3) Model rebuilding guided by experimental density data

In this scenario, we introduce a tool, RosettaCM, for building missing portions of a model guided by density data. While primarily geared towards comparative modeling, it may also be useful for building portions of a protein that are disordered when crystallized or difficult regions in hand-built models. In this scenario, we introduce the basic rebuilding protocol, then show how the tool may also be used to:

- Combine pieces from multiple template models guided by density
- Rebuild with user-defined restraints
- Iteratively rebuild models in difficult cases difficult cases

As a running example, we use the 20S proteasome (Xueming Li *et al.*, *Nature Methods*, 2013), where only a subset of particles were used, resulting in a 4.1Å reconstruction. We are building models starting from a homologous structure (pdb id: 1iru) as the starting model (25%/32% sequence identity to chains A/B).

Example 3A: Preparing templates for use in RosettaCM

In many cases, much of the setup work is handled by a script, *setup_RosettaCM.py* in RosettaTools (a separate repository available from rosettacommons.org). This script takes an input alignment in a variety of formats, and prepares the inputs automatically. It is executed by running the command:

```
setup_RosettaCM.py \
--fasta t20s.fasta \
--alignment tmpl.fasta \
--alignment_format fasta \
--templates tmpl.pdb \
--rosetta_bin ~/Rosetta/main/source/bin \
--verbose
```

Inputs include the full-length fasta, an alignment file – in either fasta, ClustalW, or HHSearch format – and the corresponding template PDB files. This script will prepare all the necessary inputs in order to run RosettaCM.

Alternately, the setup may be performed manually. In this case, since we are using some nonstandard features (symmetry and density), and we have two chains in the asymmetric unit we will do this; alternately, the inputs from the previous step may be used as a starting point and subsequently modified. In this case, we first convert our alignment to Rosetta format (*scenario2_model_rebuilding/20S_1iru.ali*):

```
## 1XXX_ 1iruAH_thread
# hhsearch
scores_from_program: 0 1.00
0 TVFSPDGRLFQVEYAREAVKK-GSTALGMKFANGVLLISDKKVRSRLIEQNSIEKIQLIDDYVAAVTSGLVADAR...
0 TIFSPEGRLYQVEYAFKAINQGGLTSVAVRGKDCAVIVTQKKVPDKLLDSSTVTHLFKITENIGCVMTGMTADSR...
--
```

In this format, the first line is '##' followed by a code for the target and one for the template. The second line identifies the source of the alignment; the third just keep as it is. The fourth line is the target sequence and the fifth is the template; the number is an 'offset', identifying where the sequence starts. However, the number doesn't use the PDB resid but just counts residues *starting at 0*. The sixth line is '--'. Multiple alignments may be concatenated in a single file, with the template code identifying the template corresponding to each alignment.

RosettaCM takes as inputs *partially threaded* models, that is models where aligned positions have their

residue identities remapped, and unaligned residues are not present. To generate these models from an alignment file and template, we can run the Rosetta command (*3_model_rebuilding/A_partialthread.sh*):

```
$ROSETTA3/source/bin/ partial_thread.macosclangrelease \
  -database $ROSETTA3/database/ \
  -in::file::fasta t20s.fasta \
  -in::file::alignment 20S_1iru.ali \
  -in::file::template_pdb 1iruAH_aln.pdb
```

This will output a partially threaded model in *1iruA_thread.pdb* that is correctly numbered for input into RosettaCM.

Next, we need to set up symmetric modeling with RosettaCM. As in Scenario 1, we use the *make_symmdef_file.pl* script in order to generate a symmetry definition file for use in Rosetta. A straightforward way to do so is to use Chimera to dock the necessary chains into density. We need a single "primary chain" and a couple of an adjacent chain in each point group; since the proteasome features D7 symmetry, that means we need an adjacent chain in the 7-fold complex, as well as a chain in the opposite ring. An example has been created in *scenario2_model_rebuilding/setup_symm.pdb* where three copies of the threaded model have been docked into density with Chimera. To generate our D7 symmetry file from this input, we then simply have to run the command (*3_model_rebuilding/B_make_symmdef.sh)*:

```
~/rosetta_source/src/apps/public/symmetry/make_symmdef_file.pl \
  -m NCS -a A -i B C \
  -p setup_symm.pdb –r 1000 > D7.symm
```

