Workshop #8: Loop Modeling

Loop modeling is an important step in building homology models, designing enzymes, or docking with flexible loops.

Suggested Readings

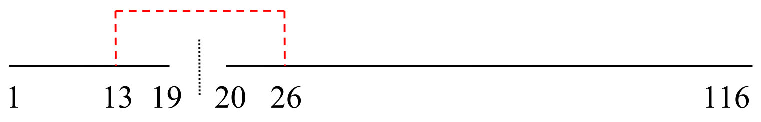
1. A. A. Canutescu & R. L. Dunbrack, "Cyclic coordinate descent: A robotics algorithm for protein loop closure," *Protein Sci.* **12**, 963-972 (2003).
2. C. Wang, P. Bradley & D. Baker, "Protein-protein docking with backbone flexibility," *J. Mol. Biol.* **373**, 503-519 (2007).
3. D. J. Mandell, E. A. Coutsias & T. Kortemme, "Sub-angstrom accuracy in protein loop reconstruction by robotics-inspired conformational sampling," *Nature Meth.* **6**, 551-552 (2009).

Fold Tree

Because we typically want to isolate the conformational changes to the loop region, we need a framework to hold the rest of the protein together. This is accomplished — as was the case with protein-protein docking — with a *fold tree*, which is a graph that dictates the propagation of conformational changes throughout the `Pose`.

For the following exercises, you can use the sample files found in `PyRosetta4/tests/data`. There you will find `test_in.pdb` and a 3mer fragment file, `test_in.frag3`.

Load the `test_in.pdb` structure (116 residues). We want to operate on the first loop, residues 15–24. For the fold tree, we place the jump anchors two residues outside of the loop range, *i.e.*, residues 13–26. In loop modeling, the jump points are set at *i*-2 and *j*+2, where *i* and *j* are the beginning and end residues of the loop, respectively. The relevant fold tree looks like this:



That is, we want a cut between residues 19 and 20, to allow motions in the loop that do not propagate through the rest of the protein. To tie the pieces together, we use a jump between residues 13 and 26. These residues will stay connected to each other.

To make such a tree in PyRosetta, first we create a `FoldTree` object:

```
ft = FoldTree()
```

Then we add the "edges" and the jump. Both edges and jumps are entered using the `FoldTree`'s `add_edge(start, end, code)` command, with peptide edges coded with a "-1" and jumps enumerated with the positive integers. (The first jump is coded "1", the second "2", *etc.*) The first edge is from residues 1 to 13:

```
ft.add_edge(1, 13, -1)
```

Then the second is from 13 to 19. An edge must always start from a residue that has already been defined in another edge, thus we use 13 here and not 14. (The one exception is the first edge, which starts from the graph's "root").

```
ft.add_edge(13, 19, -1)
```

Next, the jump, which is specified with the integer code 1, tells Rosetta that this is a rigid-body connection, not a peptide edge:

```
ft.add_edge(13, 26, 1)
```

Finally, we add the last two edges, both starting from 26, which is the residue that has been previously defined in the tree:

```
ft.add_edge(26, 20, -1)
ft.add_edge(26, 116, -1)
```

Print the fold tree and check that this tree is valid:

```
print ft
ft.check_fold_tree()
```

The latter command will return `False` if there are any invalid connections, disconnected parts, or undefined residues.

Finally, we attach this fold tree to the pose, overwriting its default fold tree:

```
pose.fold_tree(ft)
```

1. Test out your fold tree. Do `pose.set_phi(res, 180)` for `res` values of 10, 13, 16, 23, 26 and 29. View each in PyMOL with the PyMOL mover or viewer. (It may be helpful to color the structure by its foldtree using the `PyMOLMover.send_foldtree()` method.) What do you observe in these structures?

2. Sketch a fold tree that you could use for modeling a loop from residues 78–83. Remember that a loop from residues *i* to *j* uses a fold tree with a jump from residues *i*-2 to *j*+2.

3. What edges would you use to generate the above fold tree?

4. Visualize the foldtree that you just sketched above in PyMOL using `PyMOLMover.view_foldtree_diagram(pose, ft)`. Does your sketch match what is drawn on the screen? _____

To save some time and help avoid mistakes, there are a couple functions that will assist in the creation of fold trees:

5. Try each of the following and print the fold tree. What does each of the following do?

```
ft.clear()
ft.simple_tree(116)
ft.new_jump(76, 85, 80)
```

6. Use these `FoldTree` methods to check your answer to question 3.

7. Use the above commands to make a fold tree to model both loops (15–24 and 78–83) simultaneously.

Cyclic Coordination Descent (CCD) Loop Closure

Canutescu & Dunbrack's CCD routine is implemented as a `Mover`. It first requires that the loop is defined using the `Loop` class. You will also need to create a `MoveMap` with the loop residues marked as flexible (both backbone and side-chain torsion angles). Like many specialty objects in Rosetta, the CCD mover is located in its own namespace and is not loaded in by default when the rest of Rosetta is imported into Python. You can either import the module or refer to its namespace when calling it. Here we show the module import.

```
from rosetta.protocols.loops.loop_closure.ccd import *

loop1 = Loop(15, 24, 19)
ccd = CCDLoopClosureMover(loop1, movemap)
```

Before running the CCD algorithm, it is important to convert the residues around the cut site of the loop to "cut-point variants":

```
add_single_cutpoint_variant(pose, loop1)
```

8. Open the loop using `set_phi`, and run the CCD Mover. Does it close the loop? _____ Is the bond across the cut point protein-like? _____

Note also that if you have a loop defined in a Loop object, you can set your fold tree with the command:

```
set_single_loop_fold_tree(pose, loop)
```

Multiple Loops

Multiple loops can be stored in a `Loops` object. We can create a `loop2` object for the 78–83 loop and then create a `loops` object:

```
loops = Loops()
loops.add_loop(loop1)
loops.add_loop(loop2)
```

To use CCD on all loops, we have to iterate over each one.

