

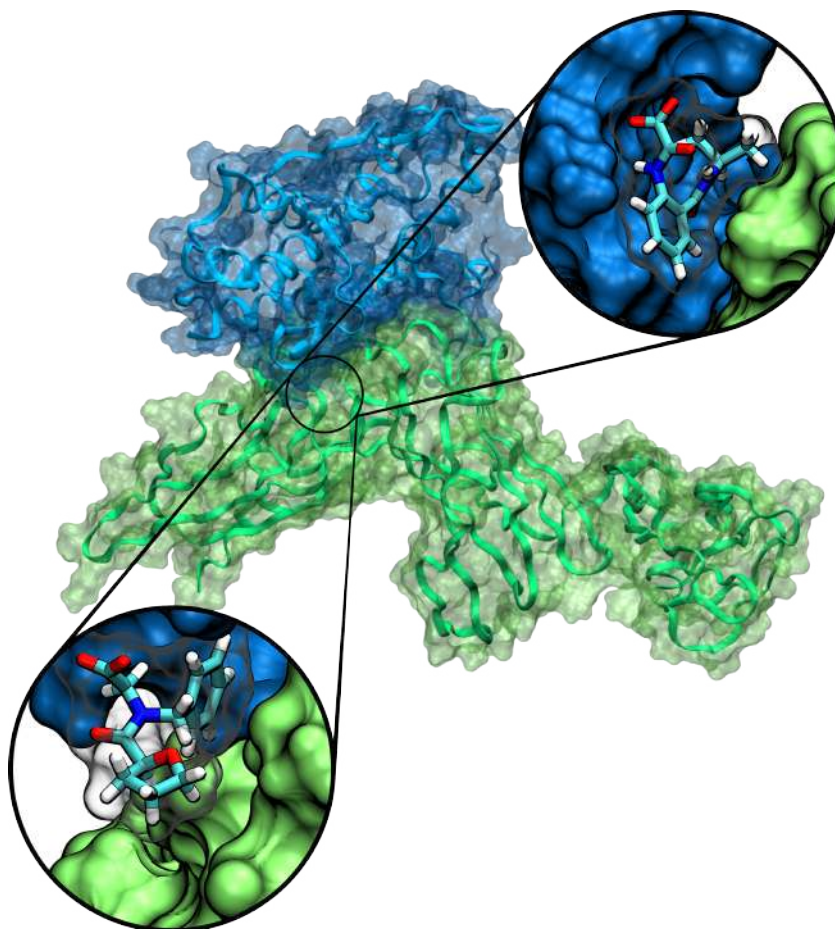
Dr. Shu-Yu Chen | Luis Vollmers | Prof. Martin Zacharias

## Molecular Dynamics (SS2025) Exercise 9

### A Demonstration on *in silico* Identification of a Protein-Protein Interaction Stabilizer

#### Report Tasks

1. Try to complete each of the steps of this exercise
2. At the end of this exercise, suggest 1-2 chemical compounds as potential protein-protein-interaction-stabilisers for the receptor complex at hand.
3. Discuss the suggested compounds and explain why these compounds seem promising. Use the plots and measurements conducted in the respective steps to support your reasoning.
4. In case the exercise could not be finished, describe any problems that you encountered as detailed as possible. Also document your troubleshooting attempts.



## Contents

<b>0</b>	<b>Software Installation</b>	<b>4</b>
0.1	Create Conda Environment and Install Packages . . . . .	4
0.2	Install MGLTools for Receptor Preparation . . . . .	5
0.3	Introduction to Jupyter and Environment Testing . . . . .	5
0.4	Installation of OpenBabel . . . . .	6
0.5	Installation of Autodock-GPU (optional) . . . . .	6
<b>1</b>	<b>Protein Construction</b>	<b>7</b>
1.1	Download the PDB structure and sequence . . . . .	7
1.1.1	Extracting the targeted proteins and solvent . . . . .	7
1.1.2	Extracting the targeted sequence . . . . .	8
1.2	Modelling the missing residues with MODELLER . . . . .	8
1.2.1	Sequence alignment . . . . .	8
1.2.2	Modelling the missing residues . . . . .	9
1.3	Visualization in Jupyter Notebook/Lab . . . . .	9
<b>2</b>	<b>Interface Pocket Detection</b>	<b>10</b>
2.1	Pocket detection with Fpocket . . . . .	10
2.2	Pocket analysis with Python . . . . .	11
<b>3</b>	<b>Ligand Preparation</b>	<b>13</b>
3.1	Query the ZINC20 Database and Download the Chemical Compounds. . . . .	13
3.2	Ligand feature extraction with RDkit . . . . .	14
3.3	Generating Ligand 3D Conformers in PDBQT Format . . . . .	16
<b>4</b>	<b>Molecular Docking</b>	<b>17</b>
4.1	Generating PDBQT files for Receptor . . . . .	17
4.2	Docking Test with AutoDock Vina . . . . .	18
4.3	Large-Scale Screening with AutoDock-GPU . . . . .	21
4.3.1	Setting up the docking grid of the receptor . . . . .	22
4.4	Docking with AutoDock-GPU . . . . .	22
4.4.1	Analysis and Visualisation of Docking Results from Autodock-GPU . . . . .	24
<b>5</b>	<b>Molecular Dynamics Simulation with GROMACS</b>	<b>27</b>
5.1	Ligand Paramterization and MD simulation . . . . .	27
<b>6</b>	<b>Binding free energy estimation with MM/PBSA</b>	<b>32</b>

## Introduction

Drug development is an expensive task in pharmaceutical industry and the involvement of computers in the drug development process is very beneficial both in terms of cost effectiveness and obtained molecular insight. In this tutorial, we will get to know an example workflow in computer-assisted drug design.

Typically, in drug design, one deals with a “receptor” protein (i.e. the drug target) and one wants to find a drug binding to this protein. The drug plays the role of a “ligand” that is supposed to bind to a given location of the protein, the so-called “binding pocket”. A binding pocket to which a drug can bind is called a “druggable” binding pocket. A good binding is

indicated by a negative binding free energy. Of course, in reality, a drug has to fulfil much more properties than just a favorable binding free energy. Most importantly, a drug must not be toxic, i.e. it must not have too serious side effects. That means, among many other things, for example, that the drug should not affect too strongly other important proteins in the body. Ideally, it should selectively bind to the given drug target and not influence other processes in the body which may give rise to undesired side effects. Also, it must be possible to administer the drug to the patient somehow in a feasible way, e.g. via an orally taken tablet, or via an intravenous injection. From a practical point of view, it should also be possible for chemists to synthesize the drug somehow, i.e. the drug must not be too complicated for standard synthesis procedures. However, we do not care about the latter points for now. The purpose of this tutorial is solely to introduce and illustrate standard computer tools in an example drug discovery process with the goal to find a drug that is able to bind in a favorable fashion to a given receptor system. Furthermore, the goal is not to gain an in depth understanding of every tool and every calculation, but rather to savour the incredible versatility of computational methods in biochemistry.

The workflow we will do in this tutorial is, for an example protein system:

- preparation of the protein system for subsequent computer-assisted tasks
- find a pocket in the protein to which a drug could bind, i.e. find a druggable pocket
- search through a database of possible compounds and find a compound that binds to the pocket – this is called a “virtual screen”; it helps to reduce the number of possible drug candidates
- from the given set of candidates, find a “best” one; this can be done by a procedure called “molecular docking”; molecular docking evaluates the possible poses of a ligand in the given binding pocket and ranks them according to favorable receptor-ligand interactions (this ranking is referred to as “scoring”); for example, a drug candidate which has many good interactions (e.g. hydrogen bonds) with the receptor has a high score
- for the best drug-receptor complex, a molecular dynamics simulation and binding free-energy calculation can be done to further assess the characteristics of the drug-receptor complex

What kind of drug are we looking for in this tutorial? We will look for a novel class of drugs, protein-protein interaction modulators. Protein-protein interactions have pivotal roles in biological processes. Perturbation of physiological protein-protein interactions is associated with a wide range of diseases. Therefore, targeting protein-protein interactions is a valuable therapeutic strategy. How do we want to modulate protein-protein interactions? In this case, we want to stabilize a protein-protein interaction. That means, for a given protein-protein complex, we want to find a drug that enhances the interaction between the two proteins. In other words, the two proteins play the role of the “receptor”. The drug binding pocket we want to find is supposed to lie somewhere at the interface of the two proteins such that the drug can optimally strengthen the interactions between the two proteins. We do not yet know how the drug will act in terms of detailed molecular interactions. But computer tools will help us to find a drug and to rationalize its effect on the relevant protein-protein interactions.

In principle, the tasks and tools applied in this tutorial can be applied to a wide range of drug development for interaction enhancement. For example, besides protein-protein interactions (PPI), enhancing the interaction between DNA and transcription factors can be also of significant therapeutic value.

## 0 Software Installation

### Working Directory: 00\_install\_prep/

The workflow of the identification of potential stabilizer is composed of many computational programs, many of which can be downloaded either by installers or via anaconda, which is a python-based package manager. This tutorial utilises anaconda for most purposes. For installation of anaconda, please visit: <https://docs.conda.io/projects/conda/en/latest/index.html> and follow the instructions.

### 0.1 Create Conda Environment and Install Packages

With anaconda installed, the next step is to create a python environment and install all relevant programs which are called packages in the anaconda context. Conveniently, there is a script provided `conda_env.sh`, which should create the environment and install all the packages automatically upon execution. Nevertheless, read this chapter carefully, before running `conda_env.sh` to avoid confusion and mistakes. This process can take a very long time, varying from a couple of minutes to a couple of hours and is not guaranteed to succeed. However, once all required packages are installed, the environment can be activated via `conda activate ppis`

```
conda activate ppis
```

**Always remember to activate the environment when opening a new shell.** Upon activation, (ppis) should appear at the very left side of your command line, at least that would be the standard location. The content and thus the packages installed by `conda_env.sh` are listed and summarised below.

```
conda create --no-default-packages -n ppis python=3.7 -y
conda install -n ppis -c rdkit rdkit -y
conda install -n ppis -c anaconda pandas numpy seaborn jupyter -y
conda install -n ppis -c conda-forge mplcursors ipympl -y
conda install -n ppis -c salilab modeller -y > modellerconfig.txt
conda install -n ppis -c conda-forge fpocket mdtraj nglview ambertools=23 -y
conda install -n ppis -c conda-forge jupyterlab jupyter_contrib_nbextensions -y
conda install -n ppis -c conda-forge -c schrodinger pymol-bundle -y
conda install -n ppis -c bioconda autodock-vina autogrid -y
conda install -n ppis -c conda-forge mdanalysis
```

- `pandas` [11][16], `numpy` [5], `seaborn` [15], `ipympl` [6], and `mplcursors` are basic python analysis tools for data handling and visualization.
- `jupyter` [7] enables an interactive coding interface. Common coding user interfaces are JupyterNotebook and JupyterLab. We also need `jupyter_contrib_nbextensions` to enable the table of content (TOC) extension.
- `modeller` [17] is a homology modeling package and is necessary if you have some parts of your protein missing. To use this package you need to register for a license for free at <https://salilab.org/modeller/> and activate the license by replacing XXXX in the license file `config.py`. The location of the file varies depending on system, conda installation, etc. However, it should be saved in the conda environment folder in the `ppis` environment. E.g. the path could be `/anaconda3/envs/ppis/lib/modeller-10.4/modlib/modeller/config.py`. Technically, after executing `conda_env.sh` there should be a file called `modellerconfig.txt` in the same directory, which contains the location of `config.py`.

- `rdkit` is important for compound filtering and visualization. Visit <https://www.rdkit.org/> for more information.
- `fpocket` [8] is a pocket detection program that quickly screens through the surface of your protein complex and reveals druggable pockets. Visit <https://www.ddl.unimi.it/manual/pdf/Fpocket.pdf> for more information.
- `openbabel` [10] allows us to transform structure files between PDBQT (format for docking), PDB (format for reading and analysis), and MOL2 (format for ligand parameterization). Visit [http://openbabel.org/wiki/Main\\_Page](http://openbabel.org/wiki/Main_Page) for more information.
- `mdtraj` [9] analyzes the surface area and load the biomolecular structures/trajectories to be visualized using `nglview`. Visit <https://www.mdtraj.org/1.9.8.dev0/index.html> and <https://github.com/nglviewer/nglview> for more information.
- `ambertools` [2] will be needed to parameterize the ligand and generate the simulation box for MD simulations and MM/GBSA calculation. Visit <https://ambermd.org/> for more information.
- `pymol` [14] is a common visualization tool.
- `autodock-vina` [3] and `autogrid` will be used for docking. For large-scale docking, Vina-GPU (<https://github.com/DeltaGroupNJUPT/Vina-GPU>) or Autodock-GPU (<https://github.com/ccsb-scripps/AutoDock-GPU>) is recommended. Herein, Autodock-GPU is used. Visit <https://autodock-vina.readthedocs.io/en/latest/> for more information. For Windows user, the `conda install autodock-vina` will not work, please provide a Vina installation through <https://vina.scripps.edu/downloads/>.

## 0.2 Install MGLTools for Receptor Preparation

MGLTools[12] is a user-friendly graphical interface for structural visualization and docking preparation. It can be downloaded from <https://ccsb.scripps.edu/mgltools/> and simplifies visualisation and structure preparation by a margin. After you have downloaded MGLtools, you can execute it by typing `pmv` in your terminal. Especially AutoDock which relies on PDBQT and GPF files benefits greatly from MGLTools' graphical interface. A very easy installation instruction for Ubuntu can be found on <https://dev.to/abdnahid/how-to-install-gromacs-pymol-autodock-vina-vmd-mgltools-avogadro-open-babel-in-ubuntu-2004-1867>. Notice, that command that is referenced on this web site `source ./initMGLtools.sh` must be executed for every newly opened terminal window, together with the correct file path to `initMGLtools.sh`.

