2) Model refinement via iterative local rebuilding

In this section, we introduce our **cryoEM refinement protocol**, which uses an iterative local rebuilding procedure to escape local minima during refinement. The section is divided into two parts, in the first, we introduce the method for non-symmetric systems; in the second, we describe how to use this method for symmetric systems.

As a running example, we refine models of the transmembrane region of the TRPV1 ion channel, using a 3.4 Å cryoEM single particle reconstruction (M. Liao, E. Cao, D. Julius, Y. Cheng, *Nature*, 2013), and the deposited model (id: 3j5p) as a starting model. We will first refine this asymmetrically, and then introduce symmetric refinement.

Example 2A: Asymmetric refinement into cryoEM density

A summary of the XML used for refinement (*2_cryoem_refinement/A_asymm_refine.xml*) is shown below. Following, a brief description of the movers and options available is provided.

```
<ROSETTASCRIPTS>
   <SCOREFXNS>
      <ScoreFunction name="cen" weights="score4_smooth_cart">
         <Reweight scoretype="elec_dens_fast" weight="20"/>
      </ScoreFunction>
      <ScoreFunction name="dens_soft" weights="beta_soft">
         <Reweight scoretype="cart_bonded" weight="0.5"/>
         <Reweight scoretype="pro_close" weight="0.0"/>
         <Reweight scoretype="elec_dens_fast" weight="%%denswt%%"/>
      </ScoreFunction>
      <ScoreFunction name="dens" weights="beta_cart">
         <Reweight scoretype="elec_dens_fast" weight="%%denswt%%"/>
         <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>
   <MOVERS>
      <SetupForDensityScoring name="setupdens"/>
      <LoadDensityMap name="loaddens" mapfile="%%map%%"/>

      <SwitchResidueTypeSetMover name="tocen" set="centroid"/>

      <MinMover name="cenmin" scorefxn="cen" type="lbfgs_armijo_nonmonotone"
         max_iter="200" tolerance="0.00001" bb="1" chi="1" jump="ALL"/>

      <CartesianSampler name="cen5_50" automode_scorecut="-0.5" scorefxn="cen"
         mcscorefxn="cen" fascorefxn="dens_soft" strategy="auto" fragbias="density"
         rms="%%rms%%" ncycles="200" fullatom="0" bbmove="1" nminsteps="25" temp="4"
         fraglens="7" nfrags="25"/>
      <CartesianSampler name="cen5_60" automode_scorecut="-0.3" scorefxn="cen"
         mcscorefxn="cen" fascorefxn="dens_soft" strategy="auto" fragbias="density"
         rms="%%rms%%" ncycles="200" fullatom="0" bbmove="1" nminsteps="25" temp="4"
         fraglens="7" nfrags="25"/>
      <CartesianSampler name="cen5_70" automode_scorecut="-0.1" scorefxn="cen"
         mcscorefxn="cen" fascorefxn="dens_soft" strategy="auto" fragbias="density"
         rms="%%rms%%" ncycles="200" fullatom="0" bbmove="1" nminsteps="25" temp="4"
         fraglens="7" nfrags="25"/>
      <CartesianSampler name="cen5_80" automode_scorecut="0.0" scorefxn="cen"
         mcscorefxn="cen" fascorefxn="dens_soft" strategy="auto" fragbias="density"
         rms="%%rms%%" ncycles="200" fullatom="0" bbmove="1" nminsteps="25" temp="4"
         fraglens="7" nfrags="25"/>

      <ReportFSC name="report" testmap="%%testmap%%" res_low="10.0" res_high="%%reso%%"/>
```

```
      <BfactorFitting name="fit_bs" max_iter="50" wt_adp="0.0005" init="1" exact="1"/>

      <FastRelax name="relaxcart" scorefxn="dens" repeats="1" cartesian="1"/>
   </MOVERS>


   <PROTOCOLS>
      <Add mover="setupdens"/>
      <Add mover="loaddens"/>
      <Add mover="tocen"/>
      <Add mover="cenmin"/>
      <Add mover="relaxcart"/>
      <Add mover="cen5_50"/>
      <Add mover="relaxcart"/>
      <Add mover="cen5_60"/>
      <Add mover="relaxcart"/>
      <Add mover="cen5_70"/>
      <Add mover="relaxcart"/>
      <Add mover="cen5_80"/>
      <Add mover="relaxcart"/>
      <Add mover="relaxcart"/>
      <Add mover="report"/>
   </PROTOCOLS>
   <OUTPUT scorefxn="dens"/>

</ROSETTASCRIPTS>
```

The protocol is a bit involved, but is described in the following. The first thing to note is the use of macros like "%%denswt%%". These are command arguments that may be specified from the command line through the flag –parser::script_vars denswt=25.0. The protocol above uses these macros in place of parameters that users would most like to change; other parameters should be left as is except for advanced users.

