Guide for Using Rosetta when Designing Ligand Binding Sites

C. Coll[1], A. Sandelin[1], A. Gynter[2], M. Rajakenttä[1], T. Jämsä[2], D. Pająk[1], E. Barannik[2], G. Åberg[2], N. Lindholm[2], J. Manninen[1]

All authors are part of the Aalto-Helsinki 2020 iGEM team
[1]University of Helsinki
[2]Aalto University

ABSTRACT

During the past years, computational methods and tools for biology have been developed at a speed never seen before. They allow scientists to carry out preliminary studies before working in the lab, saving both time and costs. One of the most commonly used tools for protein studies is Rosetta, a software that includes algorithms for computational modelling and analysis of protein structures. It allows enzyme design, *de novo* protein design, ligand docking, as well as structure prediction of biological macromolecules and complexes among others. Here, we present a guide on how to use Rosetta for the design of ligand binding sites. During our iGEM project, we have seen that although extremely useful, Rosetta and other software may lack suitable documentation for unexperienced computational biologists. We hope this guide will help to change this.

Key words: Rosetta, ligand binding site(s).

INTRODUCTION

We present a step-by-step guide on how to design ligand binding sites with the Rosetta software. This guide is based on the article *Rosetta and the Design of Ligand Binding Sites* published by Moretti and the colleagues (2016). Importantly, this guide is only addressed to users that want to design ligand binding sites of proteins that bind to one or more ligand molecules, as long as the molecules bound are the same. We acknowledge that Moretti's article is already concrete and useful, but non-computational biologists might have some trouble following the steps and understanding what is being done. We hope this guide will help to change this. We are first going to point out the considerations needed to be taken into account when designing a ligand binding site, the materials and software required, the necessary steps and, ultimately, how to interpret scores in Rosetta.

CONSIDERATIONS

The first thing that non-computational biologists need to know is that Rosetta is a non-graphical software exclusively run in a Unix-like environment. Nonetheless, after enough practice, they can see it is not as difficult to use as it might seem at the beginning.

Before starting the design of a ligand binding site, there are several important considerations regarding the protein that need to be taken into account. First of all, the protein and the ligand structures to be modelled need to be in a database in order for the user to be able to work with the files (see more in Materials and Software section). It is of great importance that the user knows the structure of the protein, especially the number of chains (named A, B, C…), how many ligand molecules does it bind to, and if it binds to more than one ligand, whether the molecules are the same. This will be of special relevance when preparing the scripts for the modelling of the binding sites.

MATERIALS AND SOFTWARE

In order to use Rosetta for the design of ligand binding sites, several materials and software are required.

Materials

1. The structure of the protein to be re-designed, in PDB format. The main database of tridimensional protein structures is Protein Data Bank (PDB).
2. The structure of the ligands that bind to the protein that wants to be re-designed. Examples of chemical databases where ligands can be found are PubChem, ChemSpider or Zinc, among others.

Software

1. **A *Unix*-like operating system such as Linux.**
2. **Access to a computer cluster.** A computer cluster is a set of connected computers with combined computational power, which allows to cut significantly the calculation time for heavy tasks. Rosetta's simulations are computationally heavy and normally the users' computers take a lot of time to run the jobs or even do not have enough memory or CPU's (central processing units) to run them. Thus, access to a computer cluster is needed if results are wanted to be obtained in a reasonable time-frame.
3. **Rosetta.** A license is needed to download Rosetta. There are two types of licenses: academic or commercial. After requesting it, the software can be downloaded on the following page: https://www.rosettacommons.org/software/license-and-download.
4. **PyMOL.** PyMOL is an open-source molecular visualization software used to study the structure of proteins and molecules. The software is free to download on the following website: https://pymol.org/2/. Another existing software with the same purpose is Chimera (https://www.cgl.ucsf.edu/chimera/download.html).
5. **OpenBabel.** OpenBabel is a software used to study chemical data. In our case, it is used to introduce some modifications to the ligands of the protein that is going to be re-designed. The download and installation instractions can be found on https://openbabel.org/docs/dev/Installation/install.html.
6. **MGL tools (AutoDock 1.5.6).** AutoDockTools is a graphical front-end for setting up and running AutoDock - an automated docking software designed to predict how small molecules, such as substrates or drug candidates, bind to a receptor of known 3D

structure. The software can be downloaded from here: http://mgltools.scripps.edu/downloads.

7. **For MacOS: XQuartz.** XQuartz is needed to run the AutoDockTools. The XQuartz project is an open-source effort to develop a version of the X.Org X Window System that runs on MacOS. Together with supporting libraries and applications, it forms the X11.app that Apple shipped with MacOS versions 10.5 through 10.7. It can be downloaded from here: https://www.xquartz.org/.
8. **AutoDock Vina.** AutoDock Vina is an open-source program for doing molecular docking, which can be downloaded here: http://vina.scripps.edu/download.html.

STEP-BY-STEP GUIDE

In this step-by-step guide we provide the user with the steps and commands they have to run in order to re-design the ligand binding sites of the protein of interest. Notes for the reader: (i) the text in italics should be changed according to the user's preference and (ii) to use the Rosetta files specified for the commands here provided, the user needs to know the file and the path to it. Here we provide the paths to the files at the date June 2020. However, depending on the Rosetta version, these paths may change. If that is the case the user will see an error message when running the command. In order to find the path to the file the command `find -name` *`name_of_the_Rosetta_file_to_be_used`* can be used. Then, they will have to change the path to the file in the commands that we provide.