For any other operating system, a quick google search might be helpful.


## 0.3 Introduction to Jupyter and Environment Testing








The major numerical analysis of structure, docking score and trajectories are performed in the jupyter notebook/lab editor. We can open a jupyter notebook/lab by opening a new terminal tab by `Ctrl` + `T` or `Ctrl` + `Alt` + `T`. Remember to activate the ppis conda environment with

```
conda activate ppis
```

In the current working directory there should be a file called `TOY.ipynb` which is the standard jupyter format. Instead of opening it with a command, we open the jupyter interface with either the notebook or the lab command shown below.

```
jupyter notebook # For notebook user
jupyter lab # For lab user
```

The interface should open in the default web browser and it should show the file contents of the current directory. Clicking on the `TOY.ipynb` in the jupyter interface should open up an IPython (interactive python) editor. The script should be somewhat commented with some explanation. Some useful shortcuts in jupyter notebook/lab are listed below (press  first to escape from the scripting mode).

-  +  : Run cell
-  +  : Delete cell
-  : Append a cell above
-  : Append a cell below
-  : Show all shortcut command

If everything got installed correctly, all commands in `TOY.ipynb` should execute normally. Only the last command might cause trouble, but a solution can be found in the respective jupyter cell in `TOY.ipynb`.

## 0.4 Installation of OpenBabel

OpenBabel[10] (<https://openbabel.org/>) is a powerful C++-based tool to convert ligand files between different formats in a very straight-forward manner. Although an online web server of OpenBabel is also available, installing it on your local machine can significantly simplify the operation and automation. Therefore, an installation is recommended if you plan on more ligand screening in the future. Only utilising the web server will result in testing around 9 different chemical compounds later on in the tutorial. The installation can either be completed by compiling from source or by installing it from the software repository of the respective operating system. E.g. in Ubuntu OpenBabel can be installed by using `apt-get` (with sudo rights).

```
sudo apt-get install openbabel
```

For a different operating system, a quick google search might offer some help.

## 0.5 Installation of Autodock-GPU (optional)

Note that a CUDA installation is a prerequisite for Autodock-GPU[13]. Instructions for both installations can be found here <https://github.com/ccsb-scripps/AutoDock-GPU/wiki/Guide-line-for-users>. The benefit of GPU-acceleration cannot be understated. A GPU resource is able to cut the docking time by a factor of 10 via AutoDock-GPU[13]. For careful installation, please visit <https://github.com/ccsb-scripps/AutoDock-GPU> for detail. For those who do not want to read the github site a compilation command that worked once is shown below, however, it might throw you lots of errors. Nonetheless, for consistency the repository should be cloned in the `PPIS_tutorial/` directory, thus the same directory that contains `00_install_prep`, `code01_receptor_prep` and so on. In the following block of code, this is implied by the first line.

```
cd path/to/PPIS_tutorial/
git clone https://github.com/ccsb-scripps/AutoDock-GPU
export GPU_INCLUDE_PATH=/usr/local/cuda/include
```



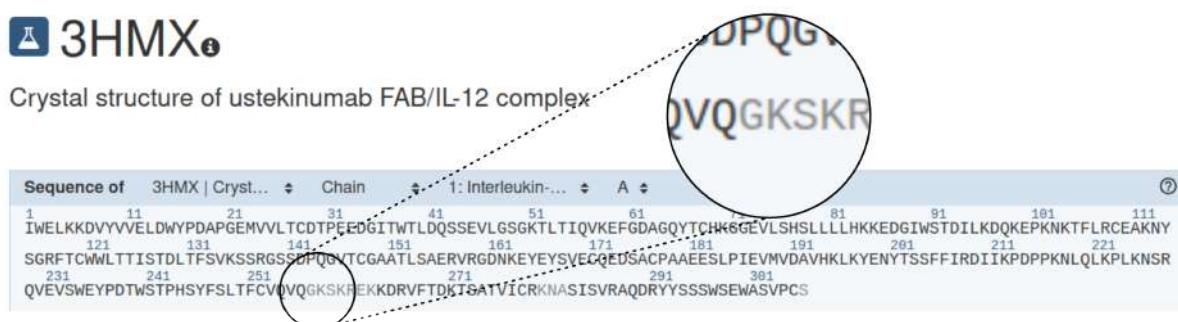
```
export GPU_LIBRARY_PATH=/usr/local/cuda/lib64
cd AutoDock-GPU
make DEVICE=GPU NUMWI=64
```

Unfortunately, AutoDock-GPU only supports CentOS, Ubuntu, and macOS system. Windows users might need to compile Vina-GPU<sup>1</sup> or Vina-GPU+<sup>2</sup>. Notice, that if Autodock-GPU is not being installed, the virtual screening cannot be conducted. Instead, the docking and the following steps will then need to be conducted with only a few test compounds, which are introduced in section 4.2.

## 1 Protein Construction

### Working Location: 01\_receptor\_prep

In the demonstration, we will use the protein-protein complex of interleukin (IL) as an example. The structure of the IL-12 $\alpha$ /IL-12 $\beta$ -complex can be visualized at <https://www.rcsb.org/3d-view/3HMX/1>. As shown in Figure 1, there are some residues shown in gray instead of solid black. This indicates that these residues are not resolved in the X-ray experiment. If we change the sequence display from **Interleukin-12 subunit alpha** to **Interleukin-12 subunit beta**, we can also see there are a lot of missing residues. Therefore, the first step is to model the missing residues back. This is a useful practice since missing residues are encountered frequently in the PDB.



**Figure 1:** Webpage of RCSB with PDB code 3HMX. The missing residues of Interleukin-12 subunit alpha are light grey instead of black.

### 1.1 Download the PDB structure and sequence

#### 1.1.1 Extracting the targeted proteins and solvent

To start with, we need to download the structure and the sequence from the website to our folder 01\_Download/. The structure can be downloaded in a PDB format and the sequence can be downloaded in a fasta format. For convenience and consistency these files should be save as 3hmx.fasta and 3hmx.pdb. Since our task is the target the interface between Interleukin-12 subunit alpha (chain A) and Interleukin-12 subunit beta (chain B), we can extract these two chains using grep and regular expression to hmx.pdb (see below). The PDB format is a well defined and standard format for protein coordinates which can be studied elsewhere, since a detailed description is out of scope for this tutorial.

```
grep -E 'ATOM\s+[0-9]+\s+[A-Z]+\s+[A-Z]+\s+A' 3hmx.pdb > hmx.pdb
echo "TER" >> hmx.pdb
grep -E 'ATOM\s+[0-9]+\s+[A-Z]+\s+[A-Z]+\s+B' 3hmx.pdb >> hmx.pdb
```

<sup>1</sup><https://github.com/DeltaGroupNJUPT/Vina-GPU>

<sup>2</sup><https://github.com/DeltaGroupNJUPT/Vina-GPU-2.0>

### 1.1.2 Extracting the targeted sequence

We also need to extract the sequence of chain A and chain B to a new file `hmx.fasta`. This needs to be in a special format to be read by MODELLER in the next step. The content of the file should look like the example below.

```
>P1;hmx
sequence:hmx:::::0.00:0.00
<CHAIN-A-SEQUENCE>/
<CHAIN-B-SEQUENCE>*
```

The text `hmx` is the align code used for the next step. You can find the sequences of chain A and chain B in the `3hmx.fasta` file you downloaded from RCSB and paste it to the file. The `/` symbol in the file indicates the end of a chain and a start of the next chain and the `*` symbol indicates the end of the whole protein sequence(s).

## 1.2 Modelling the missing residues with MODELLER

The homology modeling software MODELLER[4] takes the sequence alignment file and the structure template(s) to model a structure. Modelling the missing residues consists of two steps: sequence alignment and homology modeling.

### 1.2.1 Sequence alignment

Sequence alignment in our case is only to find out the position of the missing residues in the structural file. This can be done more efficiently by using a python script `salign.py` whose content is depicted below.

```
from modeller import *

env = Environ()
aln = Alignment(env)
mdl = Model(env, file='hmx')
aln.append_model(mdl, align_codes='hmx_structure',
                atom_files='hmx.pdb')
aln.append(file='hmx.fasta', align_codes='hmx')
aln.salign()
aln.write(file='hmx.ali', alignment_format='PIR')
```

This python script reads the structure from the path `./hmx.pdb` with an align code `hmx_structure` and align it with the sequence from `./hmx.fasta` with an align code `hmx`. We can run the script by the following command in the PPIS conda environment:

```
modl0.4 salign.py # you can ignore the 'import site' failed message
```

The sequence alignment output is written to `hmx.ali` and consists of the original `fasta` string and the aligned string that contains gap insertion symbols `-`. The gap insertion symbol communicates missing residues to `model.py` in the next step and therefore any `-` must be aligned on a missing residue. However, the `-` placement can be erroneous and usually we need to manually modify this file so that the missing residues from the `hmx_structure` sequence are all replaced by the dash sign. The missing residues can be looked up at the PDB web site <https://www.rcsb.org/3d-view/3HMX/1> where they are printed in light grey instead of black as mentioned before. Each of the grey letters should be replaced with a `-` in `hmx.ali`. From experience however, `hmx.ali` contains a mistake at the boundary of the two sequences `S/RNLPVATPDGMFP`. Technically, the gap insertion by `salign.py` should result in `-/-----MFP`, but usually, the first dash and the `M` is misplaced requiring a manual fix. Thus, the correctness of `modeller` usually requires quality control.



### 1.2.2 Modelling the missing residues

With a corrected sequence alignment file `hmx.ali`, the program `model.py` can be used to construct the coordinate of the missing residues. The contents of the program are shown below.

```
from modeller import *
from modeller.automodel import *

env = Environ()
a = automodel(env, alnfile='hmx.ali',
              knowns='hmx_structure', sequence='hmx',
              assess_methods=(assess.DOPE))
a.starting_model = 1
a.ending_model = 5
a.make()
```

Essentially, we are generating the model using the `automodel` module from `modeller` with the DOPE scoring method. The python script can be executed in the same fashion as for `salig.py`.

```
mod10.4 model.py
```

This generates 5 models, as specified by `a.ending_model = 5`. A good thing about `modeller` is that it uses a simulated annealing method to generate the structure, obeying the following process: minimization → heat-up → cool-down → minimization. We can also track the energy during the process of the first model by

```
vim hmx.D00000001
```

and check the final energy of all generated structures by

```
tail -n 10 hmx.D*
```

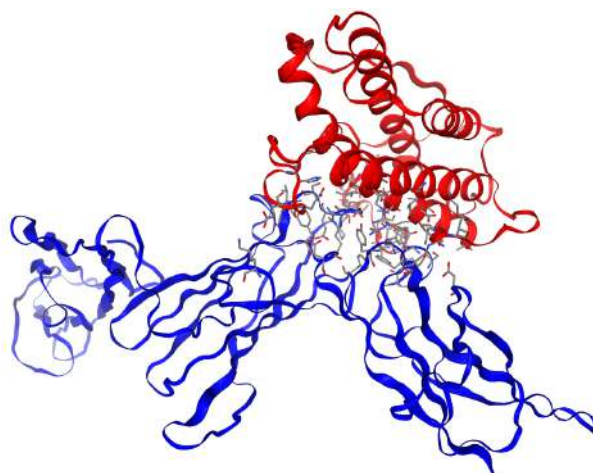
However, the structure assessment is not done only by the energy but also by the DOPE score, as it was specified in `model.py`. The DOPE score can be checked by looking into the log file.

```
tail -n 10 model.log
```

In some cases, the first structure is not the best one. In such cases we might need to generate either more models or even include restraints. For more detail of module `automodel`, refer to <https://salilab.org/modeller/10.4/manual/node44.html>. Herein, `hmx.B00000001.pdb` serves as the target structure.

### 1.3 Visualization in Jupyter Notebook/Lab

In the current directory there is a jupyter notebook named `01_interface_visualisation.ipynb`. Open the jupyter notebook as described in section 0.3 and execute the cells to visualise the structure. The notebook reads the prepared receptor file and calculates the interface residues via an intermolecular, interatomic cut-off distance of 5 Å. Upon successful execution, the visualisation should appear similar to figure 2. Each mouse key has a different control element to it. While the left button rotates the structure, the right button translates it. Residues or atoms can be centered in the visualization by simply left-clicking on a particle of choice. Zoom in by scrolling up with the mouse wheel in the visualization panel.



**Figure 2:** Visualization of the residue ID and the protein complex. The interface residues of protein A (blue cartoon) and protein B (red cartoon) are shown in the licorice representation.

Of course, the jupyter notebook is set for this case study and is not generally applicable to any receptor complex. For instance, if any system of interest contains more than 2 proteins, such as a ternary DNA-protein complex, residue numbers might need to be changed as the visualisation commands.

## 2 Interface Pocket Detection

### Working Location: 01\_receptor\_prep

Now, we already have a complete protein-protein (PP) complex structure and could potentially submit to ligand docking. However, performing molecular docking without specifying the whole box is not applicable for the identification of PPI stabilizer. A good PPI stabilizer needs to bind at the interface with a relatively even contact surface with both sides of the complex. For this, we can make use of Fpocket[8] to first detect a druggable pocket at the PP surface.

