Protocol capture

Introduction

The following is a protocol capture of how to run the parallelized RECON multistate design method. It will review an example of designing the influenza antibody C05 against a panel of influenza antigens and evaluating the models. For simplicity's sake we have condensed the protocol to a set of five antigens rather than the 13 discussed in the manuscript.

All Rosetta commands for this publication were run with version 3e41de71be009712db5ba0f3b0cd1080a1603181, from March 2016. Note that all analysis scripts will only function properly if they are in the correct directory as provided.

All materials from this protocol capture can be downloaded from https://github.com/sevya/parallelized_RECON_protocol_capture

Dependencies:

Several scripts require the use of Python 2.7 as well as the Biopython package (https://github.com/biopython/biopython.github.io/). We recommend installing all necessary packages before beginning this protocol.

Structure preparation

First, the C05 Fab structure (PDB ID 4fnl) was downloaded from the Protein DataBank (PDB; www.rcsb.org) and manually processed in PyMol. All waters were removed from the structure and non-protein residues were also removed, and all chains but H and L were deleted. The antibody was processed to remove the constant domain of the Fab fragment – this was done to reduce the total size of the system and simulation time. In this case, residues 114 – 214 on chain H and residues 108 – 213 on chain L were removed. The complex was then saved to a PDB file. Next the structures of the antigenic proteins of interest were downloaded from the PDB. In this example we will use five H1 structures as a test case – PDB IDs 1rvx, 1ruy, 4hkx, 3ubq, and 4lxv. These five structures were downloaded from the PDB and waters and non-protein residues were removed. In addition for each antigen all chains except for one HA1 monomer were deleted. The HA1 subunit was also truncated to the head domain, based on the start and end points of structure 4hkx. The HA1 subunits were renamed so that the chain ID is A, so that the chain IDs would be uniform between different complexes, using the following command:

```
alter 1rvx, chain='A'
```

This command was repeater for all five HAs. Mock co-complexes of C05 in complex with each of these antigens were created by aligning to the known co-crystal structure of C05 in complex with an H3 antigen from PDB ID 4fp8. After downloading the 4fp8 structure and deleting all but one copy in the asymmetric unit, the C05 Fab structure was aligned to the antibody in the 4fp8 co-complex and the antigens were aligned to the antigen in the 4fp8 co-complex. Alignments were done using the following commands:

```
super 4fnl, 4fp8
super 1rvx, 4fp8
super 1ruy, 4fp8
super 4hkx, 4fp8
super 3ubq, 4fp8
super 4lxv, 4fp8
```

Next, mock co-complexes were created and saved to use in multistate design. We used the `create` command in PyMol to create an object for each complex, which then can be save and processed in ROSETTA. An example command is shown below:

```
create C05_1rvx, 4fnl or 1rvx
save C05_1rvx.pdb, C05_1rvx
```

This command was repeated for all antigens and the new complexes were saved. Also save the C05 Fab structure for use later using the command:

```
save C05.pdb, 4fnl
```

These complexes were renumbered using a python script. The following script both renumbers the PDB chains and reorders the chains, so that chains H and L come first in the file. Note that this does not change any of the atomic coordinates in the structure, only reorders them. Run this command for all antigens used in the panel, as well as for the C05 apo structure.

```
python reorder_pdb_chains.py --new_chain_order H,L,A
C05_1rvx.pdb C05_1rvx_renum.pdb

python reorder_pdb_chains.py --new_chain_order H,L C05.pdb
C05_renum.pdb
```

