VI.5 Appendix V - Computational Standard Operating Procedures

Here I will detail the computational procedures including running ROSETTA and analyzing data with scripts that are often available in the Meiler Lab Scripts repository or available on request. Also a lot of the procedures are detailed in IPython Notebooks and will also be available on request. **Using ROSETTA version 80616601370**

VI.5.1 Chapter I - Multi-State Design

Here I will detail how I ran ROSETTADESIGN for multi-state design and how I analyzed the results. I will use the simplified example of IGH_V5-51 which only contains three sets of molecular structures. There is a great protocol capture for complex procedures attached to the publication by Andrew Leaver-Fay showing how to *design for* and *design against* in multiple states (Leaver-Fay et al., 2011a).

VI.5.1.1 Running ROSETTA Multi-State Design

To run multi-state design, I have to prepare several files.

- Entity File - A file containing the amount of residue positions to design as well as instructions for the packer to behave on all proteins. For example, we could want the packer only to use certain rotamers around the interface. This could be handled with the entity file.
- Correlation File - Tells how residues correlate to each other. For example, residue 1 on protein A should be designed with residue 2 on protein B etc.
- Secondary Residue File - This is a residue file as defined in the documentation, but will only instruct the packer to operate on each protein individually. Every state must have it's own secondary residue file.
- Fitness File - A master file containing all other files as well as instructions for the fitness function.

Clean PDB - First, I can download all three protein PDBs with the clean_pdb.py script. Clean PDB supports the following syntax:

```
clean_pdb.py <PDB_ID> <CHAINS>
```

We only want the asymmetric unit in the crystal structure, so it helps to manually inspect the PDBs. We need one heavy chain, one light chain, and one antigen. I just go to www.pdb.org to find these chain codes.

```
clean_pdb.py 2B1A HLP
clean_pdb.py 2XWT ABC
clean_pdb.py 3HMX HLB
```

Although not absolutely necessary, it makes it easier to label the chain IDs the same. All heavy chains have H, light L, and antigen A. There is a change PDB id script which allows us to quickly rename chain IDs. The script takes the following syntax.

```
set_pdb_chain_id.py old_chain new_chain input output
```

I have looked through all the PDBs and figured out which names to change.

```
set_pdb_chain_id.py P A 2B1A_HLP.pdb 2B1A_HLA.pdb
set_pdb_chain_id.py A H 2XWT_ABC.pdb 2XWT_HBC.pdb
set_pdb_chain_id.py B L 2XWT_HBC.pdb 2XWT_HLC.pdb
set_pdb_chain_id.py C A 2XWT_HLC.pdb 2XWT_HLA.pdb
set_pdb_chain_id.py B A 3HMX_HLB.pdb 3HMX_HLA.pdb
```

To ensure the starting structures use the correct numbering scheme, we should renumber each chain starting with 1.

```
renumber_pdb.py 2B1A_HLA.pdb 2B1A_clean.pdb
renumber_pdb.py 2XWT_HLA.pdb 2XWT_clean.pdb
renumber_pdb.py 3HMX_HLA.pdb 3HMX_clean.pdb
```

Now we can remove all temporary files. Only *_clean.pdb files should remain in the working directory. The next thing to do would be to find all positions with at least one difference. This requires manual inspection of the alignment. For the V_H gene, there are 29 amino acid positions that will differ from germline in at least one position. These positions will be considered. Given that data, we can construct the entity residue file.

```
#The entity.resfile
#The number of positions to design
29
#Allow all amino acids
#except cystine and use rotamer libraries 1,2
#and aromatic 2.
ALLAAxc EX 1 EX 2 EX ARO 2
#beginning of residue file
start
```

The correlation file maps how each residue in each file should map to the others. There will be three correlation files, one for each state. Since each amino acid lines up, i.e. design position 5 in 2B1A with position 5 in 3HMX, all the correlation files will be the same. Here is an example of one correlation file.

```
#all.corr
#The first column is the entity,
#the second is the residue number for that state,
#the last is the chain.
1 5 H
2 14 H
3 16 H
4 23 H
5 24 H
6 29 H
7 30 H
8 31 H
9 32 H
```

```
10 34 H
11 40 H
12 46 H
13 48 H
14 51 H
15 52 H
16 54 H
17 58 H
18 65 H
19 70 H
20 72 H
21 74 H
22 76 H
23 77 H
24 80 H
25 84 H
26 88 H
27 93 H
28 97 H
29 98 H
```

A secondary residue file is also needed in case any extra packing tasks are needed to be supplied to each state. For example, we could tell one PDB state to design around the interface in single state design mode while everything in the correlation file designs together. I do not require extra design tasks for this protocol so all secondary residue files will be the same. For example,

```
#tells the packer to use all natural side
#chain configurations for everything
#that is not being designed.
NATRO
#the input side chain is allowed
use_input_sc
#start the residue file
start
```