### 2.1 Pocket detection with Fpocket

Fpocket is a handy program and straightforward program. After activating the conda environment, it can be executed with the respective PDB file as an option to the `-f` flag. The command itself and its appropriate output are shown below.

```
fpacket -f hmx.B99990001.pdb
***** POCKET HUNTING BEGINS *****
***** POCKET HUNTING ENDS *****
```

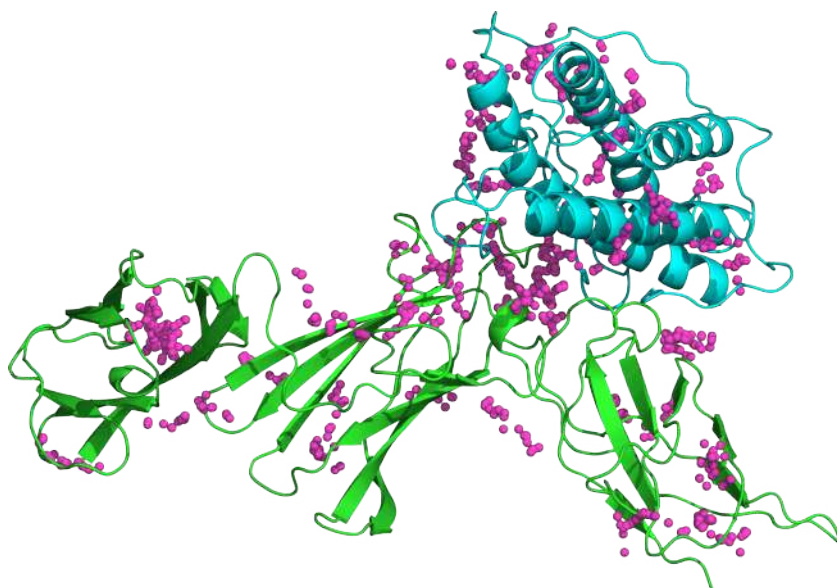
This generates a folder `hmx.B99990001_out/`. After changing the directory the pockets can be visualized by executing a custom bash script.

```
cd hmx.B99990001_out/
bash hmx.B99990001_PYMOL.sh
```

Executing said bash script should open a PyMol window with the two protein chains and the pockets that were found by fpocket (see figure 3. In the `hmx.B99990001_out/pockets/` folder, further information can be found about all detected pockets. The residues that form the pocket are written to `pocket*_atm.pdb` and the position of the probes are written in `pocket*_vert.pqr`. You can check several pocket properties of the first pocket calculated by

`fpocket` using the following line of code. The file path assumed that the current directory is `01_receptor_prep`.

```
head hmx.B99990001_out/pockets/pocket1_atm.pdb -n 20
```



**Figure 3:** All pockets found by `fpocket` and visualised by `pymol`. This is the output obtained from the command shown above. The coloring is slightly altered from the default, so that one chain is shown in green, the other one is shown in cyan and the pockets are shown in magenta.

## 2.2 Pocket analysis with Python

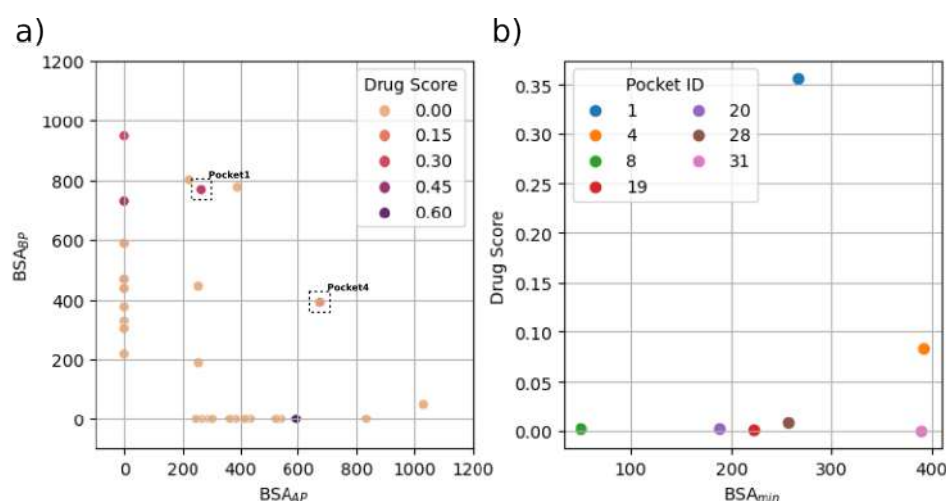
`fpocket` basically yields pockets that are filtered by geometrical criteria but also by a scoring method, that evaluates how well drug-like molecules would fit in the respective pocket.[8] `fpocket` calculates the binding pocket by: (1) detect alpha spheres, i.e. a dummy sphere that intersects with exactly four atoms, (2) Clustering the spheres into individual pockets and (3) ranking the spheres based on descriptor-based pocket scoring and drug scoring methods. However, information about a possible protein-protein-interface are not present and therefore, a pocket that forms contacts with both proteins must be identified from the output of `fpocket`. Furthermore, some interfacial pocket might not resembles a good drug site which would then be excluded from further screening.

Technically, the output files generated by `fpocket` contain plenty information to evaluate the goodness of each pocket, however, they are very thorough and virtually impossible to analyse manually. Therefore, another jupyter notebook called `02_pocket_analysis.ipynb` can be found in the current directory which takes care of the analysis. Executing every cell should calculate the solvent-accessible surface area (SASA) and buried surface area (BSA) of the two proteins and each pocket respectively. Here, we treat each pocket probes as an artificial ligand. E.g. the BSA of pocket  $P$  within protein A is obtained by summing up the SASA of protein A and the pocket ( $SASA_A$ ,  $SASA_P$ ) and then subtract the  $SASA_{AP}$  of the complex of the two.

$$BSA_{PA} = (SASA_A + SASA_P) - SASA_{AP} \quad (1)$$

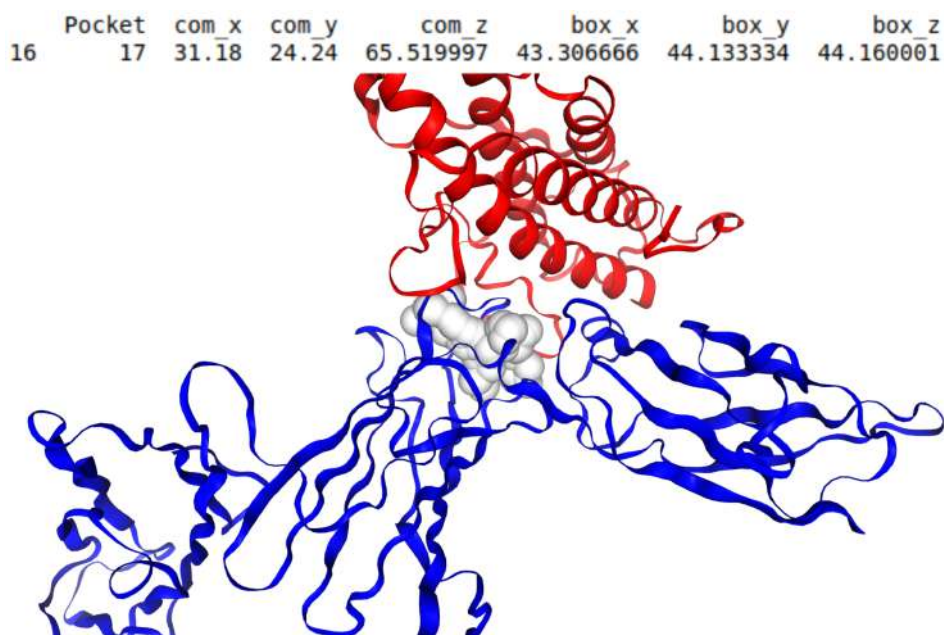
The BSA of protein B and the pocket is evaluated in the same fashion in the jupyter notebook. A criteria for a good stabiliser binding pocket is a large BSA with both proteins, and a good pocket score and/or drug score. If all lines in the jupyter notebook are executed

successfully, we should obtain plots similar to figure 4 but not identical, since every pocket detection yields slightly different results. Herein, the best pockets can be identified as pocket 1 and pocket 4 since both have a high  $BSA_{min}$  and a high drug score. A closer look into figure 4 reveals that the BSA values for pocket 4 are more balanced which is considered a significant criterion for PPI-stabilisation. The pocket score is also reported by `fpocket` and can be used to settle ties, if two pockets are scoring equally well. It is featured in the jupyter notebook as well and should be plotted if everything executes successfully. For this tutorial it is sufficient to chose one pocket for closer examination, or maybe two. Notice, that `02_pocket_analysis.ipynb` also creates a file called `config` which should specify the best available pocket. Of course, this is is subject to each user's intuition. It is possible to create more than one config by changing the code, however, the filename `config` is hard coded in a later step in the executable `03_docking_prep/vina.sh` and thus would also need to be changed.



**Figure 4:** Data visualization of the protein-pocket BSA and drug score. **a)** Distribution of  $BSA_{AP}$  and  $BSA_{BP}$  of all detected pockets with color coded drug score. **b)** Distribution of  $BSA_{min}$  and Drug score of the interface pockets.

The jupyter notebook also contains cells for visualisation of the pockets for a visual inspection of eventual candidates. The result should look somewhat similar to figure 5. For the actual ligand docking the information of box center and box dimensions are necessary. These information are calculated by the jupyter notebook and are printed together with the 3D structure visualisation. How to use these specifically is further elucidated in section 4.



**Figure 5:** Structure visualization with a detected pocket. The center of mass (COM) and box size (SIZE) information are also printed on the top of the graphics. Feel free to visualize other pockets by changing `P_id` to other numbers in the notebook.

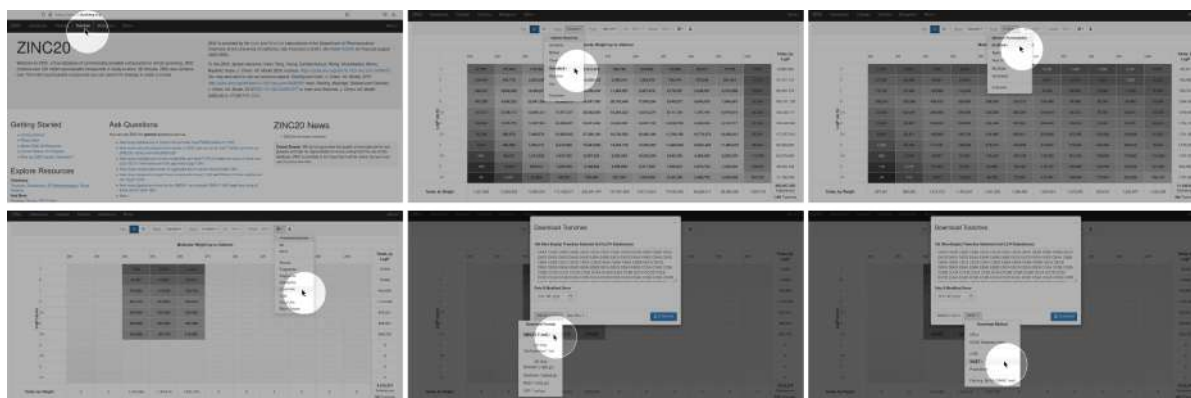
### 3 Ligand Preparation

Working location: `02_ligand_prep`

Having the docking box of the receptor prepared, the next step now is to prepare the ligands for docking. In theory, there are around  $10^{60}$  'drug-like' compounds, which is impossible to screen through with current hardware technology. Instead, we want to use the ZINC database (<https://zinc20.docking.org/>) in this demonstration. This database contains 0.8 billion compounds, which is still not feasible to screen within this tutorial which is why some filters will be applied to further reduce the amount of compounds for docking.

#### 3.1 Query the ZINC20 Database and Download the Chemical Compounds.

There are two stages of filtering that are applied in this tutorial. The first one is integrated into the query system of the ZINC database itself and therefore is relatively static. The second stage of filtering is a custom program, that in theory can be adapted to fit any scientific criteria and utilises the RDkit implementation of Python. The ZINC data base can be queried by opening the aforementioned link. The website should look similar to the one shown in figure 6. The process of downloading a tiny fraction of the database's compounds is illustrated in the same figure, although the 'tiny fraction' still engulfs around 4 million compounds.



**Figure 6:** In these six pictures the workflow to narrow down compounds from the ZINC database is shown. First access the website and click the Tranches button. Then filter the large number of compounds with some custom selections, shown in the second, third and fourth picture. Finally, the compounds need to be downloaded in SMILES format and downloaded as a wget file. Pictures as of June 2023.

### 3.2 Ligand feature extraction with RDkit

#### Working location: Step2\_Ligand\_preparation

The compounds are downloaded in the SMILES string format, which is generally computer readable and can be processed with various programs, e.g. RDkit and OpenBabel. The way SMILES are generated and work can be read into elsewhere<sup>3</sup>. The file that was downloaded in the wget format is not a data file per se but an executable which will then download the respective files for the user. After moving this file from the Downloads directory to the current directory, it can be executed via `bash`

```
mv ~/Downloads/ZINC-downloader-2D-smi.wget ./
bash ZINC-downloader-2D-smi.wget
```

This command should start the download of several different files that contain the compounds filtered from ZINC20 in section 3.1. For consistency a new directory should be created in `02.ligand_pre` called `SMILES/`. With the following commands, the compounds-containing SMI files can be transferred into `SMILES/`, and the then empty directories are being removed.

```
mkdir -p SMILES/
mv -v */*.smi SMILES/
rmdir */*
```

After moving all SMI files into the SMILES directory, the index row can be deleted via `sed`.

```
for i in SMILES/*.smi; do sed -i '/smiles/d' $i; done
```

Another option would be to remove the first line of each file, however, this is not safe for multiple executions.