Since we have already created the input templates using the *partial_thread* application, we can ignore the *setup_symm_INPUT.pdb* file and use the output of partial thread as the input. However, we still need to align all the threaded models to this input structure. This can either be done with the program TMalign (external to Rosetta) or by using Chimera to dock the individual threaded models into density. In this case, where we have just one template, it has already been aligned to the template in *scenario2_model_rebuilding/tmpl_thread_aln.pdb.*

As in Scenario 1, we need to make a small edit to the symmetry definition file for density refinement. Change the following lines:

```
set_dof JUMP0_0_to_com x(35.3434689631743)
set_dof JUMP0_0_to_subunit angle_x angle_y angle_z
set_dof JUMP0_0 x(39.2905097135684) angle_x
```

To (*3_model_rebuilding/D7_edit.symm*):

```
set_dof JUMP0_0_to_com x y z
set_dof JUMP0_0_to_subunit angle_x angle_y angle_z
```

Note: The 20S proteasome case we are using contains two chains in the asymmetric unit. To specify this as inputs to RosettaCM, we need to list the fasta file, **separating the chains by the slash character '/'**. This is really only necessary in the fasta provided as input to RosettaCM (next step) however, there is no harm is doing this in every step.

Example 3B: Running RosettaCM using a single template model as input.

Like the methods introduced in Scenario 1, RosettaCM is controlled through an XML script using RosettaScripts. The XML is as follows (*3_model_rebuilding/C_rosettaCM_singletarget.xml)*:

```
<ROSETTASCRIPTS>
    <TASKOPERATIONS>
    </TASKOPERATIONS>
    <SCOREFXNS>
        <ScoreFunction name="stage1" weights="score3" symmetric="1">
            <Reweight scoretype="atom_pair_constraint" weight="0.1"/>
            <Reweight scoretype="elec_dens_fast" weight="10"/>
        </ScoreFunction>
        <ScoreFunction name="stage2" weights="score4_smooth_cart" symmetric="1">
            <Reweight scoretype="atom_pair_constraint" weight="0.1"/>
            <Reweight scoretype="elec_dens_fast" weight="10"/>
        </ScoreFunction>
        <ScoreFunction name="fullatom" weights="beta_cart" symmetric="1">
            <Reweight scoretype="atom_pair_constraint" weight="0.1"/>
            <Reweight scoretype="elec_dens_fast" weight="35"/>
                <Set scale_sc_dens_byres="R:0.76,K:0.76,E:0.76,D:0.76,M:0.76,
                    C:0.81,Q:0.81,H:0.81,N:0.81,T:0.81,S:0.81,Y:0.88,W:0.88,
                    A:0.88,F:0.88,P:0.88,I:0.88,L:0.88,V:0.88"/>
        </ScoreFunction>
    </SCOREFXNS>
    <FILTERS>
    </FILTERS>
    <MOVERS>
        <Hybridize name="hybridize" stage1_scorefxn="stage1" stage2_scorefxn="stage2"
                   fa_scorefxn="fullatom"  batch="1" stage1_increase_cycles="1.0"
                   stage2_increase_cycles="1.0" linmin_only="0" realign_domains="0">
            <Template pdb="1iruA_thread.pdb" weight="1.0"
                      cst_file="AUTO" symmdef="D7_edit.symm"/>
        </Hybridize>
    </MOVERS>
    <PROTOCOLS>
        <Add mover="hybridize"/>
    </PROTOCOLS>
</ROSETTASCRIPTS>
```

The main work is done through a single mover, *Hybridize* which handles all stages of model-building. Input structures are specified via *Template* lines (in this case there is only one). For each template line, we specify the pdb input, as well as a couple of other parameters: a *weight* (the relative frequency we sample each template with); a *constraint file* (setting this to "auto" sets up automatic constraints to the template, while setting this to "none" turns off all constraints, user-defined constraints are described later); and an (optional) *symmetry definition* file.

Note: Be sure that your templates are aligned to the density!