1. **Access to the computer cluster.** Depending on the computer cluster used, it might be necessary to load certain modules before starting the design itself. Importantly, these modules have to be uploaded every time the user accesses the computer cluster. Also, the user might need to load different modules depending on the computer cluster being used. To know which modules are needed the user needs to read the warning and error messages that will pop-up when running Rosetta. The modules we have used for our design are:

   ```
   module load openmpi #Module to load MPI
   module load sqlite #Module needed for relaxing the protein
   module load gcc #Module needed for running Rosetta
   ```

   Moreover, it is possible that the user needs to add some libraries to the path where they are working. Once more, to know the paths needed, the user should read the warning messages. To add the libraries the command needed is:

   ```
   LD_LIBRARY_PATH=$LD_LIBRARY_PATH:PATH TO THE LIBRARY
   ```

2. **Preparation of the Protein.** Before starting the design, the protein file has to be pre-processed. This pre-processing consists of two steps:

2.1. **Clean the PDB file.** PDB files normally contain water molecules among others, which need to be removed before starting the design. This is done with the following command:

```
rosetta/main/tools/protein_tools/scripts/clean_pdb.py
protein_file_name protein_chains
```

Note that the user has to specify the chains that the processed protein has (e.g. A, B, C…), which can be previously visualized in PyMOL. In the case the protein has two chains (A and B), instead of *protein_chains* the user should write AB. The output files should be one pdb file, which is the clean protein file, and as many fasta files as chains the protein has.

2.2. **Relaxing the protein structure.** The protein structure has to be "relaxed" so the designing of the ligand binding sites can be done. When the user relaxes a protein, basically they sample conformations of a given structure in 3D space to find the lowest-scoring variant. It is recommended to relax the structure many times (at least 10). This will give the user different output files with different scores. The user should work with at least more than 2 of these relaxed files. Also, the user has to consider this job can take some time to finish. In these cases, it is worth creating a .sh script (*protein_relaxing.sh*) to launch it and run things in the background. This means the user can close the session in the computer-cluster and close their computer and the job will still run. Note that depending on the cluster used, the user might have to write a different script. In our case, the cluster used has the workload manager Slurm, which helps to manage resources between different users on the same computer. The script written had the following structure:

```
#!/bin/bash

#SBATCH --time=05:00:00
#SBATCH --mem=4G
#SBATCH --output=relaxing.%j.out

srun mpiexec
rosetta/main/source/bin/relax.mpi.linuxgccrelease -
database rosetta/main/database -s
path_to_clean_file/clean_file -nstruct
number_of_relaxed_structures
```

Importantly, the user can specify different flags (options in command-line programs) to be performed when running the command. Specific flags can be consulted in the official Rosetta documentation (https://www.rosettacommons.org/docs/latest/application_documentation/struc ture_prediction/relax). To run the above script in the background the following command should be run:

```
sbatch protein_relaxing.sh
```

The output files should include as many pdb files as relaxed structures the user wants to get and a scoring file with the scores of all the relaxed structures. For more information on scoring files please refer to the section "Scoring in Rosetta" (Table 1). After performing the relaxation, the protein residue numbers might have changed as a result of this step. The user can look at the differences in residue numbers between the original file and the clean and relaxed file using PyMOL. This change in residue numbers is really important when the user wants to re-design specific residues from the protein. In order to do so, they will have to check what are the numbers of the residues they want to change in the new clean and relaxed file.

3. **Preparation of the Ligand.** Preparation of the ligand is also needed before starting the design. For this preparation, the user will need OpenBabel and Rosetta.

3.1. **Convert the ligand to SDF format and add hydrogens if needed.** In this step, the ligand file is converted to an appropriate format to work with Rosetta and the user can also add hydrogens to the molecule if needed. The hydrogens can also be added depending on the pH the user is expected to work with. To know more about the possible flags that can be used with Obabel visit https://openbabel.org/docs/dev/Command-line_tools/babel.html.

```
obabel ligand_file.format -flags -O
output_ligand_file_1.sdf
```

The output file should be one sdf file with the hydrogens added. The user can use PyMOL or another visualisation software to see the new file.

3.2. **Generate a library of ligand conformers.** In order to do this, the user needs to run:

```
path_to_bcl/bcl molecule:ConformerGenerator -
ensemble_filenames output_ligand_file_1.sdf -
conformers_single_file ouput_ligand_file_2.sdf
```

The output file should be one sdf file with the different conformations of the ligand. The user can use PyMOL or another visualisation software to see the output file.

3.3. **Conversion of the conformer library into a Rosetta-formatted parameters (params) file.** The previous output file cannot be read by Rosetta, that is why the user needs to transform it so it can be used with the Rosetta software in the next steps.

```
rosetta/main/source/scripts/python/public/molfile_to_para
ms.py -n output_ligand_file_3 -p output_ligand_file_3 --
conformers-in-one-file output_ligand_file_2.sdf
```

The user should get three output files: *output_ligand_file_3.params*, *output_ligand_file_3.pdb* and *output_ligand_file_3_conformers.pdb*. These three files are needed during all the design process and are the ones that the user should be working with from now on.

4. **Docking.** The next step is to manually dock the ligand into the binding pocket of the protein. The files used here are the pdb protein file obtained from the preparation of the protein and the pdb ligand file obtained from the preparation of the ligand. Since the docking is performed manually, a different software than Rosetta is used. There are different programs to do this, such as PyMOL or AutoDock. Another software to consider is SwissDock, which does not require any downloads. From our experience, we recommend AutoDock Vina, which is the newest version of the AutoDock software. Note that when docking multiple ligands, separate configuration files and dockings need to be performed. The docked ligands will not be combined in the same file until the end.

For the docking, additional preparation of the files is needed. The ligand needs to be converted to .pdbqt file format. This can be accomplished with OpenBabel in the terminal with the following command:

```
obabel /path/output_ligand_file_3.pdb -O /path/ligand.pdbqt
```

Note that with very large ligands it is harder to achieve good results and these are computationally heavier.