The last steps in compound filtering include the execution of the python script, as well as concatenating all filtered compounds into one file. The latter should not precede the former, since executing the python script on a large SMILES list can lead to memory overflow. The contents of the Python program are shown below. It reads in a text file containing SMILES strings in each row and applies certain filter criteria to the compounds. The criteria chosen in `filter_smiles.py` are arbitrary and their only purpose is to show how to filter compounds in a database given prior pharmaceutical knowledge apart of decreasing the search space. Herein,

<sup>3</sup><https://medium.com/@luis-vollmers/tutorial-to-smiles-and-canonical-smiles-explained-with-examples-fbc8a46ca29f>



the filters encompass a limit of four hydrogen bond donors and acceptors, the presence of one carboxy group and one benzene or imidazole group. Furthermore, the last filter criterion is a Bertz complexity below 500. The Bertz complexity index measures the complexity of the molecular graph. These filters are implemented in RDkit and students may alter them in any desired way.

```
import sys
import numpy as np
import pandas as pd
from rdkit import Chem
from rdkit.Chem import Fragments
from rdkit.Chem.GraphDescriptors import BertzCT

in_file = sys.argv[1] # unfiltered .txt
out_file = sys.argv[2] # filtered .txt

# Read input smiles
smiles = pd.read_csv(in_file,header=None,
dtype=str,sep=' ').to_numpy(dtype=str)[:,-1]
if 'smiles' in smiles:
    smiles = np.delete(smiles,0)

# Filtering
## Unwanted Atoms
smi = [m for m in smiles if 'Si' not in m and '+' not in m and 'B' not in m and '-' not in m and
'I' not in m]

## Feature:
## exactly one COO-, Hbond donor and acceptor < 4
## wants exactly 1 imidazol or benzene group
## BertzCT complexity < 500

ms = [Chem.MolFromSmiles(m) for m in smi]
ms = [x for x in ms if Fragments.fr_COO(x) == 1]
ms = [x for x in ms if Chem.rdMolDescriptors.CalcNumHBD(x) < 4]
ms = [x for x in ms if Chem.rdMolDescriptors.CalcNumHBA(x) < 4]
ms = [x for x in ms if Fragments.fr_imidazole(x) == 1 or Fragments.fr_benzene(x)==1]
ms = [x for x in ms if BertzCT(x) < 500 ]
sub_smiles = [ Chem.rdMolfiles.MolToSmiles(x) for x in ms ]
# Output
np.savetxt(out_file,sub_smiles,fmt='%s')
```

The python program can be executed by simply calling `python` on `filter_smiles.py` repeatedly on each file. It should run for a short time, and yield one filtered file in `SMILES/` for each unfiltered file. The following command uses a for loop to iterate through all files in `SMILES/`, prints the filename and executes the python script with the output name being adjusted automatically.

```
for i in SMILES/*.smi; do echo $i; python filter_smiles.py $i ${i/.smi/_filtered.txt}; done
```

The concatenation of filtered compounds can be achieved via the `cat` command, which is short for concatenate.

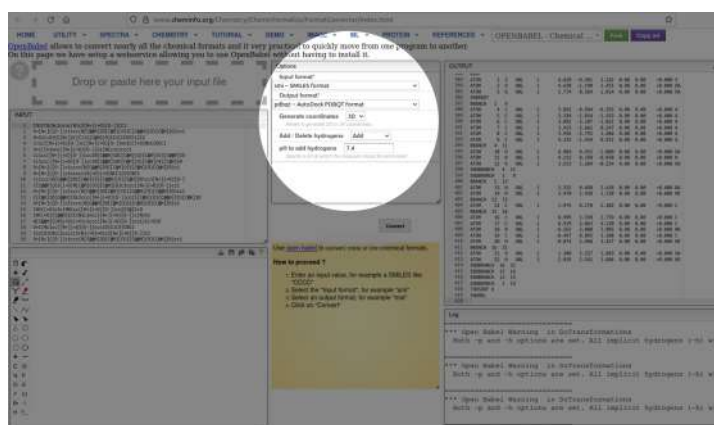
```
cat SMILES/*_filtered.txt > filtered.txt
```

The successful filtering can be assessed, by printing out the line number of the output file, which should contain less lines than originally downloaded compounds. In this tutorial, the number of downloaded compounds was around 4 million and the number of filtered compounds should be roughly 13936 compounds. The command that can be used to do so is called `wc` which is short for word count.

wc -l filtered.txt

### 3.3 Generating Ligand 3D Conformers in PDBQT Format

After filtering the compounds the next step is to generate three dimensional molecular structures also called conformers. For this task, OpenBabel is required. However, as stated in section 0.4 the local installation of OpenBabel is not mandatory, since there is also a web service which has a limit of nine structures at a time. Technically, nine structures are a bit less for a proper screening, however it suffices for this tutorial. More ambitious students are inclined to install OpenBabel for fully automated high throughput compound screening. For using the web service, visit <http://www.cheminfo.org/Chemistry/Cheminformatics/FormatConverter/index.html>. The interface should look like figure 7. The settings should be chosen as seen in this graphic.



**Figure 7:** Snapshot of the web page that offers the OpenBabel utility. The input specifications are highlighted: the format is SMILES, the output is PDBQT with three dimensional coordinates and hydrogens added at a pH of 7.4.

For the tutorial, nine randomly chosen SMILES strings can be converted to PDBQT format in the web interface. For consistency the name for the downloaded PDBQT file should be `filtered.pdbqt`.

In case of a successful installation of OpenBabel the following line of code should write the desired compounds to `filtered.pdbqt`. Notice, that the conversion of all 13936 compounds can take quite a lot of time, so a subset should be considered. E.g. the command can be interrupted after the conversion of around 2000 compounds, by pressing `Strg` + `C` on the keyboard.

```
obabel -ismi filtered.txt -opdbqt -O filtered.pdbqt --gen3d --minimize --steps 1500 --sd --ff
MMFF94
```

Sometimes OpenBabel will fail to generate a 3D structure from SMILES which results in an error, that can be ignored. There will be more than enough compounds, that result in a successful conversion.

In any case it is necessary to split the composite PDBQT files into individual ligand files, so that they can be docked individually later on. For this, create a folder in `02_ligand_prep/` called `PDBQT` and execute `vina_split` with the options shown below.

```
mkdir -p PDBQT
cd PDBQT
vina_split --input ../filtered.pdbqt --ligand ligand_
```

This should result in a directory that contains one PDBQT file for each ligand with a numeric ID, e.g. `ligand_0354.pdbqt`. These PDBQT files are the foundation for the virtual screening later on in this tutorial.

## 4 Molecular Docking

### Working Location: 03\_docking\_prep

With the receptor and the ligand candidates at hand, we can finally try to dock the ligand into the interface pocket that was chosen in section 2.2.

### 4.1 Generating PDBQT files for Receptor

Since the docking protocol of AutoDock Vina works with the PDBQT format, the receptor file needs to be converted before any docking can take place. For those students who did not install MGLTools, OpenBabel also offers utilities to convert from PDB to PDBQT. Although, MGLTools performs better and OpenBabel might produce artifacts and errors within the file that needs to be fixed in post-processing. Nevertheless, the command for converting via OpenBabel can be found below.

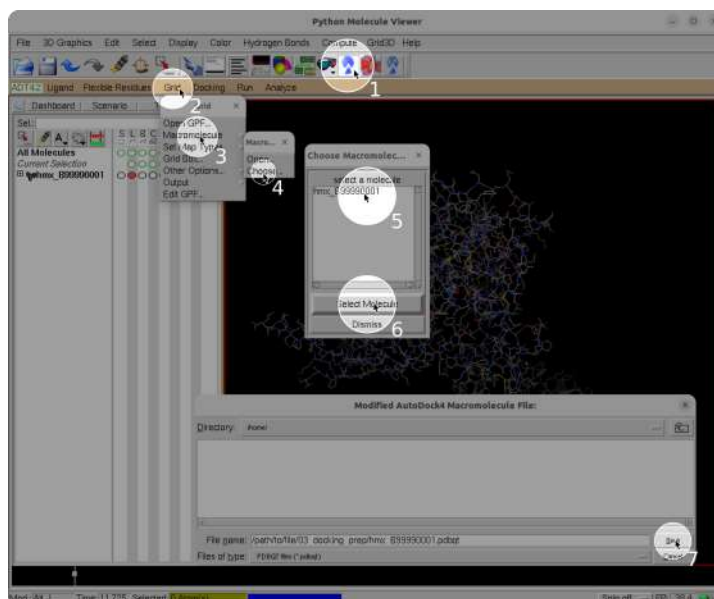
```
obabel -ipdb ../01*/hmx.B99990001.pdb -opdbqt -O hmx_B99990001.pdbqt -p 7.4 -xr
```

After successful conversion with OpenBabel, compare the new file with the original PDB file. Especially check for correct placement of the `TER` keyword and check for unrecognized residues, which are usually indicated by the `'UNK'` string.

The usage of MGLTools is a bit less troublesome in regard to postprocessing and offers a graphical user interface which can be opened by typing `pmv` in the command line.

```
pmv
```

First, the receptor molecule needs to be loaded by clicking **File** → **Read Molecule** at the upper left corner of the GUI. The path to the receptor molecule is `01_receptor_prep/hmx.B99990001.pdb`. In order to save the receptor in PDBQT format, follow the instructions shown in figure 8.



**Figure 8:** Generating PDBQT for the receptor using MGLTools. First, click the AutoDock Vina tool, then click on the newly opened Grid button and select Macromolecule and Chose.... Thereafter, a new window should open with only one molecule to select from which should then be saved into directory `03_docking_prep/hmx.B99990001.pdbqt`.

It is good practice to visualise the binding pocket to check if it matches with the findings in `01_receptor_prep/`. In this GUI, in the Grid toolbar shown in Figure 8 Grid Box can be found. By clicking this button a panel of box size parameters will open. The parameters should be adjusted according to `config` in the current directory to see if the box matches with the preferable binding pocket according to section 2.2. For better visualization, the ribbon representation of the protein can be activated by clicking the circle under the R column on the left panel shown in Figure 8. At the same time deactivating the circle under the L column removes the atomic licorice representation. Additionally, each molecule of the complex can be colored individually by clicking the triangle below the C1 column and choose the coloring scheme By Chain. If everything seems fine, MGLTools can be closed.

## 4.2 Docking Test with AutoDock Vina

With the PDBQT files for receptor and for ligand ready, a test should be performed to check whether everything is correctly setup, before screening a large set of compounds with GPU-accelerated docking. Apart of a ligand and receptor file, AutoDock Vina also requires a config file, which specifies the size and position of the docking box. Luckily, `config` is already prepared upon the successful execution of the jupyter notebook `02_pocket_analysis.ipynb`. The content of `config` should look something like shown below only that the numbers should match the preferred pocket chosen in `02.pocket_analysis.ipynb`.

```
center_x = 27.83
center_y = 38.78
center_z = 56.71
size_x = 40
size_y = 32
size_z = 40
```

For the aforementioned test a bash script called `vina.sh` is already placed in the current directory for automatic docking of the test ligands. The contents of this bash script are shown

below.

```
#!/bin/bash
receptor='./hmx_B99990001.pdbqt'
TEST='./02*/TEST'
PDBQT='./PDBQT'
PDB='./PDB'

mkdir -p -v $PDBQT $PDB

# Loop through all ligands
for file in $TEST/*
do
    # Fetch ligand basename
    name="${file##*/}"
    lig="${name%.pdbqt}"
    # Docking with Autodock vina
    vina --config config \
        --receptor $receptor \
        --ligand $TEST/$lig.pdbqt \
        --out $PDBQT/out_${lig}.pdbqt

    # Transforming and splitting the PDBQT output to PDB format
    obabel -ipdbqt $PDBQT/out_${lig}.pdbqt \
        -opdb -m -O $PDB/out_${lig}.pdb
done
# Convert pdbqt to pdb via http://www.cheminfo.org/Chemistry/Cheminformatics/FormatConverter/
# index.
html
```

This bash script should be executed and will dock the ligands in the binding pocket that is specified in `config`. The ligands are read from `./02.ligand_prep/TEST/` and the outputs are save to `PDBQT/` and `PDB/`. The command for executing the bash script and the expected output are shown below.

`./vina.sh`

```
#####
# If you used AutoDock Vina in your work, please cite: #
# #
# O. Trott, A. J. Olson, #
# AutoDock Vina: improving the speed and accuracy of docking #
# with a new scoring function, efficient optimization and #
# multithreading, Journal of Computational Chemistry 31 (2010) #
# 455-461 #
# #
# DOI 10.1002/jcc.21334 #
# #
# Please see http://vina.scripps.edu for more information. #
#####

Detected 8 CPUs
Reading input ... done.
Setting up the scoring function ... done.
Analyzing the binding site ... done.
Using random seed: -1017638142
Performing search ...
0% 10 20 30 40 50 60 70 80 90 100%
|----|----|----|----|----|----|----|----|----|
*****
done.
Refining results ... done.
```

```
9 files output. The first is ./PDB/out_1.pdb
```

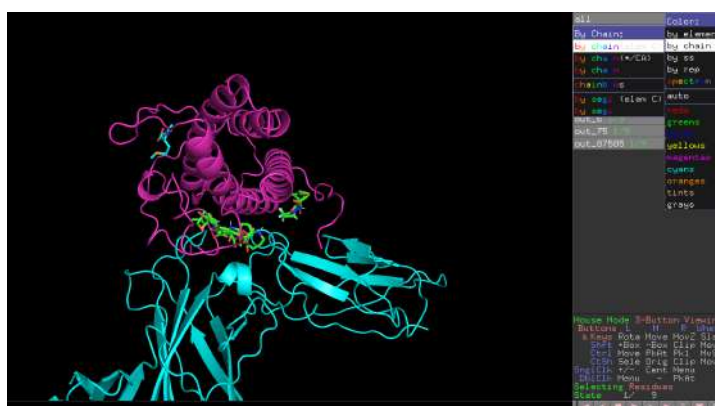
Pymol can visualise all docking results at once.

```
pymol ../01*/hmx.B99990001.pdb ./PDBQT/*
```

The representation might be improved by the following three steps:

1. Showing everything in the licorice and cartoon representation by clicking the Show (S) icon in the "all" row on the list on right panel.
2. Hide the licorice representation of the protein by clicking the Hide (H) icon in the hmx\_B99990001 row.
3. Enable the coloring method based on chain ID by clicking the Color (C) icon in the "all" row and choose by chain → by chain (elem C).