The main addition is the *CartesianSampler* mover. This mover identifies backbone segments it believes are incorrect, given local strain and local density agreement. A Z score is computed compared to other refined near-atomic-resolution structures, and anything with a Z score worse than -0.5, -0.3, -0.1, or 0.0 is selected for rebuilding (this cutoff value is controlled through the *automode_scorecut* flag). The rebuilding loop then uses predicted local backbones to replace these segments iteratively. The number of rebuilding cycles can be controlled with the *ncycles* flag. The protocol as given above, runs through four stages, gradually increasing this value as refinement proceeds.

Another option is passed to the density scoring via the <*Set scale_sc_dens_byres=.../*> tag. In the refinement protocol, this sets a per-amino-acid sidechain reweighing. A value of 1 means the sidechain density weight is equal to the backbone. The weights shown in this example were determined by fitting these parameters into refined structures into several 3-5Å cryoEM density maps; the end result is a slight downweighing of sidechain density, particularly for charged sidechains. This should not be changed except by advanced users.

The *MinMover* first minimizes the structure using a low-resolution energy function (*cen*). We have found this step is most useful for improving protein backbone geometry, particularly with hand-traced models. This low-resolution scorefunction uses the *centroid* representation, which is enabled by the *SwitchResidueTypeSet* mover.

The *FitBFactors* mover fits real-space atomic B factors to maximize model-map correlation. A constraint enforcing nearby atoms to take the same B factors is also employed, and the weight on this term is controlled with the *wt_adp* term (0.0005 is generally well-behaved). Finally, *init=1* means to do a quick scan of overall B factors before beginning refinement; if there is more than one call to this mover in a single trajectory, then only the first needs to have *init=1*. *Exact=1* should always be used.

Finally, the *ReportFSC* mover assesses model agreement to the map used for fitting as well as an independent map using the integrated FSC over high-resolution shells. We have found integrating from 10Å to the resolution of the data is best for model discrimination.

Finally, this command is executed using the following (*scenario2_cryoem_refinement/A_asymm_refine.sh)*:

```
$ROSETTA3/source/bin/rosetta_scripts.macosclangrelease \
  -database $ROSETTA3/database/ \
  -in::file::s 3j5p_transmem_A.pdb \
  -parser::protocol A_asymm_refine.xml \
  -parser::script_vars denswt=35 rms=1.5 reso=3.4 map=half1_34A.mrc testmap=half2_34A.mrc \
  -ignore_unrecognized_res \
  -edensity::mapreso 3.4 \
  -default_max_cycles 200 \
  -edensity::cryoem_scatterers \
  -beta \
  -out::suffix _asymm \
  -crystal_refine
```

In bold are the parameters that should be changed in adapting this run for other systems. The first is the input structure, which should be specified with the argument –in::file::s. The second are the parameters to be passed through to the script. Three of these describe the input maps:

map=half1_34A.mrc – the map to refine against
testmap=half2_34A.mrc – an independent map for validation
(if not using split maps, just provide the same map as the previous argument)
reso=3.4 – the resolution of the data
(note that this needs to be provided twice in the command line, once for scoring and once for reporting)

The other two are parameters to the algorithm:

denswt=25 – the weight on the experimental density data
rms=1.5 – the amount of deviation to allow in fragment insertion moves
(larger values will lead to more model deviation)

The density weight of 25 works reasonably well as a starting point, but one might want to explore several different values using an independent reconstruction. Manual inspection of output models for molprobity score, free FSC, and (free FSC – work FSC) should provide clues as to which weight works best.

Job distribution

It is generally useful to sample ~100 models from each starting point. For this purpose, it may be useful to run multiple jobs in parallel. To prevent output structures from clobbering one another, the flag –out::suffix may be useful, where each separate job is given a different suffix.

For example, on a 16-core machine, we may specify -out::suffix _$1, then (using GNU parallel) run the following:

```
parallel -j16 ./A_asymm_refine.sh {} ::: {1..16}
```

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.

Analyzing results and model selection

While this is an active topic of research, generally – once a density weight has been chosen – to select the best models from among the full set, we want to select models optimizing both model geometry and free FSC values. Model geometry may be evaluated using either: a) Molprobity, or b) Rosetta energies after subtracting density energies. The latter may be done by inspecting the score*.sc files produced as output.

Using the above script, the free FSC value may be determined from the output PDB files. The header contains a line like:

```
REMARK   1 FSC[mask=4.45657](10:3) = 0.590966 / 0.591017
```

The two numbers at the end of the line indicate the "work" and "free" integrated FSC values.

Generally, we select the best 20% of models by geometry, and selecting the best overall by free FSC. The top 5 models should be inspected for model convergence as well as visually inspected for density map agreement.

Example 2B: Symmetric refinement into cryoEM density

As this is a symmetric system, to correctly evaluate the energetics of the system, we need to model with symmetry-related copies present. This may be done through a two-step process: first, we run the *make_symmdef_file.pl* script to prepare a description of the symmetry of the system in a Rosetta-readable format. Next, we enable symmetric scoring and optimization within the XML file.

The information that Rosetta needs to know about a symmetric system is encoded in the *symmetry definition file*. It tells Rosetta: (a) how to score a structure symmetrically from only asymmetric unit interactions, and (b) how the rigid-body degrees of freedom are allowed to move to maintain the symmetry of the system.