Preparation of the receptor is done in the graphical user interface of Autodock 4.2.

1. Right-click “All molecules”, choose read molecule, choose your receptor file e.g. “*protein.pdb*”, click “Open”.
2. Click “Edit” and “Delete waters” (this is just an extra step, the waters should be removed already from your relaxed protein file).
3. Click “Edit”, choose “Hydrogens” and “Add”.
4. Click “Edit”, choose “Hydrogens” and “Merge non-polar”.
5. Click “Edit”, choose “Charges” and “Compute Gasteiger charges”.

Next, the grid box needs to be set up. The grid box defines in which space the dockings for the ligand will be searched for. If the exact binding site of the ligand is known, try to fit the grid box around that space. The general rule is to have it as small as possible, but not too small. If the binding site of the ligand is not known, blind docking is performed by setting the grid box around the whole protein. However, this is much more difficult and will not give as good results. Additionally, the larger the grid box is, the longer the computations will take. Autodock 4.2 GUI is used to set the grid box and get the coordinates to use while preparing the Autodock Vina configuration file. To open and set the gridbox click “Grid” and then choose “Grid box”. The size and orientation of the grid box can be adjusted so that it covers the binding site or ligand. Make sure to set the spacing to 1 Angstrom before you start adjusting the grid box.

Before running AutoDock Vina the configuration file in .txt format needs to be prepared (*conf.txt*). The easiest way is to define the receptor, ligand, output file, coordinates, and all other parameters in this file. The file could look like this:

```
receptor = protein.pdbqt
ligand = ligand.pdbqt

out = out.pdbqt

center_x = 24.332
center_y = 1.255
center_z = 24.103

size_x = 16
size_y = 18
size_z = 18

exhaustiveness = 10
energy_range = 25
num_modes = 25
weight_hydrogen = -2.4
```

There are a number of parameters that can be defined depending on how many outputs and scores the user wishes to get. More information about the different parameters can be found in the AutoDock Vina documentation (http://vina.scripps.edu/manual.html).

To run AutoDock Vina in the terminal you need the following command (note that all the needed files need to be in the same folder):

```
/path/vina --config conf.txt --log log.txt
```

When ready, results will be in the predefined output file (in this example *out.pdbqt*). To visualize results in PyMOL, the molecules need to be converted to .pdb format. This can be done in Open Babel as mentioned above. When looking at docking results the most vital thing to consider is the visual positioning of the ligand. If you know the binding site of the ligand, you can compare and make sure that the docked ligand is as well aligned as possible (Fig. 1). In the *log.txt* the scores of the run are also available. The binding affinity is displayed for all the conformations and if the binding affinities are too high (> -6 kcal/mol), you might want to consider adjusting the docking parameters. However, always mainly choose the ligand based on how well it is aligned in the known binding site. Multiple runs might be required and it is natural to get different results even when not adjusting the parameters. After choosing the best conformation save it in a separate .pdb file. Then perform all the multiple steps above for all the ligands (separate preparation of the same receptor is not needed). When all ligands are docked, the ligands can be combined into one .pdb file by clicking "Action" and "Copy to Object" in PyMOL.

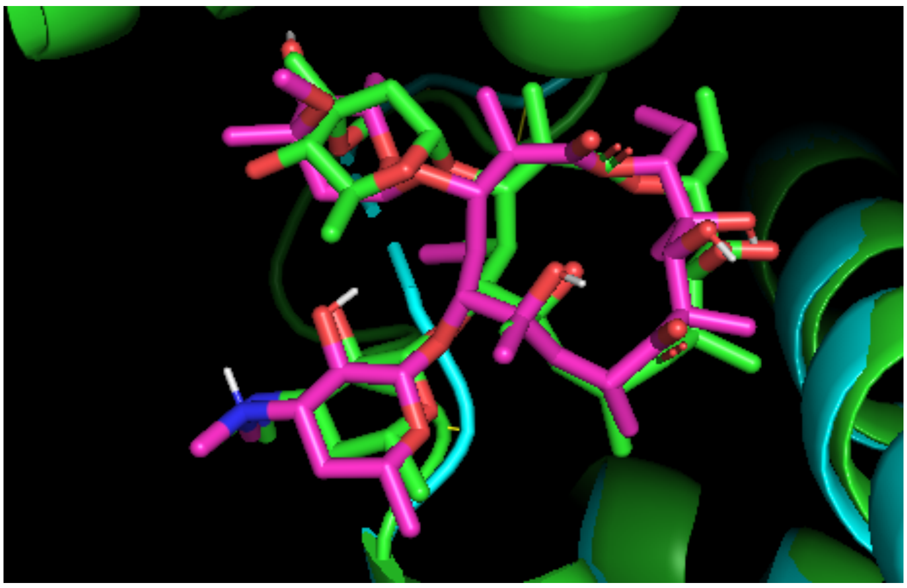

Figure 1. *Erythromycin docked to MphR(A) protein with AutoDock Vina (pink) compared to the crystallized structure of erythromycin bound to MphR(A) protein retrieved from PDB (green).*

5. **PyMOL and revising the Docked Files.** After the docking step, the user can revise the docked files to make sure the docking has been correctly performed. This step is especially important if the protein binds to more than one ligand molecule, since the ligand file will need some modifications. When doing the docking, the ligands are automatically assigned to a chain name (e.g. X, A, B…). If there is only one ligand, the name of the chain it is placed on is X. If there are two, the first ligand is placed in chain X and the second one in chain A and so on. The chain names can be visualized in PyMOL when looking at the sequence: /name of the file/segi(segment-identifier-list)/chain/residue. If the user is only placing one ligand, they do not have to do any modifications to the ligand docked file. On the other hand, when there is more than one ligand molecule, the user has to change the names of the different segi(s) and chains so all the ligands are placed under the same segi and chain names. This chain name will be ideally X, since it is, by default, the first chain name that is always assigned. In order

to do this, the user has to open the docked ligand file in PyMOL and do the following steps:

5.1. Select one of the ligands by clicking on its name in the sequence viewer.

5.2. Change the residue number of the ligand selected with the following command: `alter (sele), resi=2`

5.3. Change the residue chain of the ligand selected running the following command: `alter (chain` *`chain_to_change`*`), chain='X'`

5.4. Remove the segi of the ligand selected with the following command: `alter (segi` *`segi_to_change`*`), segi=''`

5.5. Save the file as a .pdb file: `save` *`file_name`*`.pdb`

In this way, the output file should now be a pdb file with the two or more ligands placed under the same segi and chain names and each one of them having a different residue number.