Given this XML, RosettaCM is then run with the following command line (*3_model_rebuilding/ B1_rosettaCM_singletarget.sh)*:

```
$ROSETTA3/source/bin/rosetta_scripts.macosclangrelease \
 -database $ROSETTA3/database/ \
 -in:file:fasta t20s.fasta \
 -parser:protocol C_rosettaCM_singletarget.xml \
 -nstruct 50 \
 -relax:jump_move true \
 -relax:dualspace \
 -out::suffix _singletgt \
 -edensity::mapfile t20S_41A_half1.mrc \
 -edensity::mapreso 5.0 \
 -edensity::cryoem_scatterers \
 -beta \
 -default_max_cycles 200
```

The input command is similar to those seen before, but with a few key differences. First, the input to Rosetta

is specified with -in:file:fasta rather than -in:file:s. Also note that the input argument –nstruct 50 is given, telling Rosetta to generate 50 models for each process. Generally, *hundreds to thousands of models* are necessary to sufficiently sample conformational space; more and longer regions to rebuild require more models.

Running without symmetry

Running without symmetry requires only two small changes to the XML file:

- Remove the tag: symmdef="D7_edit.symm"
- Remove the three tags symmetric=1

Job distribution

As with section 2, a combination of -out::suffix and GNU parallel is useful for distributing jobs. For example, one may replace the –out::suffix line above with –out::suffix _$1, then launch jobs with:

```
parallel –j16 ./ B1_rosettaCM_singletarget.sh {} ::: {1..16}
```

The total number of structures generated is the number of structures specified with –nstruct times the number of jobs launched (in this instance, 50 structure times 16 jobs = 800 structures). Depending on the runtime per structure (variable depending on structure size) and the number of CPUs available, both of these numbers may be adjusted.

Finally, GNU parallel allows launching of jobs remotely if SSH keys have been set up for passwordless login. To run:

```
parallel –S 16/node1,16/node2,16/node3,16/node4 --workdir . ./ B1_rosettaCM_singletarget.sh {}
::: {1..48}
```

This will launch instead 48 jobs spread across four machines. See the GNU parallel documentation for more information.r

---

Example 3C: Running RosettaCM using multiple template models as input.

One of the strengths of RosettaCM is its ability to make use of multiple template structures, and to recombine portions of these models during conformational sampling. This is particularly useful when multiple homologous structures are available, some with closer sequence identity, and some with more complete coverage. The protocol allows the combination of features of both models.

To make use of this feature, we simply add additional template lines in the input XML. In this case, we add the template 1ryp (*scenario3_model_rebuilding/ C1_rosettaCM_multitarget.xml)*:

```
...
      <Hybridize name="hybridize" stage1_scorefxn="stage1" stage2_scorefxn="stage2"
                 fa_scorefxn="fullatom"  batch="1" stage1_increase_cycles="1.0"
                 stage2_increase_cycles="1.0" linmin_only="0" realign_domains="0">
         <Template pdb="1iruA_thread.pdb" weight="1.0"
                cst_file="auto" symmdef="D7_edit.symm"/>
         <Template pdb="1rypA_thread.pdb" weight="1.0"
                cst_file="auto" symmdef="D7_edit.symm"/>
      </Hybridize>
...
```

The rest is handled automatically by the protocol. However, there are a few caveats when using multiple

input structures:

- With density, we need to ensure that all input models are aligned to the density. This can be done using either **TMalign** or **Chimera's alignment tools**.
- In each trajectory, a starting model is chosen at random; the constraints and symmetry from this selected model are chosen at the start of each run. **If we wish to use a portion of a model, but do not want to use its symmetry or constraints, we can assign it a weight of 0**: backbone conformations from this model will be used in conformational sampling, but the symmetry and constraints will never be used.
- Similarly, gaps in the selected starting model are rebuilt before recombination occurs. **If one of the templates has poor coverage, but provides valuable structural features, it should be used, but with weight 0.**

Example 3D: Running RosettaCM with user specified constraints.

Another strength of RosettaCM is the ability to make use of additional experimental information that provides restraints over conformational space. While previously, we have used *cst_file=auto* to automatically generate constraints from template structures, if experiments provide distance constraints (or some other positional restraint, we may make use of them in model rebuilding as well.