To aid in creating a symmetry definition file from a symmetric (or near-symmetric) PDB, an application, *make_symmdef_file.pl*, has been included in src/apps/public/symmetry. To generate the symmetry definition file for TRPV1, we run the command in *2_cryoem_refinement/B1_make_symmdef.sh*.

```
$ROSETTA3/source/src/apps/public/symmetry/make_symmdef_file.pl \
   -m NCS -a A -i B \
   -p 3j5p_transmem.pdb -r 1000 > TRPV1.symm
```

This script needs a few pieces of information: with –m, the type of symmetry to generate (here NCS), with –a, the primary chain (here A), and with –i, an adjacent chain in each symmetry group, separated by spaces (here just B). For C*n* symmetries, only one adjacent chain is given; for D*n*, two are given. Finally, with –r, we give the contact distance between a neighbor chain and the primary chain necessary to include that subunit explicitly (here, 1000, to ensure every symmetrically related copy is included). If the input system is asymmetric, the script will make a symmetrical version of it (sometimes significantly perturbing it in the process). There are a lot of other options, including forcing symmetrical order and helical and higher-order symmetries, see the documentation!

In addition to the definition file written to STDOUT, the script will also write a file **3j5p_transmem_symm.pdb**, containing the symmetrized version of the input file, and a file **3j5p_transmem_INPUT.pdb**, that contains only the mainchain, to be used as input (in addition to the

symmetry definition file).

The symmetry definition file looks something like this:

```
symmetry_name TRPV1__4
E = 4*VRT0_base + 4*(VRT0_base:VRT3_base) + 2*(VRT0_base:VRT2_base) anchor_residue CoM
...
set_dof JUMP0_to_com x(11.7023996817515)
set_dof JUMP0_to_subunit angle_x angle_y angle_z
...
```

The omitted sections describe a system of virtual residues that maintain the symmetry of the system, and they generally should remain unedited.

The two *set_dof* lines should be edited. There are two possibilities when refining symmetric structures into density:

A) we **don't want to refine the rigid body orientation of the entire system**;
we know the symmetry axes and we don't want them to move

B) we **do want to refine the orientation of the entire system**, including symmetry axes

Generally, in cryoEM, where the maps are symmetrically averaged, and the symmetry is known, we want to use strategy A. However, in some cases (for example, if our starting model was not perfectly symmetric) we want to use strategy B. In both cases, a minor edit to the set_dof lines in the symmdef file is necessary.

For strategy **A**, because we are using density, we need to change the first set_dof line to the following:

```
set_dof JUMP0_to_com x y z
```

For strategy **B**, we leave the two lines unchanged and instead add a third line:

```
set_dof JUMP0 x y z angle_x angle_y angle_z
```

For D*n* symmetries, the changes are similar, except in **A** the jump name is JUMP0_0_to_com. The rest of this section uses strategy **A**; the edited symmetry definition file is in *scenario1_cryoem_refinement/C4_edit.symm*

Once a symmetry definition file has been generated, then refining structures in Rosetta symmetrically is straightforward. The changes to the previous XML file are indicated below (see *2_cryoem_refinement/B2_symm_refine.xml*):

```
...
<ScoreFunction name="cen" weights="score4_smooth_cart" symmetric="1">
<ScoreFunction name="dens_soft" weights="eta_soft" symmetric="1">
<ScoreFunction name="dens" weights="talaris2013_cart" symmetric="1">
...
<SetupForSymmetry name="setupsymm" definition="%%symmdef%%"/>
...
<SymMinMover name="cenmin" scorefxn="cen" type="lbfgs_armijo_nonmonotone"
         max_iter="200" tolerance="0.00001" bb="1" chi="1" jump="ALL"/>
```

In all three declared scorefunctions, symmetric=1 must be given. Additionally, the *SetupForDensityScoring* mover must be replaced with the *SetupForSymmetry* mover. Finally, the *MinMover* must be replaced with its symmetric counterpart, *SymMinMover*.

The command for running this script is largely the same as before, with a few additions:

```
$ROSETTA3/source/bin/rosetta_scripts.macosclangrelease \
  -database $ROSETTA3/database/ \
  -in::file::s 3j5p_transmem_A.pdb \
  -parser::protocol A_asymm_refine.xml \
  -parser::script_vars \
    denswt=25 rms=1.5 reso=3.4 map=half1_34A.mrc \
    testmap=half2_34A.mrc symmdef=TRPV1_edit.symm \
  -ignore_unrecognized_res \
  -score_symm_complex false \
  -edensity::mapreso 3.4 \
  -default_max_cycles 200 \
  -edensity::cryoem_scatterers \
  -beta \
  -out::suffix _symm \
  -crystal_refine
```

The command **symmdef=TRPV1_edit.symm** passes the symmetry definition file to Rosetta. The flag -score_symm_complex false depends on whether strategy (a) or (b) was employed above. If (a), then *false* should be used; if (b), then *true* should be used.

Note: The input PDB (-in::file::s) is of the monomer (that is, the asymmetric unit).