We next to create a states file that has the PDB, correlation file and secondary resfile names in it. Name them 2B1A.states, 2XWT.states, and 3HMX.states. They should look like the following when opened.

```
#2B1A.states
input_files/2B1A_clean.pdb input_files/all.corr input_files/
    all.2res

#2XWT.states
input_files/2XWT_clean.pdb input_files/all.corr input_files/
    all.2res
```

```
#3HMX.states
input_files/3HMX.pdb input_files/all.corr input_files/all.2
    res
```

Lastly, a fitness file needs to be constructed to tell multi-state how to design. I call this file fitness.daf and it points to the locations of the states files.

```
#initialize the states and what the states file name is
STATE_VECTOR A input_files/2B1A.states
STATE_VECTOR B input_files/2XWT.states
STATE_VECTOR C input_files/3HMX.states


#tell design to minimize energy for each state
SCALAR_EXPRESSION best_A = vmin( A )
SCALAR_EXPRESSION best_B = vmin( B )
SCALAR_EXPRESSION best_C = vmin( C )

#Fitness - design to minimize all energies simultaneously
FITNESS best_A + best_B + best_C
```

Running Rosetta Multi-State Design

The ROSETTAexecutable is called mpi_msd.mpi.<operatingsystem>. It must be compiled in MPI mode as each state is assigned to a processor. The command line takes the following options.

- entity_resfile - The resfile that we created in the input portion
- fitness_file - The fitness file we created in the input portion
- ms::pop_size - How many sequences to keep in memory at once (100 is a good number)
- ms::generation - How many sequence generations should MSD go through see (Leaver-Fay et al., 2011a) to find see how the genetic algorithm selects sequences.
- ms::numresults - How many results to output. Will output top N sequences.
- ms:fraction_by_recombination - How often should a cross-over even take place between sequences in the population. Read (Leaver-Fay et al., 2011a) for details on the genetic algorithm.
- database - The location of the database.

I construct an options file with all those options (options.txt) that looks like this.

```
-entity_resfile entity.resfile
-fitness_file fitness.daf
```

```
-ms
    -pop_size 100
    -generations 435
    -numresults 100
    -fraction_by_recombination .04
-database my/rosetta/database/location/
```

Finally we can run ROSETTA using the following command after starting MPD.

```
mpd && mpiexec -n 4 /my/rosetta/location/mpi_msd.mpi.
   myoperatingsystem \
@input_files/options.txt
```

Warning!
ROSETTA may complain about some of the comments (anything starting with #) not being recognized, if so, just remove it from the file
The output will be 300 files, 100 for each of the states. We only need to analyze 100 files considering that the designed entities will be the same for all three files. For example, position 5 will be the same for 2B1A, 2XWT, and 3HMX.

VI.5.1.2 Analysis of MSD Output

I wrote a design analysis script called design_analysis.py which encompasses many design analysis tools. I will only go into the functionality that is necessary to use, but you can read the options file for more use.

```
design_analysis.py --help
>>                Design Analysis
-------------------------------------------------------------

This script is intented to encompass the entire
   functionality
of design analysis. Everything you could want to do with
   design
is called upon in this script. The most basic functionality
   is
to pass a list of pdbs or get a position matrix of
   occurences
count of just one line.

The functionality extends from there by giving bitscores,
changes in energy, giving position specific scoring matrices
of your design, giving a customizable sequence logo. This is
   a
combination of many scripts and classes.

optional arguments:
  -h, --help show this help message and exit
```

```
Necessary:
  PDB files have to be included
  *.pdb  The PDB files to be analyzed

Recommended Options:
  Will give you a more complete analysis based on a res file
    ,
  and a native pdb to compare it to.

  --native_pdb N_PDB, -p N_PDB
        The native pdb file to compare against
  --corr corr.corr, -c corr.corr
        Get the results defined only in the corr file
  --res resfile.resfile, -r resfile.resfile
        Get the results defined only in the residue file

Output Options: Please read carefully:
  These arguments change file name, which file is printed,
  which is output to a dictionary,and give verbose printing

  --verbose, -v
        everything printed to a file will also be shown
        on the screen
  --prefix PREFIX, -P PREFIX
        The prefix for what all the output files will be
  --score_files O_FILES [O_FILES ...], -s
    What do you want output to a file?
    Can list as a space seperated (eg -s n d nd):
    a - full analysis dict
    d - give analysis of just designed residues
    n - just the native residues scores are shown
    nd - just the native residues of the residues designed
            Defaults to full analysis dictionary
  -b Should the output be in bit score?
        Defaults to occurences instead of bitscore
  -S If you specify a native file and a design file,
        it will give you an output of the stats of the
          design

Rosetta Energy Analysis:
  Options for outputing options about energy scores,
  the dictionaries analyzed depend on what you asked
  for using the -s output options flag
```

```
  --rosetta, -t  This option will output a .csv file of the
  model,chain,residue,residue number,and rosetta scores.

Bit Score Options:
  options for bitscore metric for each designed residue

  -n     do you want the bit scores to be
         normalized by the shannon entropy

Sequence Logos Options:
  These options handle the sequence logos that can be output
  from the design analysis script, and uses the api of
     weblogo
  to do so.

  --seq, -l
    Turn on Sequence Logos for all the dictionaries
    you supply given in an .eps file
  --path LOGO\_PATH, -lp LOGO\_PATH
    What is the path to weblogo software?
    Defaults to meilerlab enviroment
  --format {eps,jpg,png,png_print,pdf,jpeg,svg,logodata}
    What format do you want the sequence logo in?
  --units {bits,nats,kt,kJ/mol,kcal/mol,probability}
    What do you want the units of the sequence
    logo to be in? Defaults to bits.
  --stacks S_STACKS
    How many sequences per line in the logo, default=after
    forty letters it will go to a new line.
  --stack_width S_STACK_WIDTH
    How wide is each stack in the logo. Value of 25 is
       useful
    for x-axis labels >3 characters and 30
    for labels as 'sequence_numbers'.
  --title S_TITLE
    The title of your sequence logo
  --x_label S_X_AXIS
    What do you want the x axis titled?
  --y_axis_height S_Y_HEIGHT
    How high do you want the Y axis,
    currently 4.32 which is the maximum
    acheivable score in a unbiased design
  --y_label S_Y_LABEL
    Title of Y-Axis
  --error_bars S_Y_ERROR
```

```
    Do you want error bars turned on, YES/NO?
  --fine_print S_FINE
    Fine Print
  --color_scheme
    {auto,chemistry,charge,classic,
    hydrophobicity,monochrome}
    The color scheme of the sequence logo.
    Defaults to Classic
  --labels {sequence,numbers,sequence_numbers}
    The x-axis labels can either take on the
    native residues sequence given with a native
    pdb file or the numbering of the pdb residue.
  --debug
    Get the full command line of what was put into weblo
```

