docker[2] containers that are described in a `docker-compose.yml` file. The application consists of the following components:

- **gateway** - a reverse proxy that is responsible for routing the requests to the respective backend services
- **rabbitmq** - a broker that is used for communication between the gateway and the backend services
- **flower** - a tool for monitoring the RabbitMQ broker and Celery workers
- **web-server** - a WSGI server that is responsible for serving the web application
- **executor-p2rank** - a backend service that is responsible for running the P2Rank tool, employs Celery workers
- **executor-docking** - a backend service that is responsible for running the docking tool, employs Celery workers docking not done yet though
- **prometheus** - a tool for monitoring the Docker containers

Now we will present the Docker containers in more detail to get a broader knowledge of the architecture.

[1] `https://github.com/rdk/p2rank`
[2] `https://www.docker.com/`

1.3.1 Gateway

The gateway container is a reverse proxy that is responsible for routing the requests to the backend. In PrankWeb, we utilize the Nginx web server as a reverse proxy. The Nginx configuration file is located in the `gateway/nginx.conf` file. The server configuration includes not only the reverse proxy routes, but a mapping to the Flower and Prometheus services as well.

Moreover, `gateway/Dockerfile` is responsible for the installation of the frontend. The frontend is a React application built via webpack. PrankWeb utilizes two main external libraries for the bioinformatic part of the application, MolStar and RCSB Saguaro Feature 1D Viewer. We will discuss these libraries in more detail in the next chapter.

The entire frontend is written in TypeScript, JavaScript, CSS, SCSS, and HTML.

1.3.2 RabbitMQ

todo + other containers

1.4 Similar web-tools

todo

Chapter 2

More complicated chapter

After the reader gained sufficient knowledge to understand your problem in chapter 1, you can jump to your own advanced material and conclusions.

You will need definitions (see definition 1 below in section 2.1), theorems (theorem 1), general mathematics, algorithms (algorithm 1), and tables (table 2.1). Figures 2.1 and 3.1 show how to make a nice figure. See figure 2.2 for an example of TikZ-based diagram. Cross-referencing helps a lot to keep the necessary parts of the narrative close — use references to the previous chapter with theory wherever it seems that the reader could have forgotten the required context.

See documentation of package `booktabs` for hints on typesetting tables. As a main rule, *never* draw a vertical line.

2.1 Example with some mathematics

Definition 1 (Triplet)**.** *Given stuff X, Y and Z, we will write a* triplet *of the stuff as (X, Y, Z).*

Theorem 1 (Car coloring)**.** *All cars have the same color. More specifically, for any set of cars C, we have*

$$(\forall c_1, c_2 \in C)\ \textsc{Colour}(c_1) = \textsc{Colour}(c_2).$$

Proof. Use induction on sets of cars C. The statement holds trivially for $|C| \leq 1$. For larger C, select 2 overlapping subsets of C smaller than $|C|$ (thus same-colored). Overlapping cars need to have the same color as the cars outside the overlap, thus also the whole C is same-colored. □

This is plain wrong though.

2.2 Extra typesetting hints

Do not overuse text formatting for highlighting various important parts of your sentences. If an idea cannot be communicated without formatting, the sentence

Column A	Column 2	Numbers	More
Asd	QWERTY	123123	–
Asd qsd 1sd	BAD	234234234	This line should be helpful.
Asd	INTERESTING	123123123	–
Asd qsd 1sd	PLAIN WEIRD	234234234	–
Asd	QWERTY	123123	–
Asd qsd 1sd	GOOD	234234299	–
Asd	NUMBER	**123123**	–
Asd qsd 1sd	DIFFERENT	234234234	(no data)

Table 2.1 An example table. Table caption should clearly explain how to interpret the data in the table. Use some visual guide, such as boldface or color coding, to highlight the most important results (e.g., comparison winners).