6. **Design.** This is the main step where the ligand binding sites are re-designed. For this to be done, there are several steps needed.

6.1. **Prepare a residue specification file (*mutations.resfile*).** In this file, the user specifies which residues should be re-designed. The user can allow all residues to mutate or only some of them. It is important to remember that the residues have been renumbered when processing the protein file. The structure of a resfile is the following:

```
Command applied to all residues not specified in the body
AUTO #Use the default behaviour
start #After this command the body starts
Residue_Number Chain_in_Protein Command applied
```

Next, you can see an example of a resfile in which only the natural amino acid is allowed for all the residues not specified in the body (NATAA command) and where the amino acids 52, 59, 85, 86, 93, 116, 137, 144, 145, 234, 241, 267, 268, 275, 298, 319, 326, 327 are allowed to change to all the amino acids except cysteine (ALLAxc). Amino acids 52-145 are in chain A of the protein and amino acids 234-327 in chain B. The alterations to cysteine are not allowed, because cysteine tends to form stable sulphur bonds. Allowing for changes to this amino acid would lead to almost all residues being changed to cysteine, even if it would not be the most optimal mutation. To know more about the possible options when designing a resfile, the user can visit the following

website:
https://www.rosettacommons.org/manuals/archive/rosetta3.5_user_guide/d1/d97/resfiles.html.

```
NATAA
AUTO
start
52 A ALLAxc
59 A ALLAxc
85 A ALLAxc
86 A ALLAxc
93 A ALLAxc
116 A ALLAxc
137 A ALLAxc
144 A ALLAxc
145 A ALLAxc
234 B ALLAxc
241 B ALLAxc
267 B ALLAxc
268 B ALLAxc
275 B ALLAxc
298 B ALLAxc
319 B ALLAxc
326 B ALLAxc
327 B ALLAxc
```

6.2. **Prepare a docking and design script (*design.xml*).** This script will optimize the location of the ligand in the binding pocket, re-design the surrounding sidechains and refine the interactions in the designed context among others. The following script is based on the script provided by Moretti and colleagues (2016). The modifications added are needed for the script to be run in the last release of Rosetta at date June 2020. Between <!-- --> the user can find explanations of some of the commands. Notes for the user: (i) the resfile has to be specified in the body of the script where indicated; (ii) it is important the user knows which scoring function is being used when performing the design, since each Rosetta score function scores the structures (for more information on scoring functions visit https://www.rosettacommons.org/). The scoring functions used here can be found in the command ScoreFunction of the next script:

```
<ROSETTASCRIPTS>
  <SCOREFXNS>
    <ScoreFunction name="ligand_soft_rep"
weights="ligand_soft_rep"/>
    <ScoreFunction name="hard_rep"
weights="ligandprime"/>
```

```
  </SCOREFXNS>
  <TASKOPERATIONS>
    <DetectProteinLigandInterface name="design_interface"
cut1="6.0" cut2="8.0" cut3="10.0" cut4="12.0" design="1"
resfile="mutations.resfile"/>
  </TASKOPERATIONS>
  <LIGAND_AREAS> <!--Describes parameters specific to
each ligand-->
    <!--cutoff: the distance in angstroms from the ligand
an amino-acid’s C-beta atom can be and that residue still
be part of the interface-->
    <!--add_nbr_radius: increases the cutoff by the size
of the ligand neighbor atom’s radius specified in the
ligand .params file-->
    <!--Calpha_restraints: Calpha_restraints greater than
0, backbone flexibility is enabled-->
    <LigandArea name="docking_sidechain" chain="X"
cutoff="6.0" add_nbr_radius="true" all_atom_mode="true"
minimize_ligand="10"/>
    <LigandArea name="final_sidechain" chain="X"
cutoff="6.0" add_nbr_radius="true" all_atom_mode="true"/>
    <LigandArea name="final_backbone" chain="X"
cutoff="7.0" add_nbr_radius="false" all_atom_mode="true"
Calpha_restraints="0.3"/>
  </LIGAND_AREAS>
  <INTERFACE_BUILDERS> <!--Describes how to choose
residues that will be part of the protein-ligand
interface. These residues are chosen for repacking,
rotamer trials, and backbone minimization during ligand
docking-->
    <!--ligand_areas: list of strings matching Ligand
Area names-->
    <!--extension_window: surrounds interface residues
with residues labeled as ‘near interface’. This is
important for backbone minimization, because a residue’s
backbone cannot really move unless it is part of a
stretch of residues that are flexible-->
    <InterfaceBuilder name="side_chain_for_docking"
ligand_areas="docking_sidechain"/>
    <InterfaceBuilder name="side_chain_for_final"
ligand_areas="final_sidechain"/>
    <InterfaceBuilder name="backbone"
ligand_areas="final_backbone" extension_window="3"/>
  </INTERFACE_BUILDERS>
  <MOVEMAP_BUILDERS> <!--Constructs a movemap: A movemap
is a 2xN table of true/false values, where N is the
number of residues of your protein/ligand complex. The
two columns are for backbone and side-chain movements-->
```

```
    <MoveMapBuilder name="docking"
sc_interface="side_chain_for_docking"
minimize_water="true"/>
    <MoveMapBuilder name="final"
sc_interface="side_chain_for_final"
bb_interface="backbone" minimize_water="true"/>
  </MOVEMAP_BUILDERS>
  <SCORINGGRIDS ligand_chain="X" width="25">
    <ClassicGrid grid_name="vdw" weight="1.0"/>
  </SCORINGGRIDS>
  <MOVERS>
    <FavorNativeResidue name="favor_native"
bonus="1.00"/>
    <Transform name="transform" chain="X" box_size="5.0"
move_distance="0.1" angle="5" cycles="500" repeats="1"
temperature="5" rmsd="4.0"/>
    <HighResDocker name="high_res_docker" cycles="6"
repack_every_Nth="3" scorefxn="ligand_soft_rep"
movemap_builder="docking"/>
    <PackRotamersMover name="design_interface"
scorefxn="hard_rep" task_operations="design_interface"/>
    <FinalMinimizer name="final" scorefxn="hard_rep"
movemap_builder="final"/>
    <InterfaceScoreCalculator name="add_scores"
chains="X" scorefxn="hard_rep"/>
    <ParsedProtocol name="low_res_dock">
      <Add mover_name="transform"/>
    </ParsedProtocol>
    <ParsedProtocol name="high_res_dock">
      <Add mover_name="high_res_docker"/>
      <Add mover_name="final"/>
    </ParsedProtocol>
  </MOVERS>
  <PROTOCOLS>
    <Add mover_name="favor_native"/>
    <Add mover_name="low_res_dock"/>
    <Add mover_name="design_interface"/>
    <Add mover_name="high_res_dock"/>
    <Add mover_name="add_scores"/>
  </PROTOCOLS>
</ROSETTASCRIPTS>
```