The Rosetta documentation provides a good overview of the types of constraints that may be used, with a number of different constraint types and functional forms possible. For this demo, we will assume we have knowledge on the distance between residues 107 and 143 that we want to use during rebuilding.

This information can be encoded in a constraint file (*scenario3_model_rebuilding/D1_constraints.cst)*:

```
AtomPair CA 107 CA 143 HARMONIC 5.0 1.0
```

Note: The numbering of residues is based upon the order in the input fasta file (and does not reset between chains!).

We then replace the *cst_file=auto* lines in the XML with our own constraint file (*scenario3_model_rebuilding/D1_constraints.xml)*:

```
...
   <Template pdb="tmpl_thread_aln.pdb" weight="1.0"
         cst_file="D1_constraints.cst" symmdef="D7_edit.symm"/>
...
```

We can then rebuild and refine as before.

Example 3E: Model selection and running RosettaCM iteratively

With possibly hundreds of generated models, there are a few strategies to identify the best-sampled models. Generally, models should be filtered on two different criteria – the *total score* and the *density score* – in some way. We often select the best 10-20% of models based on total score, and the sort these models by density score, but visual inspection of the best by both criteria can often be beneficial in difficult cases.

Finally, one strategy for solving difficult structures is to apply RosettaCM iteratively. Using the above criteria, we can select the best 5-10 models from the first round of refinement, and feed them as inputs into the next round. Models can be selected by energy by looking at the score column in the output .sc files.

This is very briefly illustrated in the following XML (*scenario3_model_rebuilding/E1_rosettaCM_iter.xml)*:

```
 ...
      <Hybridize name="hybridize" stage1_scorefxn="stage1" stage2_scorefxn="stage2"
            fa_scorefxn="fullatom" batch="1">
         <Template pdb="expected_outputs/S_multitgt_0001_A.pdb" weight="1.0"
               cst_file="NONE" symmdef="D7_edit.symm"/>
         <Template pdb="expected_outputs/S_multitgt_0002_A.pdb" weight="1.0"
               cst_file="NONE" symmdef="D7_edit.symm"/>
         <Template pdb="expected_outputs/S_multitgt_0003_A.pdb" weight="1.0"
               cst_file="NONE" symmdef="D7_edit.symm"/>
      </Hybridize>
...
```

There is also some manipulation of input models that can prove beneficial. If one wants to force rebuilding some segment of backbone, they can simply delete those residues in all input models. Similarly if one wants to force some region to adopt a conformation taking in one input model, they can delete all other conformations from all models.

Example 3F: Running with Ligands and/or Nucleic Acids

RosettaCM can be run with ligands as well as nucleic acids. While nucleic acids are read by Rosetta natively, ligands require an additional input to Rosetta, a *params* file.

For ligands, a mol2 file of the ligand which contains hydrogen atoms is required. The program OpenBabel (http://openbabel.org/wiki/Main_Page) is capable of both converting from PDB to mol2 as well as adding hydrogens to a molecule given a PDB file of the ligand.

Given a mol2 file, *molfile_to_params.py*, included in Rosetta, at $ROSETTA3/source/scripts/python/public/ will generate a params file. To use:

```
python $ROSETTA3/source/scripts/python/apps/public/molfile_to_params.py \
      --keep-names --clobber -centroid XXX.mol2 -p XXX -n XXX
```

Replace "XXX" with the ligands three letter code. This script will create an XXX.params file as well as several other files. This file can be passed into Rosetta using the flag:

```
-extra_res_fa XXX.fa.params
-extra_res_cen XXX.cen.params
```

If there is more than one unique ligand, run the script on each unique ligand, and pass a list of params files using each of the flags.

When modelling with ligands or nucleic acids in RosettaCM, two additional things are needed:
1. The ligand or nucleic acids must be added to the END of each template file with a weight > 0
2. An additional flag needs to get added to Hybridize:

```
<Hybridize name="hybridize" stage1_scorefxn="stage1" stage2_scorefxn="stage2"
            fa_scorefxn="fullatom" batch="1" add_hetatm="1">
```