Refinement of input structures

After preparing and saving the complexes to be used for design, we first want to refine them to prevent small clashes from introducing artifacts into the score function. We will use Rosetta relax with restraints to the backbone coordinates to do a subtle refinement. The restraints are placed on all Cα atoms with a standard deviation of 1 Å. We use the following command, XML and options file to run the restrained relax. We must also make the output folder for our models to go in.

```
mkdir C05_templates_relaxed

/path/to/Rosetta/main/source/bin/rosetta_scripts.default.linuxgc
crelease @relax.options -s C05_1rvx_renum.pdb C05_1ruy_renum.pdb
C05_4hkx_renum.pdb C05_3ubq_renum.pdb C05_4lxv_renum.pdb

relax.options:
-in:file:fullatom
-out:file:fullatom
```

```
-database /path/to/Rosetta/main/database
-out:pdb_gz
-ex1
-use_input_sc
-nstruct 1
-out:path:pdb C05_templates_relaxed/
-parser:protocol relax_cst.xml

relax_cst.xml:
<ROSETTASCRIPTS>
    <SCOREFXNS>
        <ScoreFunction name="talaris_rwt"
weights="talaris2013.wts" >
                <Reweight scoretype="coordinate_constraint"
weight="1.0" />
          </ScoreFunction>
     </SCOREFXNS>
     <FILTERS>
     </FILTERS>
     <TASKOPERATIONS>
          <InitializeFromCommandline name="ifcl"/>
          <RestrictToInterfaceVector name="rtiv"
chain1_num="1,2" chain2_num="3" CB_dist_cutoff="10.0"
nearby_atom_cutoff="5.5" vector_angle_cutoff="75"
vector_dist_cutoff="9.0" />
          <RestrictToRepacking name="rtr"/>
     </TASKOPERATIONS>
     <MOVERS>
          <FastRelax name="relax" task_operations="ifcl,rtr"
scorefxn="talaris_rwt" />
          <ddG name="ddg" per_residue_ddg="0" repack_unbound="1"
chain_num="3" task_operations="rtiv,ifcl,rtr"
scorefxn="talaris2013" />
          <AtomCoordinateCstMover name="cst" />
          <VirtualRoot name="root" />
     </MOVERS>
     <APPLY_TO_POSE>
     </APPLY_TO_POSE>
     <PROTOCOLS>
          <Add mover_name="root" />
          <Add mover_name="cst" />
          <Add mover_name="relax"/>
          <Add mover_name="ddg" />
     </PROTOCOLS>
     <OUTPUT scorefxn="talaris2013" />
</ROSETTASCRIPTS>
```

Multistate design

After refining the structure we are ready to run multistate design. To define the residues which will be included in design, we used a script to calculate residues within a 5 Å cutoff of the antigen. This script calculates residues in contact with the antigen using the 4hkx complex as a template. In addition it will calculate residues on the antigen which are in

contact with the antibody and designate these residues for repacking. Note that we use the original mock co-complex to calculate contact residues, not the refined structure. The script is shown below:

```
python define_interface.py --side1 HL --side2 A --design-side 1
--repack --output C05 C05_4hkx_renum.pdb
```

In addition we will generate a repacking only resfile that will identify the same residues as the previous resfile, but will designate all residues as repack only.

```
python define_interface.py --side1 HL --side2 A --design-side 1
--repack --native --output C05_repack C05_4hkx_renum.pdb
```

At this point the files are prepared for multistate design of the five complexes. First we will make an output folder for the models to go into. Then we will run multistate design with the following command.

```
mkdir models/

mpiexec -n 5 \
/path/to/Rosetta/main/source/bin/rosetta_scripts.mpi.linuxgccrel
ease @msd.options -l templates.list

templates.list:
C05_templates_relaxed/C05_1rvx_renum_0001.pdb.gz
C05_templates_relaxed/C05_1ruy_renum_0001.pdb.gz
C05_templates_relaxed/C05_4hkx_renum_0001.pdb.gz
C05_templates_relaxed/C05_3ubq_renum_0001.pdb.gz
C05_templates_relaxed/C05_4lxv_renum_0001.pdb.gz

msd.options:
-in:file:fullatom
-out:file:fullatom
-database /path/to/Rosetta/main/database/
-out:pdb_gz
-use_input_sc
-ex1
-run:msd_job_dist
-nstruct 50
-mute protocols.simple_moves.GenericMonteCarloMover
-parser:protocol msd_brub.xml
-scorefile msd.fasc
-out:path:pdb models
-nstruct 100
-out:suffix _msd_rlx

msd_brub.xml
<ROSETTASCRIPTS>
     <SCOREFXNS>
          <ScoreFunction name="talaris_rwt"
weights="talaris2013_cst.wts" />
```

```
     </SCOREFXNS>
     <TASKOPERATIONS>
          <InitializeFromCommandline name="ifcl" />
          <RestrictToRepacking name="rtr" />
          <RestrictToInterfaceVector name="rtiv"
chain1_num="1,2" chain2_num="3" CB_dist_cutoff="10.0"
nearby_atom_cutoff="9.0" vector_angle_cutoff="75"
vector_dist_cutoff="9.0" />
          <ReadResfile name="repackable"
filename="C05_repack.resfile" />
     </TASKOPERATIONS>
     <MOVERS>
          <Backrub name="backrub_man" pivot_residues="106-118"
/>
          <GenericMonteCarlo name="backrub"
mover_name="backrub_man" scorefxn_name="talaris2013"
trials="500" temperature="0.8" recover_low="1" />

          <PackRotamersMover name="design"
scorefxn="talaris_rwt" task_operations="ifcl" />

          <MSDMover name="msd1" design_mover="design"
post_mover="backrub" constraint_weight="0.5"
resfiles="C05.resfile" debug="0" />
          <MSDMover name="msd2" design_mover="design"
post_mover="backrub" constraint_weight="1"
resfiles="C05.resfile" debug="0" />
          <MSDMover name="msd3" design_mover="design"
post_mover="backrub" constraint_weight="1.5"
resfiles="C05.resfile" debug="0" />
          <MSDMover name="msd4" design_mover="design"
post_mover="backrub" constraint_weight="2"
resfiles="C05.resfile" debug="0" />

          <FindConsensusSequence name="finish"
scorefxn="talaris2013" resfiles="C05.resfile" debug="1"
task_operations="ifcl,repackable" repack_one_res="1" />

          <FastRelax name="rlx" task_operations="ifcl,rtr,rtiv"
scorefxn="talaris_rwt" />

          <FavorSequenceProfile name="fnr" pdbname="C05_H.pdb"
weight="0.25" scaling="prob" matrix="IDENTITY" />
          <ClearConstraintsMover name="clear_cst" />

          <InterfaceAnalyzerMover name="ddg"
scorefxn="talaris2013" packstat="0" pack_input="0"
pack_separated="1" fixedchains="H,L" />

          <AtomCoordinateCstMover name="cst" coord_dev="1.0" />
          <VirtualRoot name="root" removable="1" />
          <VirtualRoot name="rmroot" remove="1" />
     </MOVERS>
     <FILTERS>
```

```
          <FitnessFilter name="fitness" output_to_scorefile="1"
/>
     </FILTERS>
     <PROTOCOLS>
          <Add mover="fnr" />
          <Add mover="msd1" />
          <Add mover="msd2" />

          <Add mover="msd3" />
          <Add mover="msd4" />
          <Add mover="finish" />

          <Add mover="root" />
          <Add mover="cst" />
          <Add mover="rlx" />
          <Add mover="rmroot" />

          <Add mover="ddg" />

          <Add filter="fitness" />

     </PROTOCOLS>
     <OUTPUT scorefxn="talaris2013" />
</ROSETTASCRIPTS>
```