6.3. **Run the design application.** In this step, the user runs the design of the binding site itself. It is recommended to do from 1000 to 5000 designs of the protein, which means the user will obtain from 1000 to 5000 pdb files with modifications with respect to the original clean and relaxed pdb file. This is computationally heavy and can take several hours even when using a computer cluster. For this reason, we recommend starting with making a script (*launch_design.sh*) that will be run in the background. Note that depending on the cluster used, the user might have to write a different script. In our case, the cluster used has Slurm, and the script written had the following format:

```
#!/bin/bash

#SBATCH --time=20:00:00
#SBATCH --array=0-49
#SBATCH --mem=3100M
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --
output=path_where_the_output_is_wanted/output_job_name.ou
t

NUM=20

srun mpirun rosetta_scripts.mpi.linuxgccrelease -ex1 -ex2
-linmem_ig 10 -restore_pre_talaris_2013_behavior -
parser:protocol design.xml -extra_res_fa
output_ligand_file_3.params -s
"clean_and_relaxed_protein.pdb docked_ligand.pdb" -
nstruct $NUM --out:prefix $SLURM_ARRAY_TASK_ID -
out:file:scorefile
/path_where_the_output_is_wanted/output_design_file.out
```

In the above script, the job has 20 hours to run, and it is assigned to 50 arrays (0-49), each one of which will perform 20 designs. Therefore, the user obtains 1000 pdb files (20x50=1000). To run the script the user has to type the following command:

```
sbatch launch_design.sh
```

After the run, the user should get 1000 pdb files and one score file with the scores of all the pdbs. To see more about scoring go to the next section, "Scoring in Rosetta" (Table 2).

7. **Filtering.** After running the design, the user needs to filter the 1000 pdb files and choose the ones with the score that is more convenient for them. There are two main filtering steps.

7.1. **Prepare a metrics file (*metric_thresholds_1.txt*).** This file specifies the thresholds to use when filtering the output of the design run. It will filter the 1000-5000 pdb previous files. It is important to take into account that this filtering can be done for different parameters. Here we present the parameters we have used in our protein design. To see other parameters that can be used for filtering, refer to Table 2. From our experience, we recommend to use the average values as the cutoff. In order to obtain these values, the user needs to run the command in step 7.2.

```
req total_score value < -1606.71
req if_X_fa_atr value < -50.60
req fa_rep value < 197.15
req if_X_fa_rep value < 14.29
req ligand_is_touching_X value > 0.8
output sortmin interface_delta_X
```

7.2. **Filter on design metrics.** Next, the user uses the previous file to filter the pdb files with the following command:

```
perl
$WRKDIR/rosetta/main/source/src/apps/public/enzdes/Design
Select.pl -d <(grep SCORE output_design_file.out) -c
metric_thresholds_1.txt -tag_column last >
filtered_designs.sc
```

When the user runs this command, in the terminal there will appear the average values for the score metrics. Once the user has these ones, they can be specified in the *metric_thresholds_1.txt* file and run this same command again. The user will get a *filtered_designs.sc* file, which has a list of the scores and the name of the files that have passed the filtering. Next, the user needs to create a file consisting of a list with only the names of the files that have passed the filtering plus the .pdb extension. This is done with the following command:

```
awk '{print $NF ".pdb"}' filtered_designs.sc >
filtered_pdbs.txt
```

7.3. **Calculate additional metrics.** Once the first filtering step has been done, the user needs to calculate additional metrics that are focused on the protein-ligand interface. These metrics are needed to do the second filtering step and calculating them also requires a lot of computational power. Therefore, we recommend making another script (*launch_interfaces.sh*) to be able to run this step on the background of the computer cluster. Note again, that depending on the cluster used, the user might have to write a different script. In our case, the cluster used has Slurm, and the script is the following:

```
#!/bin/bash

#SBATCH --time=08:00:00
#SBATCH --mem=4000M
```

```
#SBATCH --
output=path_where_the_output_is_wanted/output_job_name.ou
t

srun mpirun InterfaceAnalyzer.mpi.linuxgccrelease -
interface AB_X -compute_packstat -pack_separated -
score:weights ligandprime -no_nstruct_label -
out:file:score_only interfaces.sc -l filtered_pdbs.txt -
extra_res_fa output_ligand_file_3.params
```