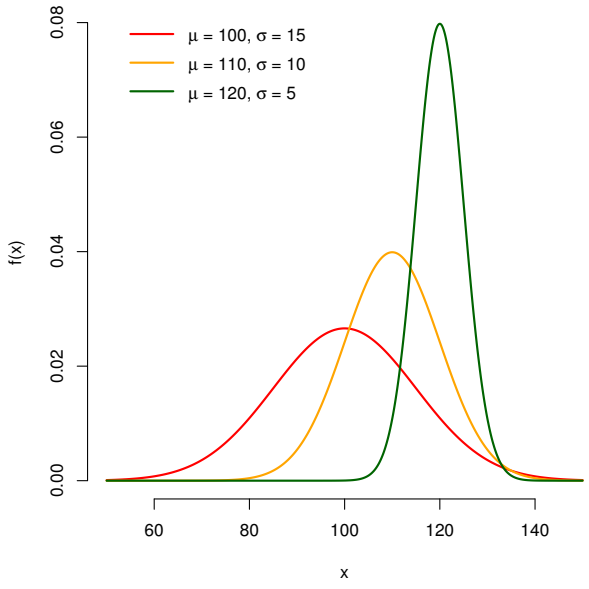

Figure 2.1 A figure with a plot, not entirely related to anything. If you copy the figures from anywhere, always refer to the original author, ideally by citation (if possible). In particular, this picture — and many others, also a lot of surrounding code — was taken from the example bachelor thesis of MFF, originally created by Martin Mareš and others.

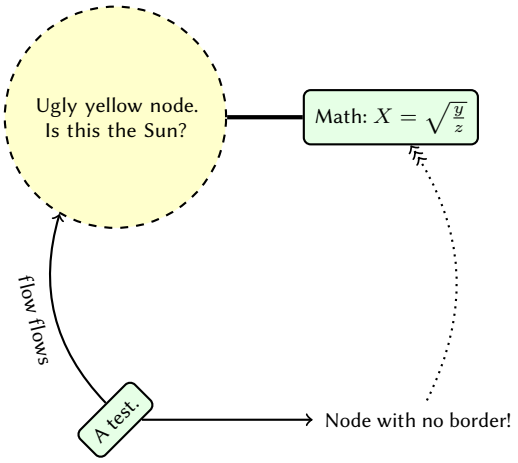

Figure 2.2 An example diagram typeset with TikZ.

Algorithm 1 Algorithm that executes an action with high probability. Do not care about formal semantics in the pseudocode — semicolons, types, correct function call parameters and similar nonsense from 'realistic' languages can be safely omitted. Instead make sure that the intuition behind (and perhaps some hints about its correctness or various corner cases) can be seen as easily as possible.

function ExecuteWithHighProbability(A)
    $r \leftarrow$ a random number between 0 and 1
    $\varepsilon \leftarrow 0.00000000000000000000000000000000000000042$
    if $r \geq \varepsilon$ **then**
        execute A . . . . . *We discard the return value*
    else
        print: `Not today, sorry.`
    end if
end function

probably needs rewriting anyway. Imagine the thesis being read aloud as a podcast — the storytellers are generally unable to speak in boldface font.

Most importantly, do not overuse bold text, which is designed to literally **shine from the page** to be the first thing that catches the eye of the reader. More precisely, use bold text only for 'navigation' elements that need to be seen and located first, such as headings, list item leads, and figure numbers.

Use underline only in dire necessity, such as in the previous paragraph where it was inevitable to ensure that the reader remembers to never typeset boldface text manually again.

Use *emphasis* to highlight the first occurrences of important terms that the reader should notice. The feeling the emphasis produces is, roughly, "Oh my — what a nicely slanted word! Surely I expect it be important for the rest of the thesis!"

Finally, never draw a vertical line, not even in a table or around figures, ever. Vertical lines outside of the figures are ugly.