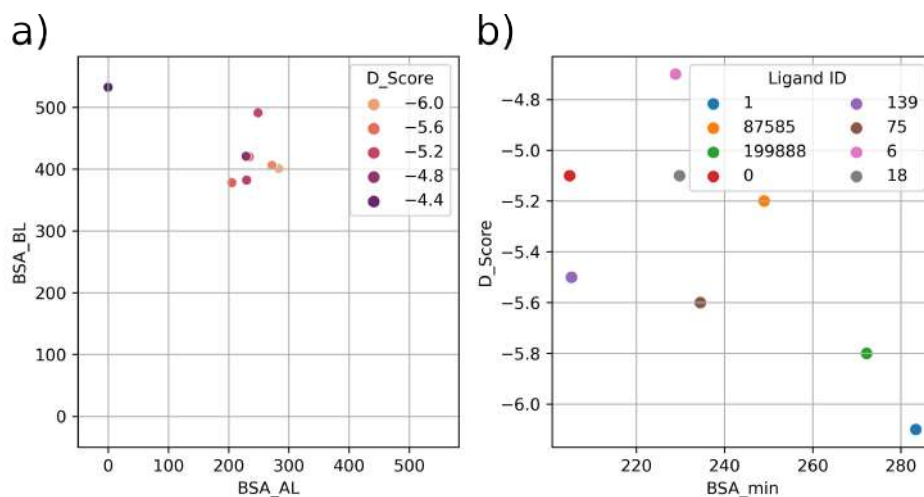
The result graphic should be something similar like Figure 9. The main point of the test docking test here is to check if your ligands are mostly docked in the desired interface pocket. If the visualisation looks substantially different, decreasing the pocket size in MGLTools might help or selecting another interface pocket altogether.



**Figure 9:** Visualization of the test docking output. In the lower right corner it reads State 1/9. This means a maximum of 9 docking poses are available for the docked ligands which can be navigated with the arrow keys.

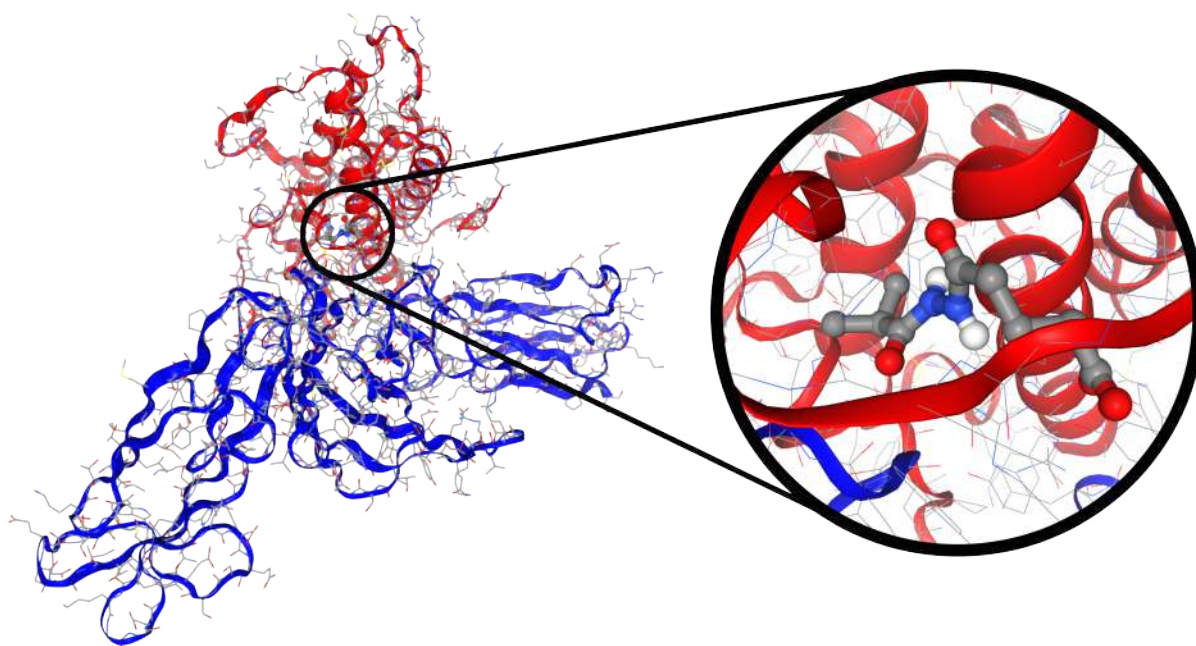
After the desired results are ensured, the BSA between the test ligands and two proteins can be calculated for comparison as well as their docking scores. The program `01_vinatest_analysis.ipynb` contains the analysis procedure. Executing all cells in said jupyter notebook, should visualize the distribution of BSA as well as the docking score similar to the pocket analysis in section 2.2. Note that the docking score here is in units of kcal/mol and the lower the docking score the stronger the expected binding affinity. From figure 10, it becomes apparent that ligand 1 has the lowest docking score and the largest  $BSA_{min}$ , thus it is potentially the best PPI stabilizer candidate among the test ligands.





**Figure 10:** Visualization of the docking result from AutoDock Vina in terms of BSA and docking score.

The jupyter notebook also visualizes the docking poses using NGLview. The resulting output of the visualization panel should look similar to figure 11. We can choose the ligand and the ligand state can be changed by setting `L_path` accordingly. Because calculating the BSA for a docking pose is relatively time-consuming, here in the code only the docking pose with the lowest docking score is analysed.



**Figure 11:** Visualization of the docking pose of the 1st docking mode of ligand 1 with a magnification of the ligand.

### 4.3 Large-Scale Screening with AutoDock-GPU

**Working Location:** 04\_virtual\_screening

Although docking with a few molecules using AutoDock Vina already yielded some good results such as **ligand 1**, screening a larger ligand database has a higher chance to discover a better PPIS candidate. With the confirmation that the obtained docking box is able to

successfully dock the ligands at the protein-protein interface, we can continue to use these parameters for large-scale ligand screening using AutoDock-GPU. Using a Nvidia RTX2080ti GPU enables a screening speed of up to 0.6 million compounds per week.

### 4.3.1 Setting up the docking grid of the receptor

Because AutoDock-GPU reads a set of grid parameter files, including `map`, `maps.fld`, `maps.xyz`, for docking, we need to create them with a ready made GPF file called `hmx.gpf` that can be found in `04.virtual_screening/`. Its contents can be seen below.

```
npts <BOXX> <BOXY> <BOXZ> # Box dimensions xyz (must be integer)
gridfld hmx_B99990001.maps.fld # grid_data_file
spacing 0.375 # spacing(A)
receptor_types A C N NA OA SA # receptor atom types
ligand_types A C F I NA Cl OA N P S Br HD # ligand atom types
receptor hmx_B99990001.pdbqt # macromolecule
gridcenter <COMX> <COMY> <COMZ> # box center xyz-coordinates
smooth 0.5 # store minimum energy w/in rad(A)
map hmx_B99990001.A.map # atom-specific affinity map
map hmx_B99990001.C.map # atom-specific affinity map
map hmx_B99990001.F.map # atom-specific affinity map
map hmx_B99990001.I.map # atom-specific affinity map
map hmx_B99990001.NA.map # atom-specific affinity map
map hmx_B99990001.Cl.map # atom-specific affinity map
map hmx_B99990001.OA.map # atom-specific affinity map
map hmx_B99990001.N.map # atom-specific affinity map
map hmx_B99990001.P.map # atom-specific affinity map
map hmx_B99990001.S.map # atom-specific affinity map
map hmx_B99990001.Br.map # atom-specific affinity map
map hmx_B99990001.HD.map # atom-specific affinity map
elecmap hmx_B99990001.e.map # electrostatic potential map
dsolvmap hmx_B99990001.d.map # desolvation potential map
dielectric -0.1465 # <0, AD4 distance-dep.diel;>0, cons
```

As per usual the name `hmx_B99990001` refers to the receptor complex of this tutorial and needs to be adjusted if any other protein is being screened. For tidiness reasons, the respective map files should be generated in a separate folder. The `autogrid4` command can automatically generate all related files, given the GPF file and the receptor file. The required commands can be seen below.

```
mkdir autogrid
cp -v ../03_docking_prep/hmx_B99990001.pdbqt hmx.gpf autogrid/
cd autogrid/
autogrid4 -p hmx.gpf
```

Upon successful execution a lot of new files should appear in the `autogrid` directory.

## 4.4 Docking with AutoDock-GPU

With the MAP files ready, the ligands prepared in `02.ligand_pre/PDBQT/`, and the receptor grid parameters prepared in `04.virtual_screening/autogrid/`, the next step is to generate a file for AutoDock-GPU to perform automatic docking experiments, called `docking.dat` which contains the locations of the input and output files and the location of the MAP files. Creating the file `docking.dat` and the docking itself should not be absolved manually, instead a readily available bash script `virtual_screening.sh` is supposed to take care of the procedure. The file contents are shown below.

```
# variable definitions for the input data
fld_path=autogrid/hmx_B99990001.maps.fld
lig_path=../02_ligand_prep/PDBQT

# variable definitions for the output directory
doc_path='docking'

# create directory if not already present
mkdir -p $doc_path

# create the file by writing the location of the maps.fld file to it
echo $fld_path > docking.dat

# cycle through all PDBQT ligands in $lig_path
for lig in $(ls $lig_path)
do
    # get the lig_id via bash parameter expansion
    lig_id=${lig##*_}
    lig_id=${lig_id%.*}

    # write the input and output paths into docking.dat
    echo "$lig_path/$lig" >> docking.dat
    echo "$doc_path/Ligand_$lig_id" >> docking.dat

    # breaks the for loop directly after the first iteration
    # this is for testing; comment out if large scale screening is desired
    break
done

# run AutoDock-GPU; The path to the executable might deviate from user to user
../AutoDock-GPU/bin/autodock_gpu_64wi --filelist docking.dat
```

Noticeably, the directory infrastructure is handled by the script itself, thus output files can be found in the directory to-be `docking/` in `04.virtual_screening/`. Furthermore, the for loop is automatically aborted after the first iteration if `virtual_screening.sh` is executed as is for testing purposes. In case of a successful execution, the `break` statement can be commented out with a `#`. The following execution of `virtual_screening.sh` will then try to dock all ligands saved in the specified directory. Be aware, that each user might have a different location of `AutoDock-GPU/` which then needs to be adjusted for.

```
./virtual_screening.sh
```

The premier output of this docking process are a DLG file and a XML file that can be found in the `docking/` folder for each docked ligand. The DLG file is the coordinate file which is very similar to a pdbqt file format-wise only that each line is prefixed with a `DOCKED:`. The standard output to the terminal for a successful docking should look similar to the text shown below.

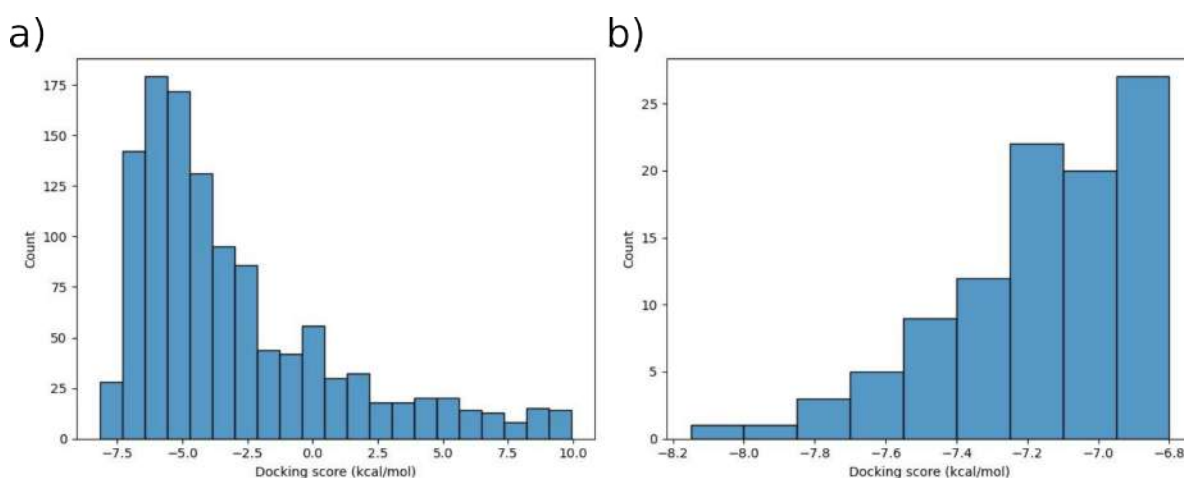
```
Running Job #2224:
  Device: NVIDIA GeForce GTX 1080
.
.
.
the best molecules of the last 4 * 5 generations, 42000 generations, or 1132076 evaluations:

Generations | Evaluations | Threshold | Average energy o...
-----+-----+-----+-----+
0 | 150 | 232675.16 kcal/mol | 0.00 +/- 0...
5 | 25110 | 232675.16 kcal/mol | 0.00 +/- 0...
.
```

0.00 +/- 0.00 kcal/mol combined.  
0 samples, best energy 232675.16 kcal/mol.