This protocol will run in parallel over 5 processors and generate 100 output decoys. The protocol will run multistate design followed by a restrained relax to refine the designed complexes, and a step to calculate the fitness, which is calculated as the sum of energy over all input states.

To analyze the sequence profile of designs we used the WebLogo tool. The following script will create a fasta file with the sequences of all designs, as well as a sequence logo summarizing the results. Note that we only need to analyze the designs from one state, since analyzing the designs from each state would be redundant. For example, the designed sequences from trajectory 1 will be identical in the 1rvx complex, 1ruy complex, etc. Here we will only analyze sequences in the 1rvx complex.

```
design_analysis.py --native C05_renum.pdb --format eps --resfile
C05.resfile --multiproc --units probability
models/C05_1rvx*pdb.gz
```

RosettaCM comparative modeling

In addition, in this manuscript we describe design against a panel of modeled HAs generated by RosettaCM comparative modeling. To do this we first downloaded HA sequences from the Influenza Research Database (www.fludb.org). We curated these sequences to identify those that comprised full-length HA sequences, aligned them to the truncated head domain from PDB ID 4hkx and truncated the sequences to the same start and end residue. We also removed redundant sequences to create a unique set of seuqences. We clustered these sequences at 95% to reduce the panel to a more tractable size using the CD-HIT software with the following command:

```
cd-hit -i all_H1_unique_head_unique_clean.fasta -o
95_pct_cluster -c 0.95
```

To illustrate the homology modeling protocol we will use the strain H1 A_Uruguay_23_2009 as an example. We used the 13 H1 antigens from the original panel as template sequences – these can be found in H1_templates.fasta. We used the following python script to generate files for RosettaCM in an automated fashion.

```
python generate_files.py A_Uruguay_23_2009
```

This file will generate Grishin alignments for the top five templates by sequence identity, plus a file called template_identity.txt identifying these templates. You must thread your target sequence over the template structure for each of the top five templates. Use the following command to do so:

```
/path/to/Rosetta/main/source/bin/partial_thread.default.linuxgcc
release -in:file:fasta A_Uruguay_23_2009.fasta -
in:file:alignment A_Uruguay_23_2009_3m6s.grishin -
in:file:template_pdb 3m6s.pdb

mv 3m6s.pdb.pdb A_Uruguay_23_2009_on_3m6s.pdb
```

Repeat this command for all five templates. Next we recommend using the Robetta server to generate fragments for use in RosettaCM modeling. After downloading 3mer and 9mer fragment files you are ready to run RosettaCM. Run the following command to generate your models. They will be saved to a silent file format to save space – you can easily extract the top scoring models after homology modeling.

```
/path/to/Rosetta/main/source/bin/rosetta_scripts.default.linuxgc
crelease @rosetta_cm.options -scorefile A_Uruguay_23_2009.fasc -
out:file:silent A_Uruguay_23_2009.silent
```