Chapter 3

Results and discussion

You should have a separate chapter for presenting your results (generated by the stuff described previously, in our case in chapter 2). Remember that your work needs to be validated rigorously, and no one will believe you if you just say that 'it worked well for you'.

Instead, try some of the following:

- State a hypothesis and prove it statistically
- Show plots with measurements that you did to prove your results (e.g. speedup). Use either `R` and `ggplot`, or Python with `matplotlib` to generate the plots.[1] Save them as PDF to avoid printing pixels (as in figure 3.1).
- Compare with other similar software/theses/authors/results, if possible
- Show example source code (e.g. for demonstrating how easily your results can be used)
- Include a 'toy problem' for demonstrating the basic functionality of your approach and detail all important properties and results on that
- Include clear pictures of 'inputs' and 'outputs' of all your algorithms, if applicable

It is sometimes convenient (even recommended by some journals, including Cell) to name the results sub-sections so that they state what exactly has been achieved. Examples follow.

[1] Honestly, the plots from `ggplot` look much better.

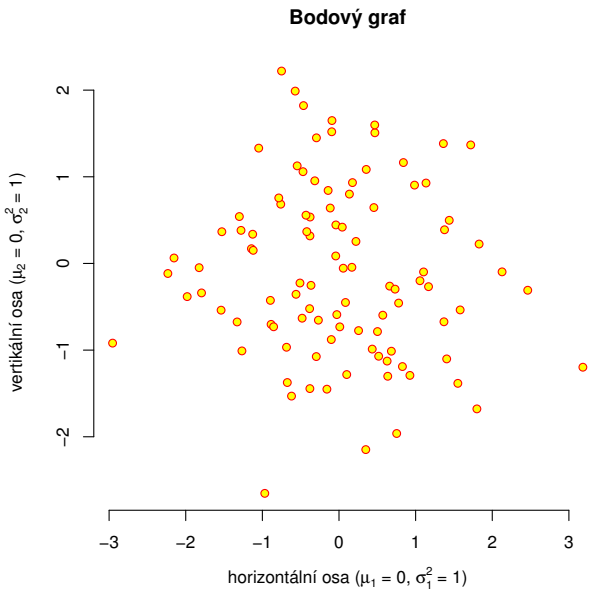

Figure 3.1 This caption is a friendly reminder to never insert figures "in text," without a floating environment, unless explicitly needed for maintaining the text flow (e.g., the figure is small and developing with the text, like some of the centered equations, as in theorem 1). All figures *must* be referenced by number from the text (so that the readers can find them when they read the text) and properly captioned (so that the readers can interpret the figure even if they look at it before reading the text — reviewers love to do that).

3.1 SuperProgram is faster than OldAlgorithm

3.1.1 Scalability estimation

3.1.2 Precision of the results

3.2 Weird theorem is proven by induction

3.3 Amount of code reduced by CodeRedTool

3.3.1 Example

3.3.2 Performance on real codebases

3.4 NeuroticHelper improves neural network learning

3.5 Graphics and figure quality

No matter how great the text content of your thesis is, the pictures will always catch the attention first. This creates the very important first impression of the thesis contents and general quality. Crucially, that also decides whether the thesis is later read with joy, or carefully examined with suspicion.

Preparing your thesis in a way such that this first impression gets communicated smoothly and precisely helps both the reviewer and you: the reviewer will not have a hard time understanding what exactly you wanted to convey, and you will get a better grade.

Making the graphics 'work for you' involves doing some extra work that is often unexpected. At the same time, you will need to fit into graphics quality constraints and guidelines that are rarely understood before you actually see a bad example. As a rule of thumb, you should allocate at least the same amount of time and effort for making the figures look good as you would for writing, editing and correcting the same page area of paragraph text.