Importantly, in the previous command the user needs to specify the interface where the additional metrics will be calculated with the flag -interface. The user has to specify the interface of both the protein and the ligand. To do it, they have to write the chain names of the protein (in this case, A and B) followed by an underscore and the chain name of the ligand, which is, by default, X. Once the script is done, the user can launch it with the command:

```
sbatch launch_interfaces.sh
```

The user should obtain one output file, *interfaces.sc*, with additional metrics of the files that passed the first filter. To know more about these additional metrics, please refer to the section "Scoring in Rosetta" (Table 3).

7.4. **Prepare a metrics file (*metric_thresholds_2.txt*).** This file specifies the thresholds to use when filtering the output of the interfaces run. It will filter the already filtered pdb files. It is important to take into account that this filtering step, like the last one, can be done for different parameters. Here we present the parameters we have used in our protein design. To see other parameters that can be used for filtering, please refer to Table 3. From our experience, we recommend to use the average values as the cut-off. In order to obtain these values, the user needs to run the command in step 7.5.

```
req packstat value > 0.58
req sc_value value > 0.48
req delta_unsatHbonds value < 17.88
req dG_separated/dSASAx100 value < -1.53
output sortmin dG_separated
```

7.5. **Filter on interface metrics.** Next, the user does the second filtering step using the file created in 7.4.

```
perl
$WRKDIR/rosetta/main/source/src/apps/public/enzdes/Design
Select.pl -d <(grep SCORE interfaces.sc) -c
metric_thresholds_2.txt -tag_column last >
filtered_interfaces.sc
```

When the user runs this command, in the terminal there will appear the average values for the score metrics. Once the user has these ones, they can specify them in the metric_thresholds_2.txt file and run this same command again. The output file should be the *filtered_interfaces.sc* file, with a list of the scores and the names of the files that have passed the filtering. If the user wants to have a file with only the names of the files that have passed the filtering, they can run the following command:

```
awk '{print $NF ".pdb"}' filtered_interfaces.sc >
filtered_pdbs_final.txt
```

8. **Check the filtered results.** Once the user has filtered the results two times, they will obtain a reduced list of filtered pdbs. It is recommended that the user chooses at least the three best files according to their purposes and analysing the scores obtained. After that, the user can manually inspect the pdb files and compare them with the clean and relaxed protein file with PyMOL to check (i) if there are any modifications in the desired residues and (ii) if the structure of the protein is correct.

9. **Re-Run.** Apart from doing the design of the binding sites with the three best protein files obtained after the preparation of the protein, it is recommended to re-apply the design protocol for at least the three final best pdb files obtained. So, after choosing these files, this protocol needs to be re-applied from point 6 onwards. In this case, when running the script *launch_design.sh*, the names after the flag `-s` should be changed to the name of the selected file. It is recommended to do this re-run from 3 to 5 times. Fig. 2 shows a scheme of the steps that should be followed.

10. **Extract the selected protein sequences into fasta format.** Once the design has been run 3-5 times, the user might one to obtain the mutated protein sequence in fasta format. This can be done by opening the file in PyMOL and running the following command:

```
save file_name.fasta
```