#### 4.4.1 Analysis and Visualisation of Docking Results from Autodock-GPU

After the ligand screening finished, the results should be analysed in a similar manner as for the test docking in section 4.2. In `04_virtual_screening` a jupyter notebook can be found that contains the code for said analysis. In the beginning of the notebook `01_GPUDock_analysis.ipynb` the docked ligand files (DLG format) are read and the docking scores for the best state of each compound is saved and then plotted as a histogram. The resulting plot is saved and should look similar to figure 12. Noticeably, only compounds with a scoring lower than 10 are plotted, since many compounds might have scores which are magnitudes above that which would skew the distribution.



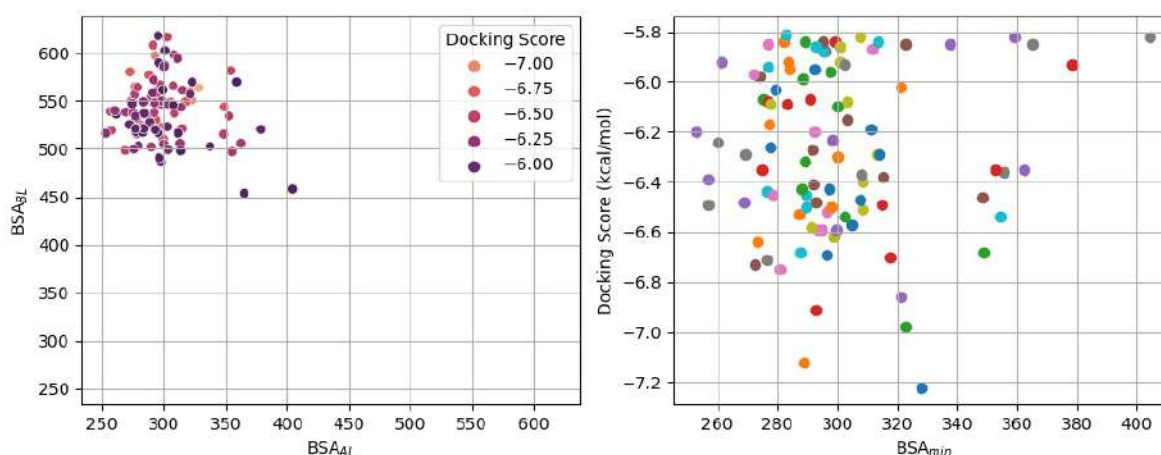
**Figure 12:** Sub-distribution of docking score outputs from AutoDock-GPU by reading the best-scored docking pose of each ligand.

The scoring of the compounds allows to undertake a selection of the compounds. Herein, the 100 compounds with the lowest scores are picked for further processing. The notebook features a histogram of their docking scores again (see figure 12), and then converts them from DLG to PDBQT format followed by yet another conversion into SMI, MOL2 and PDB format which is achieved by executing `convert.sh` from the notebook. This bash script utilises OpenBabel, thus the correct conda environment needs to be activated. The contents of the bash script are shown below.

```
#!/bin/bash
pdbqt='./PDBQT'
pdb='./PDB'
mol2='./MOL2'
smi='./SMI'
mkdir -p $pdb $mol2 $smi
for file in $pdbqt/*
do
    # get basename via parameter expansion
    name="${file##*/}"
    lig="${name%.pdbqt}"
    # converting to pdb
    echo "$pdbqt/$lig"
```

```
obabel -ipdbqt $pdbqt/$lig.pdbqt -opdb -O $pdb/$lig.pdb
# converting to mol2
obabel -ipdbqt $pdbqt/$lig.pdbqt -omol2 -O $mol2/$lig.mol2 -p 7.4
# converting to smi
obabel -ipdbqt $pdbqt/$lig.pdbqt -osmi -O $smi/$lig.smi
done
```

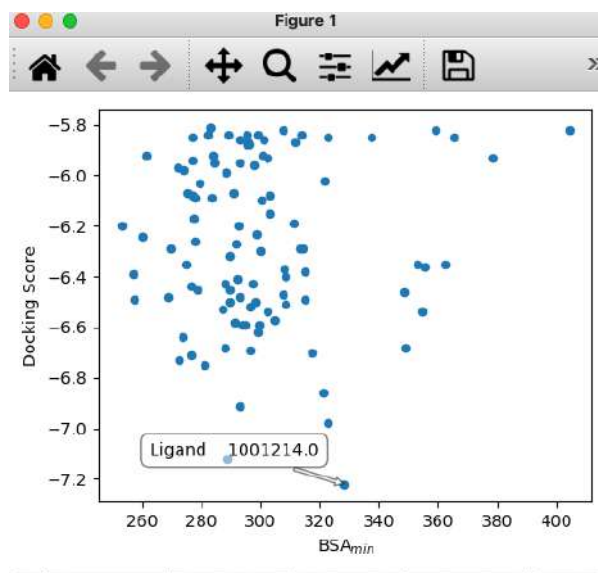
After both conversions, the well-known calculation of the BSA is conducted for the top compounds and are visualised in the usual way. Again this is done to quantify how well the ligands are staying at the interface by calculating the buried surface area with protein A and protein B. An exemplary distribution of docking score,  $BSA_{AL}$ , and  $BSA_{BL}$  are shown in figure 13.



**Figure 13:** Data visualization of the protein-ligand BSA and docking score. **Left** Distribution of  $BSA_{AP}$  and  $BSA_{BP}$  of all detected pockets with color coded docking score. **Right** Distribution of  $BSA_{min}$  and the docking score of the interface ligand.

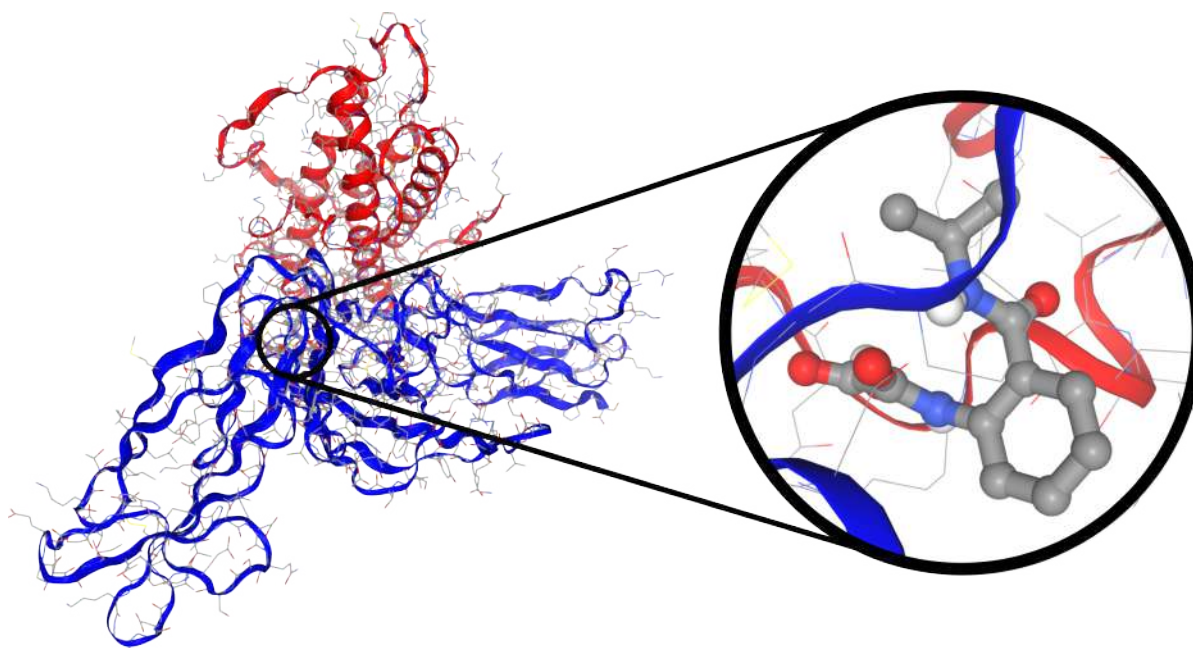
Since a good PPI stabilizer needs to have sufficiently high binding affinity to both proteins, we want to choose a ligand with both high docking score and high  $BSA_{min}$ . There is one cell in `01.GPUdock.analysis` that utilises the package `mplcursor` to plot the same distribution shown in figure 13 with the difference, that it opens in another window, and that the ligand ID appears upon hovering over the data point with the cursor, as shown in figure 14. This mechanic comes in handy for picking interesting compounds for further processing.





**Figure 14:** Data visualization with mplcursors

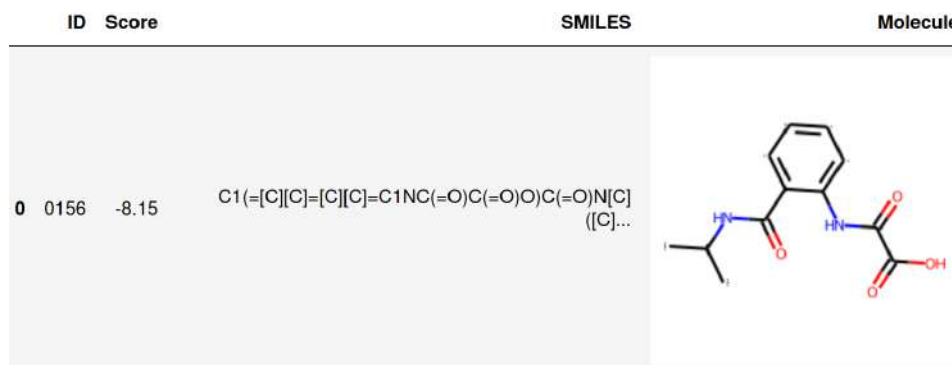
Once, a promising compound could be made out its docking pose can be visualised via `nglview` in the current notebook. By setting the `L_path` to the respective PDB output path in the respective cell, the ligand docking pose can be seen at the interface as shown in figure 15.



**Figure 15:** Visualization of Ligand 0156 docked at the interface between protein A (blue) and protein B (red).

The generated SMI files stored in `04_virtual_screening/SMI/` are used in the last step of the jupyter notebook. These files contain the SMILES (Simplified Molecular Input Line Entry System) string of each compound, which is unique to each compound (but not unique for each compound) and can be read by RDkit and used to search compounds in the ZINC20 database. Executing the respective cell in the jupyter notebook visualises the molecular graph of each compound in a Pandas Dataframe similar to figure 16.





**Figure 16:** Molecular graph of the docked compound 0156.

For some users this can raise a `.get_adjustment` Attribute error that was mentioned in section 0.3. It can be fixed as described in `00_install_prep/TOY.ipynb`, thus by changing all `.get_adjustment` to `._get_adjustment` in the PandasTools file, to which the path should be shown in the error message. Note, if you encounter this error message, you need to restart your notebook by restarting (go to kernel → Restart) before the bug fix takes effect. Apart from visualising the molecular graph, the SMILES string enables the user to easily purchase these compounds by copying it and paste it in the ZINC20 Substance page (<https://zinc20.docking.org/substances/home/>). This should direct you to the site where available vendors are listed.

This concludes the docking process. If all work until here was successful, the docked compounds can be processed further, or an even larger screening can be conducted before continuing by going back to section 4.4 and increase the number of ligands for GPU-docking.

## 5 Molecular Dynamics Simulation with GROMACS

When it comes to drug screening, the whole process can be imagined as a funnel that is increasing in sensitivity with each step. The first step conducted, was prompting the ZINC20 database more or less arbitrarily, in order to reduce the unfathomable amount of available chemical compounds by a large margin. The next step included the application of chemical criteria that the compounds should obey. Even though they were chosen arbitrarily in this tutorial they might as well incorporate empirical knowledge relevant for the system at hand. This was followed by the docking experiments assigning each compound a binding score and estimated buried surface area. The sensitivity of these metrics is already quite high and enables us to discard a large amount of unsuitable compounds. Only 2-5 of the best ranking compounds should be tested in order to taper the funnel.

The next step is to conduct a molecular dynamics (MD) simulation with GROMACS[1] to investigate the dynamical properties of the protein-protein-ligand ternary complex as well as to estimate the binding free energy with MMPBSA. MMPBSA is a middle ground between docking and free energy perturbation (FEP). Whilst docking is fast, the results are not reliable, however, FEP calculations impose a ludicrous computational cost that cannot be covered on a large scale. Especially with large ternary systems as the one that is investigated in this tutorial. Therefore, MMPBSA is the method of choice, since it is especially suited for comparing the free energies of similar systems to each other.

### 5.1 Ligand Parameterization and MD simulation

Working Location: `05_mdsimulation/`

The working directory contains five subfolders to keep things ordered: `_exec`, `_inp`, `_output`, `_sim` and `_trash`. The structure of these folders should not be changed, since it is integral to the individual steps of the MD simulations.

- `_exec` contains binaries, to be executed or to be distributed into the simulation directories.
- `_inp` contains input files to be copied into the simulation directories where they are usually modified or processed in different ways.
- `_output` contains output files from analysis of the simulations, like plots or data tables with calculated results.
- `_sim` contains the simulation directories. Herein, `_sim` will contain one further subdirectory for each ligand in which the actual simulation runs will be conducted.
- `_trash` contains seemingly unnecessary files. 'Seemingly', because deleted files tend to become necessary again right after they were deleted. Therefore, it is recommended not to use `mv` rather than `rm`. The contents of `_trash` may be deleted in the distant future.