3.5.1 Visualize all important ideas

The set of figures in your thesis should be comprehensive and complete. For all important ideas, constructions, complicated setups and results there should be a visualization that the reader can refer to in case the text does not paint the 'mental image' sufficiently well. At the bare minimum, you should have at least 3

figures (roughly corresponding to the 3 chapters) that clearly and unambiguously show:

1. the context of the problem you are solving, optionally with e.g. question marks and exclamation marks placed to highlight the problems and research questions

2. the overall architecture of your solution (usually as a diagram with arrows, such as in figure 2.2, ideally with tiny toy examples of the inputs and outputs of each box),

3. the advancement or the distinctive property of your solution, usually in a benchmark plot, or as a clear demonstration and comparison of your results.

3.5.2 Make the figures comprehensible

The figures should be easily comprehensible. Surprisingly, that requires you to follow some common "standards" in figure design and processing. People are often used to a certain form of the visualizations, and (unless you have a very good reason) deviating from the standard is going to make the comprehension much more complicated. The common standards include the following:

- caption everything correctly, place the caption at an expectable position

- systematically label the plots with 'main' titles (usually in boldface, above the plot), plot axes, axis units and ticks, and legends

- lay out the diagrams systematically, ideally follow a structure of a bottom-up tree, a left-to-right pipeline, a top-down layered architecture, or a center-to-borders mindmap

- use colors that convey the required information correctly

  Although many people carry some intuition for color use, achieving a really correct utilization of colors is often very hard without previous experience in color science and typesetting. Always remember that everyone perceives color hues differently, therefore the best distinction between the colors is done by varying lightness of the graphics elements (i.e., separating the data by dark vs. light) rather than by using hues (i.e., forcing people to guess which one of salmon and olive colors means "better"). Almost 10% of the population have their vision impaired by some form of color vision deficiency, most frequently by deuteranomaly that prevents interpretation of even the most 'obvious' hue differences, such as green vs. red. Finally, printed colors look surprisingly different from the on-screen

colors. You can prevent much of these problems by using standardized palettes and well-tested color gradients, such as the ones from ColorBrewer[2] and ViridisLite[3]. Check if your pictures still look good if converted to greyscale, and use a color deficiency simulator to check how the colors are perceived with deuteranomaly.

Avoid large areas of over-saturated and dark colors:

- under no circumstances use dark backgrounds for any graphical elements, such as diagram boxes and tables — use very light, slightly desaturated colors instead
- avoid using figures that contain lots of dark color (as a common example, heatmaps rendered with the 'magma' color palette often look like huge black slabs that are visible even through the paper sheet, thus making a dark smudge on the neighboring page)
- increase the brightness of any photos to match the average brightness of the text around the figure

Remember to test your figures on other people — usually, just asking 'What do you think the figure should show?' can help you debug many mistakes in your graphics. If they think that the figure says something different than what you planned, then most likely it is your figure what is wrong, not the understanding of others.

Finally, there are many magnificent resources that help you arrange your graphics correctly. The two books by Tufte [4, 5] are arguably classics in the area. Additionally, you may find many interesting resources to help you with technical aspects of plotting, such as the `ggplot`-style 'Fundamentals' book by Wilke [6], and a wonderful manual for the TikZ/PGF graphics system by Tantau [7] that will help you draw high-quality diagrams (like the one in figure 2.2).

3.6 What is a discussion?

After you present the results and show that your contributions work, it is important to *interpret* them, showing what they mean in the wider context of the thesis topic, for the researchers who work in the area, and for the more general public, such as for the users.

Separate discussion sections are therefore common in life sciences where some ambiguity in result interpretation is common, and the carefully developed intuition about the wider context is sometimes the only thing that the authors have.

[2] `https://colorbrewer2.org`

[3] `https://sjmgarnier.github.io/viridisLite/`

Exact sciences and mathematicians do not need to use the discussion sections as often. Despite of that, it is nice to position your output into the previously existing environment, answering:

- What is the potential application of the result?
- Does the result solve a problem that other people encountered?
- Did the results point to any new (surprising) facts?
- How (and why) is the approach you chose different from what the others have done previously?
- Why is the result important for your future work (or work of anyone other)?
- Can the results be used to replace (and improve) anything that is used currently?