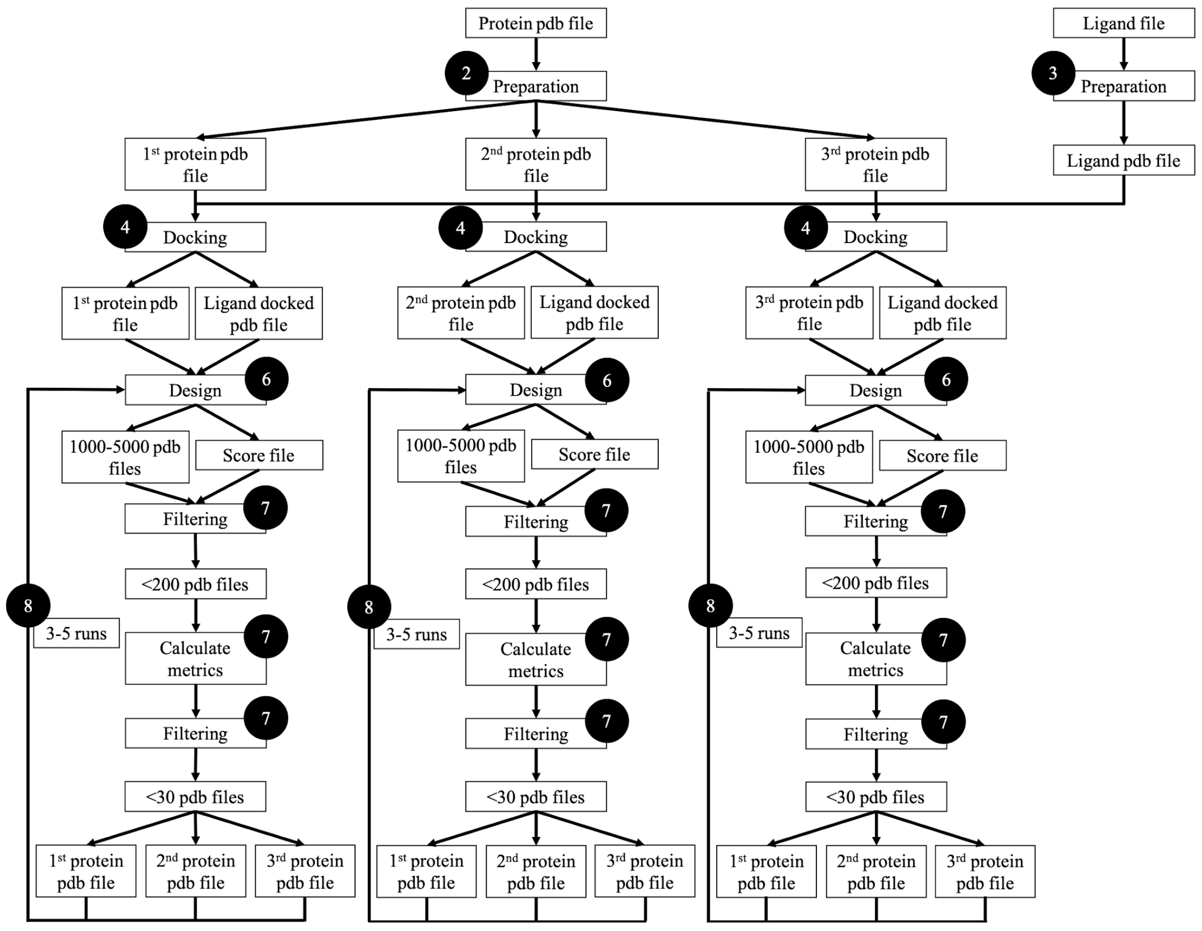

Figure 2. *Steps required when following the Rosetta Guide here presented.*

SCORING IN ROSETTA

When performing the design of ligand binding sites the user gets scoring files for some metrics of the design. These scores are in Rosetta Energy Units (REU), which cannot be converted into physical energy units like kcal/mol, but give the user a relative idea of which files are better than others. In the next tables (Table 1, Table 2 and Table 3) there is an explanation for the scoring metrics the user will encounter when following this manual. These explanations have been gathered from the RosettaCommons webpage (https://www.rosettacommons.org/) and from a manual the iGEM Technion 2016 (http://2016.igem.org/Team:Technion_Israel/Modifications/Rosetta) and iGEM TU Eindhoven 2016 (http://2016.igem.org/Team:TU-Eindhoven/Modeling/Rosetta) wrote during the iGEM competition 2016. If there are not recommended values, from our experience we recommend to take the average values as a cut-off for filtering as explained in section 7. Note that the terms without if_X_ refer to the protein as a whole and the ones with if_X_ refer to the interface of the protein with the sidechain X (by default, the ligands).

Table 1. *Scores obtained when performing protein relaxation.*

Scoring Metric	Explanation	Recommended Value
total_score	Measure of protein stability	The lower the better. Good values are -1/-2/-3 multiplied by the number of residues of the protein evaluated
dslf_fa13	Disulfide geometry potential	---
fa_atr	Lennard-Jones attractive between atoms in different residues	---
fa_dun	Internal energy of sidechain rotamers as derived from Dunbrack's statistics	---
fa_elec	Coulombic electrostatic potential with a distance-dependent dielectric	---
fa_intra_rep	Lennard-Jones repulsive between atoms in the same residue	---
fa_intra_sol_xover4	Intra-residue Lazaridis-Karplus solvation energy	---
fa_rep	Lennard-Jones repulsive between atoms in different residues	---
fa_sol	Lazaridis-Karplus solvation energy	---
hbond_bb_sc	Sidechain-backbone hydrogen bond energy	---
hbond_lr_bb	Backbone-backbone hbonds distant in primary sequence	---
hbond_sc	Sidechain-sidechain hydrogen bond energy	---
hbond_sr_bb	Backbone-backbone hbonds close in primary sequence	---
lk_ball_wtd	Asymmetric solvation energy	---
omega	Omega dihedral in the backbone. A Harmonic	---

	constraint on planarity with standard deviation of ~6 deg	
p_aa_pp	Probability of amino acid, given torsion values for phi and psi	---
pro_close	Proline ring closure energy and energy of psi angle of preceding residue	---
rama_prepro	Ramachandran preferences	---
ref	Reference energy for each aminoacid. Balances internal energy of amino acid tems. Plays role in design	---
yhh_planarity	A special torsional potential to keep the tyrosine hydroxyl in the plane of the aromatic ring	---
description	Name of the file that is being assessed	---

Table 2. *Scores obtained when performing the design of the ligand binding site(s).*

Scoring Metric	Explanation	Recommended Value
total_score	Measure of protein stability	The lower the better. Good values are -1/-2/-3 multiplied by the number of residues of the protein evaluated
Grid_score	Score of the grid	---
Transform_accept_ratio	How well the low resolution Monte Carlo stage worked.	0-1. 0,3 is ideal but anything that is not 0 or 1 is good.
(if_X_)angle_constraint	Angle between Atom2 → Atom1 vector and Atom2 → Atom3 vector …; the angle is measured in radians	---
(if_X_)atom_pair_constraint	Harmonic constraints between atoms involved in Watson-Crick base pairs specified by the user in the params file	---

(if_X_)chainbreak	A quadratic penalty on distance of the chain-ends	---
(if_X_)coordinate_constraint	How well the design fits the coordinate constraints	---
(if_X_)dihedral_constraint	Constraints that the dihedral angles defined be Atom1 → Atom2 → Atom3… be identical	---
(if_X_)dslf_ca_dih	CA dihedral score in current disulfide	---
(if_X_)dslf_cs_ang	CS angles score in current disulfide	---
(if_X_)dslf_ss_dih	Dihedral score in current disulfide	---
(if_X_)dslf_ss_dst	Distance score in current disulfide	---
(if_X_)fa_atr	Lennard-Jones attractive between atoms in different residues	---
(if_X_)fa_dun	Internal energy of sidechain rotamers as derived from Dunbrack's statistics	---
(if_X_)fa_elec	Coulombic electrostatic potential with a distance-dependent dielectric	---
(if_X_)fa_pair	Statistical residue-residue interaction potential	Negative results
(if_X_)fa_rep	Lennard-Jones repulsive between atoms in different residues	---
(if_X_)fa_sol	Lazaridis-Karplus solvation energy	---
(if_X_)hbond_bb_sc	Sidechain-backbone hydrogen bond energy	---
(if_X_)hbond_lr_bb	Backbone-backbone hbonds distant in primary sequence	---