The first step in simulating the ligands, is to change the directory into `_exec` and have a look into the `prep_setup.sh` script whose contents are shown below.

```
#!/bin/bash

# declare identifier of ligand and receptor and the pdb file created during the receptor
# preparation
R_id='3hmx'
L_ids=(1546 1306)
R_pdb='../01_receptor_prep/hmx.B99990001.pdb'

# declare variables of the respective directories
INP='../_inp'
SIM='../_sim'
MOL2='../04_virtual_screening/MOL2'

# copy the receptor pdb into the input directory for leaping it later for parameter preparation
cp -v $R_pdb $INP/$R_id.pdb

# loop over the ligand ids specified above
for L_id in "${L_ids[@]}"
do
    # declare a combined variable of receptor and ligand for easier readability
    R_L=${R_id}_${L_id}
    # create the simulation directories for each ligand automatically (more consistent)
    mkdir -v -p $SIM/$R_L
    # copy the executables to the respective sim directory and substitute the correct IDs
    cp -v prep_and_run.sh $SIM/$R_L/
    cp -v convertTOP_amb2gmx.py $SIM/$R_L/
    sed -i "s/LIGANDID/$L_id/g" $SIM/$R_L/prep_and_run.sh
    sed -i "s/RECEPTORID/$R_id/g" $SIM/$R_L/prep_and_run.sh
    chmod 777 $SIM/$R_L/prep_and_run.sh
    # copy the mol2 file of the each ligand here for manual modification (overwriting is
    # deactivated via -n)
    cp -v -n $MOL2/Ligand_$L_id.mol2 $INP/$L_id.mol2
done
```

The in the beginning of this program, there are some variables that might need adjustment from the user. Most noticeably, `L_ids` should be set to the ligand ID numbers that were found

to be the best ranked ones. As a reminder, the ranking and selecting of ligands is done with `01_GPUdock_analysis.ipynb` in the prior step. Anything else does not need to be changed as long as no unforeseen manual modifications were conducted by the user. When in doubt, the variable declaration should be checked.

Among the tasks that `prep_setup.sh` completes, one of them is to copy `prep_and_run.sh` in each ligand simulation directory and to modify them accordingly. `prep_and_run.sh` is the main executable and takes care of the parameterisation of the ligands and the MD simulation itself. The whole process is quite complicated, thus the bash script is elaborated and complex consisting of four different functions which will not be printed in this tutorial sheet, but can be inspected by opening the file via `vi`. Nevertheless, they shall be briefly introduced.

- `cp_inp_files`: Copies all necessary input files from the input directory into the current simulation directory and adjusts them for the respective ligand and receptor.
- `gen_params`: The ligand is most likely not a standard residue that can be simply looked up in a standard forcefield. Therefore, some programs from `ambertools` are used to generate custom parameters for these files. They might be not cutting-edge, but the parameters will be a reasonable trade-off between computational cost and accuracy. There are several important output files of this function like the FRCMOD file and PDB file of the complex.
- `gen_gmx_top`: The output from `ambertools` is suitable for the AMBER simulation package which is not open source unfortunately. Therefore, the output files need to be converted to GROMACS compatible files, most importantly a working `topol.top` which is the output of this function together with the respective GRO file.
- `run_gmx_sims`: Once the TOP and GRO file are present the simulation box can be setup followed by the usual simulation sequence. This is taken care of by this function (given a working GROMACS installation) in three repetitions in order to calculate more reliable average properties in the analysis.

It is recommended to execute each function individually instead of all at once. Their execution can be switched off by inserting a `#` in front of their call at the end of `prep_and_run.sh` and it can be switched on by removing it. Natively, `cp_inp_files` and `gen_params` should be switched on and the other ones should be switched off. This should not be changed at this stage, since the parametrisation of the MOL2 ligands from OpenBabel often contain artifacts. Executing the `prep_and_run.sh` in the respective ligand simulation directory often leads to a fatal error in the `antechamber` program similar to the one seen below.

```
./prep_and_run.sh
...
...
...
/home/user123/anaconda3/envs/ppis/bin/to_be_dispatched/antechamber: Fatal Error!
Weird atomic valence (3) for atom (ID: 4, Name: C1).
    Possible open valence.
...
...
```

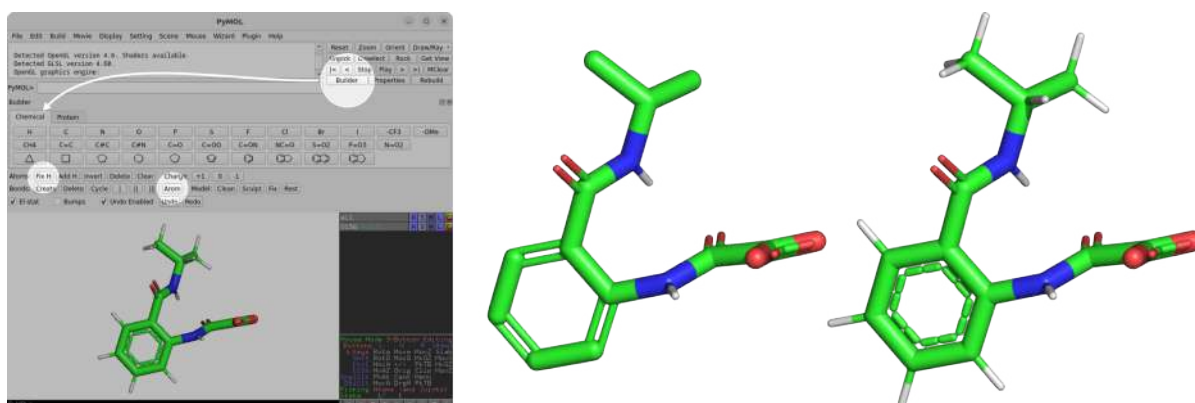
This means the valence charges assigned to the atoms of the ligand does not fulfill the octet rule, usually caused by the incorrect bonds assigned to nitrogen or misplacement between double bond or aromatic ring by OpenBabel. This problem needs to be manually corrected by using pymol. The file that needs to be corrected is the MOL2 file in the `_inp/` directory, e.g. `_inp/1234.mol2`.

```
cd path/to/_inp/  
pymol 1234.mol2
```

The conda installed Pymol version has a tool for building and editing molecular structures. Most valence problems can be solved by fixing the hydrogen atoms of the structure and by adding the correct aromaticity. The procedure depicted in figure 17 can be used as guidance. Once the structure has been edited, overwrite the MOL2 file of `1234.mol2` with the edited structure and execute `prep_and_run.sh` again.

```
cd path/to/_exec/3hmx_1234/  
./prep_and_run.sh
```

If the error persists, more editing in pymol might be attempted or a different compound can be chosen for parametrisation and simulation.



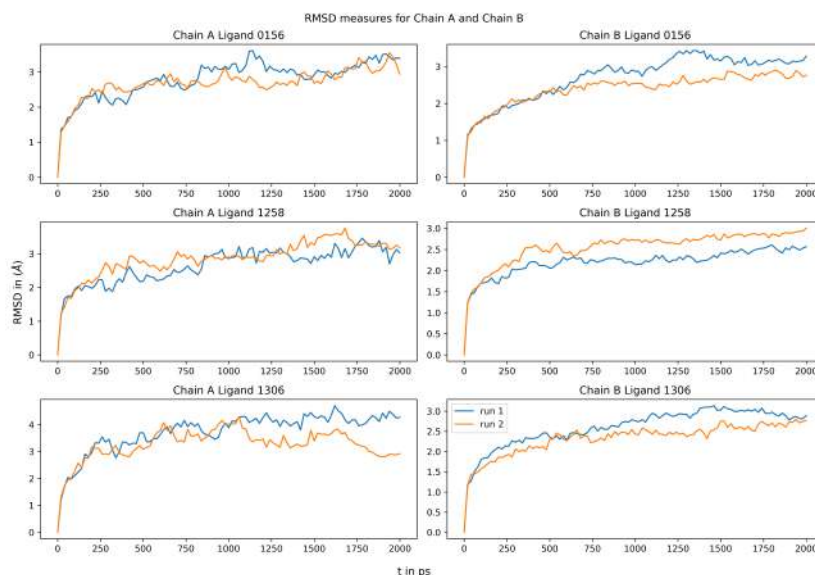
**Figure 17:** The process of editing a ligand with Pymol is shown. The left picture shows how to access the builder menu. The two options `Fix H` and `Arom` were used in this specific example. The middle picture shows the ligand in its erroneous state. Hence trying to generate parameters will fail. The right picture depicts the ligand state resulting in a successful parametrization. Noticeably, aromaticity is added as well as the explicit hydrogen atoms. There are lots of other caveats that could be wrong with a ligand, and some intuition is necessary for each fix.

In case of an error free parametrization, the next function `gen_gmxtop` can be switched on and the prior two can be switched off. There is no user interaction required, however, the function may show the following error, which can be ignored.

```
Traceback (most recent call last):  
  File "convertTOP_amb2gmx.py", line 596, in <module>  
    perm_imp = get_permutations_4_wild(dihedral[0:4])  
  File "convertTOP_amb2gmx.py", line 121, in get_permutations_4_wild  
    temp_a = perm[2]  
IndexError: list index out of range
```

The required `topol.top` and the GRO file should still be outputted. After asserting that these files exist, the last function of `prep_and_run.sh` may be executed. This function, `run_gmx_sims` carries out the gromacs simulations for the user. The MDP files were already copied from `_inp/` and the naming of the simulation output is taken care of automatically. Each production run should be 2 ns long. The simulations can take some time, since the system is quite large, however once they are done there should be three trajectories per compound, named `prod_center_<REP>.xtc` containing 100 frames each to go easy on the disk space. Once all simulations are finished, the trajectory analysis can start, which is prepared in the jupyter notebook `01_analyse_trajectory.ipynb` in the `_exec` folder. The analysis of the trajectory is not quite extensive, however, some important properties are computed, that reveal the stability

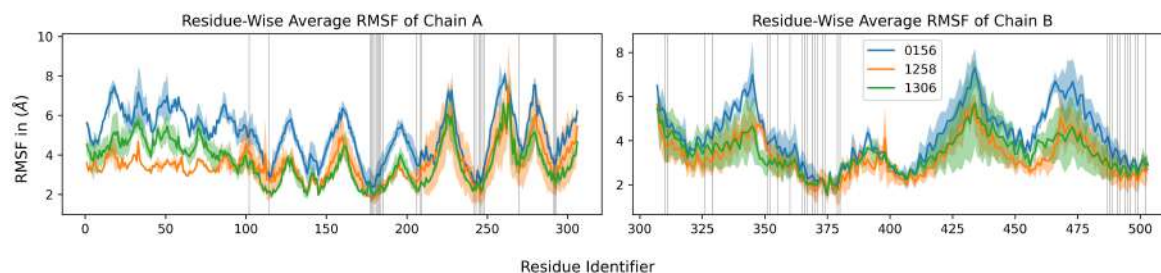
of the receptor complex. Firstly, the RMSD of over the trajectory is measured exemplary for three compounds as shown in figure 18. The RMSD is a measurement that reveals the average structural deviation from a reference structure for every time frame. Each run is depicted as an individual curve with the first frame of the trajectory as reference. Should the two curves deviate by a large margin, there could be a problem with one of the runs, since clearly different states are reached in each simulation.



**Figure 18:** The RMSD is shown vs the simulation time. Each row corresponds to a different ligand and while the columns correspond to the two protein chains.

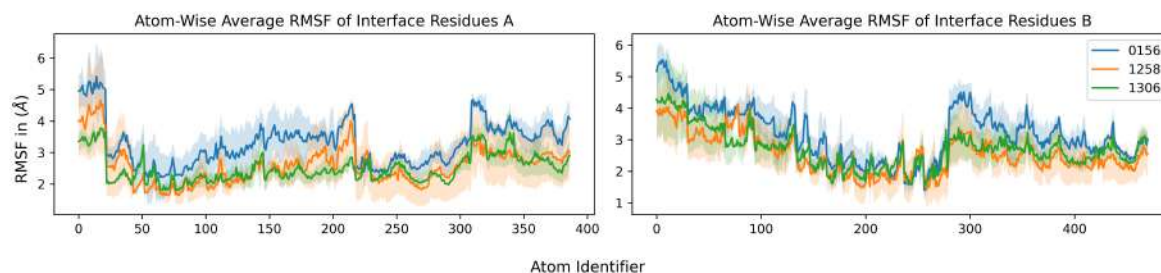
Apart from the RMSD, the RMSF is calculated as well, which gives the structural deviation as a time average per atom. This property is a bit more elucidating in terms of structural integrity, since stable atoms and stable conformation should exhibit low RMSF value. In this notebook, three different ways to measure the RMSF are conducted. Firstly, the atom-wise RMSF which is done for showcasing the function itself rather than gaining information. The plot is not shown here. Secondly, the atomic RMSF is averaged per residue, which reveals residues that are very mobile and those that are less mobile. The expectation would be, that residues in loops and unhindered parts of the protein have a large RMSF value and those residues that are buried and in the interface have a lower RMSF value. The residue-wise RMSF is also calculated for three different compounds however, the values are being averaged over the two runs. The results can be seen in figure 19 with one curve per compound. Generally, the expectation, that residues within the interface have lower RMSF is validated in this plot and there are large differences visible between each compound's trajectories. For example, The curve of Ligand 0156 is generally higher than the other two for both protein chains and the curve of compound 1258 seems to be most stable in general. Especially, the stable regions outside of the interface cannot be attributed to the compound stabilisation, but rather to the nature of the starting structure and its individual time development. Already, it can be seen, that the sampling might be insufficient to properly determine the most successful PPIS, however it suffices for didactical purposes.





**Figure 19:** The residue-wise RMSF is plotted with errorbars shown as filled in area. Although each curve follows a certain trend, the compounds largely differ with respect to protein chain stability. The interface residues are shown as grey vertical lines.

The most important information is given in the last plot of the jupyter notebook, which consists of the atom-wise RMSF but for the interfacial residues only. Again, it can be seen that the stabilisation differs greatly between each compound. The ligand 1258 seems to be the best stabiliser with respect to the interface residues together with ligand 1306 (see figure 20). Nevertheless, these conclusions should be drawn with a grain of salt, since this analysis is not thorough.