If you do not know the answers, you may want to ask the supervisor. Also, do not worry if the discussion section is half-empty or completely pointless; you may remove it completely without much consequence. It is just a bachelor thesis, not a world-saving avenger thesis.

Conclusion

In the conclusion, you should summarize what was achieved by the thesis. In a few paragraphs, try to answer the following:

- Was the problem stated in the introduction solved? (Ideally include a list of successfully achieved goals.)
- What is the quality of the result? Is the problem solved for good and the mankind does not need to ever think about it again, or just partially improved upon? (Is the incompleteness caused by overwhelming problem complexity that would be out of thesis scope, or any theoretical reasons, such as computational hardness?)

  This is quite common.
- Does the result have any practical applications that improve upon something realistic?
- Is there any good future development or research direction that could further improve the results of this thesis? (This is often summarized in a separate subsection called 'Future work'.)

Bibliography

[1] Jianyi Yang, Ambrish Roy, and Yang Zhang. "Protein–ligand binding site recognition using complementary binding-specific substructure comparison and sequence profile alignment". In: *Bioinformatics* 29.20 (Aug. 2013), pp. 2588–2595. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btt447. eprint: https://academic.oup.com/bioinformatics/article-pdf/29/20/2588/48894365/bioinformatics_29_20_2588.pdf. URL: https://doi.org/10.1093/bioinformatics/btt447.

[2] Radoslav Krivák and David Hoksza. "P2Rank: machine learning based tool for rapid and accurate prediction of ligand binding sites from protein structure". In: *Journal of cheminformatics* 10 (2018), pp. 1–12.

[3] Lukas Jendele et al. "PrankWeb: a web server for ligand binding site prediction and visualization". In: *Nucleic acids research* 47.W1 (2019), W345–W349.

[4] Edward R Tufte, Nora Hillman Goeler, and Richard Benson. *Envisioning information*. Graphics press Cheshire, CT, 1990.

[5] Edward R Tufte. *Visual display of quantitative information*. Graphics press Cheshire, CT, 1983.

[6] Claus O Wilke. *Fundamentals of Data Visualization*. O'Reilly Media, Inc., 2019. ISBN: 9781492031086. URL: https://clauswilke.com/dataviz/.

[7] Till Tantau. *The TikZ and PGF Packages (Manual for version 3.1.8b)*. Tech. rep. Institut für Theoretische Informatik Universität zu Lübeck, 2020. URL: http://mirrors.ctan.org/graphics/pgf/base/doc/pgfmanual.pdf.

Appendix A

Using CoolThesisSoftware

Use this appendix to tell the readers (specifically the reviewer) how to use your software. A very reduced example follows; expand as necessary. Description of the program usage (e.g., how to process some example data) should be included as well.

To compile and run the software, you need dependencies XXX and YYY and a C compiler. On Debian-based Linux systems (such as Ubuntu), you may install these dependencies with APT:

```
apt-get install \
  libsuperdependency-dev \
  libanotherdependency-dev \
  build-essential
```

To unpack and compile the software, proceed as follows:

```
unzip coolsoft.zip
cd coolsoft
./configure
make
```

The program can be used as a C++ library, the simplest use is demonstrated in listing 1. A demonstration program that processes demonstration data is available in directory `demo/`, you can run the program on a demonstration dataset as follows:

```
cd demo/
./bin/cool_process_data data/demo1
```

After the program starts, control the data avenger with standard `WSAD` controls.

Listing 1 Example program.

```
#include <CoolSoft.h>
#include <iostream>

int main() {
  int i;
  if(i = cool::ProcessAllData()) // returns 0 on error
    std::cout << i << std::endl;
  else
    std::cerr << "error!" << std::endl;
  return 0;
}
```