It's obvious that this analysis can do a lot, but I will stick with the basics. First a new input that is completely germline. We can do this with the packer. There is a script called make_res.py which will make a residue file from a FASTA file. I use this to take the germline sequence and thread it over one of my inputs. Then that input can be used as our template.

```
>IGHV5-51*01
EVQLVQSGAEVKKPGESLKISCKGSGYSFTSYWIGWVRQMPGKGLEW
MGIIYPGDSDTRYSPSFQGQVTISADKSISTAYLQWSSLKASDTAMY
YCAR
```

Then we can use the make_res.py script.

```
make_res.py IGHV5-51.fasta > IGHV5-51.res
```

This residue file can then be used to mutate one of our templates back to germline.

```
/path/to/rosetta/bin/fixbb.default.<operating_system> \
-s 2B1A_clean.pdb \
-resfile IGHV5-51.res -database /path/to/database \
-o 2B1A_germline.pdb
```

Now we can use the analysis script from the analysis directory

```
../analysis/design_analysis -p ../input_files/2B1A_clean.pdb
    \
-S -b -s nd -c ../input_files/all.corr
../output_files/msd_output_*A*.pdb

>>
#score vs mature
Total Bit Score of Design ===> 28.4534
Total Shannon Entropy of Design ===> 115.9577
Normalized Bit Score for design ===> 0.2454
```

and

```
../scripts/design_analysis -p analysis/2B1A_germline.pdb \
-S -b -s nd -c ../input_files/all.corr
../output_files/msd_output_*A*.pdb

>>
#score vs mature
Total Bit Score of Design ===> 50.2318
Total Shannon Entropy of Design ===> 115.9577
Normalized Bit Score for design ===> 0.4337
```

This gives a design score towards 0.35 for the germline sequence and 0.24 for the mature sequence. Everything can be repeated this exact way. For single state design you can keep the fitness file exactly the same, but remove the other states.

```
#initialize the states and what the states file name is
STATE_VECTOR A 2b1a.states
STATE_VECTOR B 2xwt.states
STATE_VECTOR C 3hmx.states


#tell design to minimize energy for each state
SCALAR_EXPRESSION best_A = vmin( A )
SCALAR_EXPRESSION best_B = vmin( B )
SCALAR_EXPRESSION best_C = vmin( C )

#Fitness - design to minimize all energies simultaneously
FITNESS best_A
```

And run the exact same procedures.

VI.5.2 Chapter II - Database and Design

This section accompanies chapter II. I will go over uploading the sequences to a database, selecting the correct sequences, threading, and finally design.

VI.5.2.1 Sequence Analysis

The methods section detailed how I actually sequence the amplicons from 64 healthy donors, but processing them takes quite a bit of computational work. The VANTAGE core at Vanderbilt returns the sequencing runs as two paired end reads in FASTQ format. They must be "stiched" together to make one read. For example, for donor 10, there are "2185-RC-10_1.fastq" for the forward read and "2185-RC-10_2.fastq" for the reverse read. I use a stitching algorithm called "pandaseq" to process theses (Bartram et al., 2011). These commands are incredibly simple to run.

```
/usr/local/bin/pandaseq -f 2185-RC-10_1.fastq -r 2185-RC-10
    _2.fastq -T 23 > donor_10.fasta
```