**Figure 20:** The atomic RMSF for the interface residues for each protein chain. In both protein chains, compound 0156 seems to be the least performant while the other two compete for the most stable interface. The errorbars are again shown as filled in area with the same color shade.

In exercise 8, students have already learned how to check convergence of a system with respect to certain measurements and how a certain sampling sufficiency can be visualised. These tests are lacking here, and furthermore, the system at hand is quite large and would indeed need a very long time to properly converge. Nevertheless, the obtained results serve as an example on how to look for potential PPIS with the trajectory only. The next step is to estimate the free binding energy with MM/PBSA to properly determine which compound has the most potential.

## 6 Binding free energy estimation with MM/PBSA

### Working Location: 06\_mmpbsa

With the `ppis` environment activated, the bash script `prep_mmpbsa.sh` is next in line. This is yet another multipurpose bash script which handles a lot of tasks at the same time. The main functionalities include the conversion of the GROMACS trajectory to an AMBER trajectory, its splitting into the different system components (Chain A, Chain B and the ligand) as well as the directory structure which needs to be set up for every selected compound. The identifiers for the ligands and the receptor can be found in the beginning of `prep_mmpbsa.sh` and need to be adjusted accordingly before executing it.



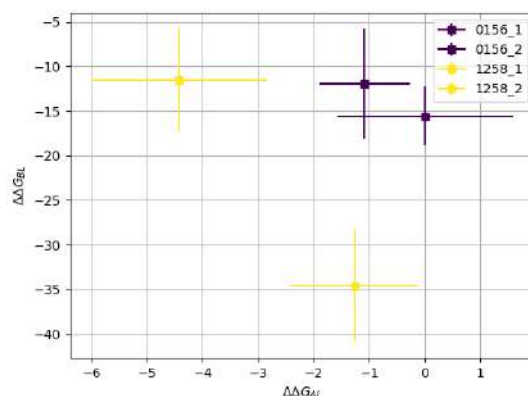
```
./prep_mmpbsa.sh
```

After an error-free execution all necessary files should already be transferred into the newly created subdirectories called `<REC-NAME>_<LIG-ID>/`. One of these files is called `mmpbsa.sh` which is used to estimate the binding free energies. Executing it will take a few hours and should result in a couple of DAT and CSV files, e.g. `AL_output_1.dat`. The progress can be tracked by using `grep` on the MDOUT file together with a line count command. Since MMPBSA analyses the trajectory for the complex, the receptor and the ligand, there are a total of 300 frames. Thus, once the following code outputs 300, the calculations should be done.

```
grep 'NSTEP' _MMPBSA*.mdout.0 | wc -l
```

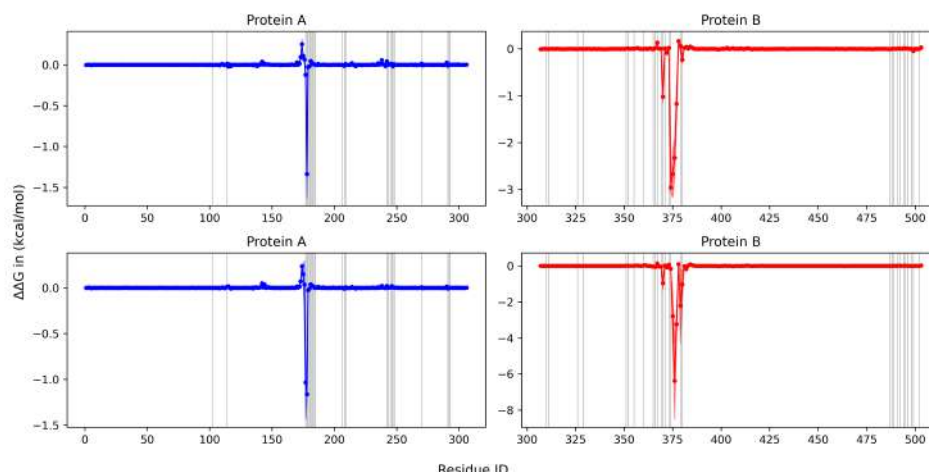
The DAT files contain the total free energy of binding and the CSV files contain the residue-wise decomposition of the free energy. The final verdict on which ligand can be elected to be the optimal PPIS should be guided by the total free energy. The residue wise decomposition can facilitate the process of lead-optimisation, i.e. the knowledge based engineering of the best compound to unlock further potential as a PPIS. To ease this process, a jupyter notebook can be found, in `06_mmpbsa` called `01_visualise_MMPBSA_results.ipynb`. As per usual, the identifier of the ligand needs to be adjusted before executing the specific cells. Also the correct conda environment needs to be activated.

The first plot of the notebook provides information about the total binding free energy from the MM/PBSA calculation which can be considered an overall score of the compound. In this notebook, the total binding free energy is shown for each run of each compound, since averaging dilutes some information. Good PPIS have small binding free energies to both protein chains while not favouring any of the two significantly (see figure 21).



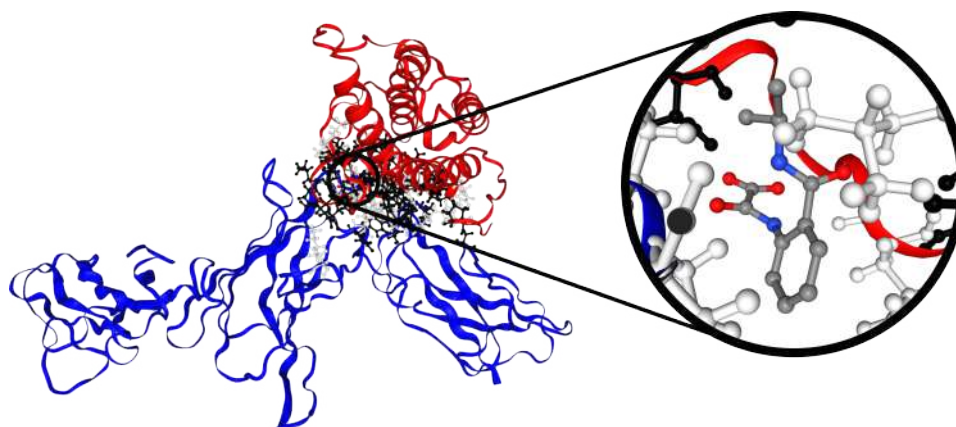
**Figure 21:** Exemplary distribution of the binding free energies of 2 runs of 2 ligands. The free binding energy estimate for protein A is plotted on the x axis and the one for protein B is plotted on the y axis.

More useful information is given in the second plot. Per default, the MM/PBSA calculation outputs a residue wise decomposition of the free energy. This output is being read by the jupyter notebook and depicted per ligand and per protein chain. Most residues do not really interact with the compound and do not contribute at all, but mainly the interface residues, depicted as grey, vertical lines contain non-zero contributions for both, protein A and B (see figure 22).



**Figure 22:** Energy Decomposition of binding free energy. Each row belongs to a compound and each column the respective protein. The black vertical lines depict interfacial protein residues.

Last but not least, the jupyter notebook also visualises the structure of the complex in a very specific way that illuminates why specific residues facilitate and others inhibit the compound binding. In this visualisation, all interfacial residues, that facilitate binding are represented in white balls and sticks, while the unfavourable residues are shown as black balls and sticks. By inspecting the residues and their behaviour over the trajectory, one may be able to optimise the drug candidate further to interact more favourably with both proteins (see figure 23).



**Figure 23:** The structure of the complex is rendered with NGLview once again, but this time the residues of the interface are colored in by their contribution to the free energy of binding. In the zoomed in section of the figure, one residue close by the ligand is interfering with the binding (black color), posing as a target for optimisation.

With the information provided in the plots of the jupyter notebook, one compound may be presented as the optimal PPIS. It is an optional task to present this compound structurally, as well as the metrics that guided the decision making during this drug screening process.

## Conclusion

This tutorial presented an example workflow in a computer-assisted drug design application to find a protein-protein interaction stabilizer from a database. It has become evident that many steps are involved and that a wide range of computer tools has to be employed, from the initial

preparation of target-protein files up to a detailed quantitative thermodynamic characterization of the drug-target complex.

Firstly, it is very important to understand that the approach chosen in this tutorial is not universal and not optimal as very many pre-made decisions present trade-offs for the sake of didactics. Secondly, one has to realize the sheer amount of pre-made decisions in this tutorial. From the MMPBSA algorithms in `gb.in`, over the number of frames in the MDP files to the scoring function in the molecular docking step. All these choices must be taken with care in order to end up with reliable results. Last but not least, the question arises on how to proceed from here. The next steps in the computational domain could engulf a lead optimisation guided with the MMPBSA decomposition, higher precision free energy calculations like free energy perturbation or both. Experimentally, one could conduct in-vitro assays to determine the real binding behaviour of the drug candidates. Yet again, a lot of choices to think about even after the virtual drug screening is successfully conducted. This makes computer-assisted drug design an exciting and challenging task. Additionally, the field evolves dynamically and certainly it will benefit from the development of new tools involving artificial intelligence.

## References

- [1] Mark Abraham, Andrey Alekseenko, Cathrine Bergh, Christian Blau, Eliane Briand, Mahesh Doijade, Stefan Fleischmann, Vytautas Gapsys, Gaurav Garg, Sergey Gorelov, Gilles Gouaillardet, Alan Gray, M. Eric Irrgang, Farzaneh Jalalypour, Joe Jordan, Christoph Junghans, Prashanth Kanduri, Sebastian Keller, Carsten Kutzner, Justin A. Lemkul, Magnus Lundborg, Pascal Merz, Vedran Miletic, Dmitry Morozov, Szilárd Páll, Roland Schulz, Michael Shirts, Alexey Shvetsov, Bálint Soproni, David van der Spoel, Philip Turner, Carsten Uphoff, Alessandra Villa, Sebastian Wingbermühle, Artem Zhmurov, Paul Bauer, Berk Hess, and Erik Lindahl. Gromacs 2023.2 manual, July 2023.
- [2] D.A. Case, H.M. Aktulga, K. Belfon, I.Y. Ben-Shalom, J.T. Berryman, S.R. Brozell, D.S. Cerutti, T.E. Cheatham III, G.A. Cisneros, V.W.D. Cruzeiro, T.A. Darden, N. Forouzesh, G. Giambasu, T. Giese, M.K. Gilson, H. Gohlke, A.W. Goetz, J. Harris, S. Izadi, S.A. Izmailov, K. Kasavajhala, M.C. Kaymak, E. King, A. Kovalenko, T. Kurtzman, T.S. Lee, P. Li, C. Lin, J. Liu, T. Luchko, R. Luo, M. Machado, V. Man, M. Manathunga, K.M. Merz, Y. Miao, O. Mikhailovskii, G. Monard, H. Nguyen, K.A. O'Hearn, A. Onufriev, F. Pan, S. Pantano, R. Qi, A. Rahnamoun, D.R. Roe, A. Roitberg, C. Sagui, S. Schott-Verdugo, A. Shajan, J. Shen, C.L. Simmerling, N.R. Skrynnikov, J. Smith, J. Swails, R.C. Walker, J. Wang, J. Wang, H. Wei, X. Wu, Y. Wu, Y. Xiong, Y. Xue, D.M. York, S. Zhao, Q. Zhu, and P.A. Kollman. Ambertools23, 2023.
- [3] Jerome Eberhardt, Diogo Santos-Martins, Andreas F. Tillack, and Stefano Forli. Autodock vina 1.2.0: New docking methods, expanded force field, and python bindings. *J. Chem. Inf. Model.*, 61(8):3891–3898, August 2021.
- [4] Narayanan Eswar, Ben Webb, Marc A. Marti-Renom, M. S. Madhusudhan, David Eramian, Min-Yi Shen, Ursula Pieper, and Andrej Sali. Comparative protein structure modeling using Modeller. *Current Protocols in Bioinformatics*, Chapter 5:Unit–5.6, October 2006.
- [5] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant,

- Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.
- [6] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.
- [7] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016.
- [8] Vincent Le Guilloux, Peter Schmidtke, and Pierre Tuffery. Fpocket: An open source platform for ligand pocket detection. *BMC Bioinformatics*, 10(1):168, 2009.
- [9] Robert T. McGibbon, Kyle A. Beauchamp, Matthew P. Harrigan, Christoph Klein, Jason M. Swails, Carlos X. Hernández, Christian R. Schwantes, Lee-Ping Wang, Thomas J. Lane, and Vijay S. Pande. Mdtraj: A modern open library for the analysis of molecular dynamics trajectories. *Biophysical Journal*, 109(8):1528 – 1532, 2015.
- [10] Noel M. O’Boyle, Michael Banck, Craig A. James, Chris Morley, Tim Vandermeersch, and Geoffrey R. Hutchison. Open Babel: An open chemical toolbox. *Journal of Cheminformatics*, 3(1):33, October 2011.
- [11] The pandas development team. pandas-dev/pandas: Pandas, February 2020. <https://doi.org/10.5281/zenodo.3509134>.
- [12] MF Sanner. Python: a programming language for software integration and development. *Journal of molecular graphics and modelling*, 17(1):57—61, February 1999.
- [13] Diogo Santos-Martins, Leonardo Solis-Vasquez, Andreas F Tillack, Michel F Sanner, Andreas Koch, and Stefano Forli. Accelerating AutoDock4 with GPUs and Gradient-Based Local Search. *Journal of Chemical Theory and Computation*, 17(2):1060–1073, February 2021.
- [14] Schrödinger, LLC. The PyMOL molecular graphics system, version 1.8. pymol.org, November 2015.
- [15] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.
- [16] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.
- [17] Andrej Šali and Tom L. Blundell. Comparative protein modelling by satisfaction of spatial restraints. *Journal of Molecular Biology*, 234(3):779–815, 1993.