Loop Building

The `MoveMap` and the `FoldTree` work together. By using a `MoveMap`, you can ensure that a `Mover` will only operate inside the loop region.

9. At this point, you should be able to write your own loop protocol that will build the loop at low-resolution using fragments. Some tips:

    - Create a `MoveMap` that will allow motions only in the two loop regions defined in our `MoveMap` above.
    - Create a `ClassicFragmentMover` using your `MoveMap` and the 3-residue fragment file provided, `test_in.frag3`.
    - Use the `cen_std` score function, but add the `chainbreak` score component with a weight of 1.
    - Do 100 fragment insertions.
    - After each fragment insertion, close the loop with CCD, then use a `MonteCarlo` object to accept or reject the combination move.
    - Bonus: use `SequenceMover` and `TrialMover` to tighten up your code.
    - Further bonus: use the `JobDistributor` to allow your program to make multiple structures.

Loop RMSD is typically measured in a fixed reference frame of the whole protein, and it can be computed on Cα atoms or all backbone atoms. PyRosetta has a built-in function for calculating deviation of all the loops, and its output can be added as additional info in the `JobDistributor`:

```
lrms = loop_rmsd(pose, reference_pose, loops, True)
jd.additional_decoy_info = " LRMSD: " + str(lrms)
```

(The fourth argument in `loop_rmsd()` tells whether or not the RMSD is calculated for C_α atoms only.)

10. If you first perturb the loop residues by setting all the residues to extended conformations ($\varphi=\psi=180°$), can your code close the two loops and find reasonable conformations? _____ What is the loop RMS? _________

High-Resolution Loop Protocol

In high-resolution, loop optimization needs smaller perturbations such as that from small and shear moves. The classic Rosetta loop refinement protocol is available as a `Mover` and is located in the rosetta.protocols.loops.loop_mover.refine namespace:

```
from rosetta.protocols.loops.loop_mover.refine import *
loop_refine = LoopMover_Refine_CCD(loops)
```

The mover uses its own default, high-resolution score function, and it will generate its own `MoveMap` based on the definition of the loops.

11. Apply this mover to a few of your low-resolution loop models after switching them to full atom versions. How far does refinement move the loops? _______ Do the loops remain closed? _____

Kinematic Closure (KIC) Protocols

The kinematic loop closure methods of Mandell *et al.* are also available in several prepared movers and protocols.

A single closure of a loop using the direct mathematical solution using polynomial resultants can be performed by applying the `KinematicMover`. Rather than using a `Loop` object, this mover is set up by specifying the three "pivot" residues. The kinematic mover determines new torsion angles for the pivot residues by solving the closure equations.

```
from rosetta.protocols.loops.loop_closure.kinematic_closure
                          import *
kic_mover = KinematicMover()
kic_mover.set_pivots(16, 20, 24)
kic_mover.apply(pose)
```

Like the `CcdLoopClosureMover`, the `KinematicMover` can be applied after various perturbations that alter or open the loop.

The full loop prediction protocol of Mandell *et al.* has been implemented as separate movers for the low- and high-resolution stages. The `LoopMover_Perturb_KIC` operates on a centroid representation pose, and it is designed to predict a loop *de novo.* The `LoopMover_Refine_KIC` operates in the full-atom representation, and it is designed to refine a loop making small perturbations from a starting conformation such as one output by the `LoopMover_Perturb_KIC` mover. Here is an example of the usage of both:

```
from rosetta.protocols.loops.loop_mover.perturb import *
from rosetta.protocols.loops.loop_mover.refine import *

sw = SwitchResidueTypeSetMover("centroid")
sw.apply(pose)
kic_perturb = LoopMover_Perturb_KIC(loops)
kic_perturb.apply(pose)

sw = SwitchResidueTypeSetMover("fa_standard")
sw.apply(pose)

kic_refine = LoopMover_Refine_KIC(loops)
kic_refine.apply(pose)
```

Because these are full prediction protocols, they will require some time to perform on your computer. Each will result in a single decoy structure; note that Mandell *et al*. generated 1000 decoy structures for blind predictions of 12-residue loops.

12. Time permitting, repeat exercises 8–10, replacing the CCD mover with the appropriate kinematic movers.

Simultaneous Loop Modeling and Docking

Antibodies have two chains, light (L) and heavy (H), which can move relative to each other. They also have a long, hypervariable H3 loop, typically residues 95–102. Antibodies are common protein drugs, and they are often created by exploiting the immune system of a mouse. There is a need for high-quality homology models of antibodies.

13. Sketch a fold tree that you could use to model an antibody with a flexible H3 loop and H and L chains that can move relative to each other.

14. Write a low-resolution protocol to alternate docking and loop modeling steps. Use your code to model cetuximab. Use the job distributor to track your decoys. What is the lowest RMSD you can create in 100 decoys? _______