(if_X_)hbond_sc	Sidechain-sidechain hydrogen bond energy	---
(if_X_)hbond_sr_bb	Backbone-backbone hbonds close in primary sequence	---
interface_delta_X	The energy of interaction between the ligand (chain X) and the protein	---
ligand_is_touching_X	How close is the ligand to the protein	0-1. 0 If the ligand is far from the protein and 1, otherwise. 1 in most cases
(if_X_)omega	Omega dihedral in the backbone. A Harmonic constraint on planarity with standard deviation of ~6 deg	---
(if_X_)p_aa_pp	Probability of amino acid, given torsion values for phi and psi	---
(if_X_)pro_close	Proline ring closure energy and energy of psi angle of preceding residue	---
(if_X_)ref	Reference energy for each aminoacid. Balances internal energy of amino acid tems. Plays role in design	---
(if_X_)res_type_constraint	How close are the results to the native sequence	---
total_score_X	Total ligand grid score	---
vdw_grid_X	---	---
description	Name of the file that is being assessed	---

Table 3. *Scores obtained when calculating additional metrics on the protein-ligand interface.*

Scoring Metric	Explanation	Recommended Value
total_score	Measure of protein stability	---
complex_normalized	Average energy of a residue	---

	in the entire complex	
dG_cross	Binding energy calculated in an “inaccurate” environment insensitive way	---
dG_cross/dSASAx100	Binding energy calculated in an “inaccurate” environment insensitive way divided by SASA (Solvent Accessible Surface Area) multiplied by 100. dSASA controls for large interfaces having more energy.	---
dG_separated	Binding energy calculated in an “accurate” way by separating the chains and optionally repacking	---
dG_separated/dSASAx100	Binding energy calculated in an “accurate” way by separating the chains and optionally repacking divided by SASA (Solvent Accessible Surface Area) multiplied by 100. dSASA controls for large interfaces having more energy.	---
dSASA_hphobic	Hydrophobic SASA (Solvent Accessible Surface Area)	---
dSASA_int	The SASA (Solvent Accessible Surface Area) buried at the interface, in square Angstroms	---
dSASA_polar	Polar SASA (Solvent Accessible Surface Area)	---
delta_unsatHbonds	How many unsatisfied bonds are introduced during the metrics calculation. These bonds are hydrogen bonds that atoms would be able to make but do not do because they are blocked by other residues	---
hbond_E_fraction	Amount of interface energy	---

	(dG_separated) accounted for by cross interface H-bonds	
hbonds_int	Total cross-interface hydrogen bonds found	---
nres_all	Total number of residues in the complex	Number of residues in the PDB file
nres_int	Total number of residues in the interface	Number of residues in the interface
packstat	Packing statistic score for the interface	0-1, the higher the better
per_residue_energy_int	Average energy of each residue at the interface	---
sc_value	Shape complementarity	0-1. The higher, the better.
side1_normalized	Average per-residue energy on one side of the interface	---
side1_score	Energy of one side of the interface	---
side2_normalized	Average per-residue energy on the other side of the interface	---
side2_score	Energy of the other side of the interface	---
description	Name of the file that is being assessed	---

CONCLUSIONS

Designing ligand binding sites is a complex process that needs several steps. All the previous existing guides for designing ligand binding sites are complete, but we have found they require some basic computational knowledge the user might lack. With this guide, we hope to help users less experienced in the computational field to be able to design ligand binding sites. If this is the first time the user uses the Rosetta software, it might be that they encounter some difficulties during the installation and during the whole designing process. We recommend starting a similar project in advance.

REFERENCES

- "Autodock Vina - Molecular Docking And Virtual Screening Program". 2020. Vina.Scripps.Edu. http://vina.scripps.edu/.
- "Computational Design Of Ligand Binding Proteins". 2016. Methods In Molecular Biology. doi:10.1007/978-1-4939-3569-7.
- "Mgltools Website - Mgltools". 2020. Mgltools.Scripps.Edu. http://mgltools.scripps.edu/.
- "Open Babel". 2020. Openbabel.Org. https://openbabel.org/.
- "Pymol | Pymol.Org". 2020. Pymol.Org. https://pymol.org/2/.
- "UCSF Chimera Home Page". 2020. Cgl.Ucsf.Edu. https://www.cgl.ucsf.edu/chimera/.
- "Xquartz". 2020. Xquartz.Org. https://www.xquartz.org/.
- "Team:Technion Israel/Modifications/Rosetta - 2016.Igem.Org". 2020. *2016.Igem.Org*. http://2016.igem.org/Team:Technion_Israel/Modifications/Rosetta.
- "Team:TU-Eindhoven/Modeling/Rosetta - 2016.Igem.Org". 2020. *2016.Igem.Org*. http://2016.igem.org/Team:TU-Eindhoven/Modeling/Rosetta.
- The Rosetta Software | Rosettacommons". 2020. *Rosettacommons.Org*. https://www.rosettacommons.org/software/.
- Moretti, R., J. Bender, B., Allison, B., & Meiler, J. (2016). Rosetta and the Design of Ligand Binding Sites, *1414*, 47–62. https://doi.org/10.1007/978-1-4939-3569-7