

Workshop #8: Loop Modeling

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

Suggested Readings

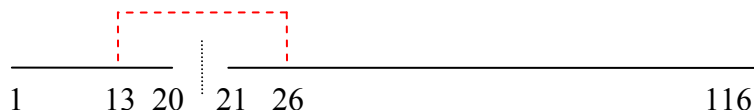
1. A. A. Cantescu & R. L. Dunbrack, “Cyclic coordinate descent: A robotics algorithm for protein loop closure,” *Protein Sci.* **12**, 963-972 (2003).
2. C. Wang, P. Bradley & D. Baker, “Protein-protein docking with backbone flexibility,” *J. Mol. Biol.* **373**, 503-519 (2007).

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 with a *fold tree*, which is a graph that dictates the propagation of conformational changes throughout the Pose.

For the following exercises, you can download the loop modeling package from the PyRosetta website (<http://www.pyrosetta.org/scripts.html>). In it 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. The relevant fold tree looks like this:



That is, we want a cut between residues 20 and 21, 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 residue 13 and 26. These residues will stay connected to each other.

To make such a tree in PyRosetta, first create a fold tree:

```
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 20. An edge must always start from a residue which 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(13,20,-1)
```

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

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

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

```
ft.add_edge(26,21,-1)
ft.add_edge(26,116,-1)
```

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

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

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

Attach this fold tree to the pose:

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

1. Test out your fold tree. Do `pose.set_phi(res)` for `res` values of 10, 13, 16, 23, 26 and 30. Output each pose and view in PyMol. 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?

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

4. 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)
```

5. Use these to check your answer to question 3.
6. Use the above commands to make a fold tree to model both loops (15-24 and 78-83) simultaneously.

Cyclic coordinate 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.

```
loop1 = Loop(15,24,20)
ccd = CcdLoopClosureMover(loop1, movemap)
```

7. 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. Create a loop2 object for the 78-83 loop and create a loops object.

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

To use CCD on all loops, you will 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.

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

- Create a MoveMap which will allow motions only in the two loop regions defined in our MoveMap.
- Create a ClassicFragmentMover using your MoveMap and the 3-residue fragment file provided, `test_in.frag3`.
- Use the centroid score function, but add the `chainbreak` score with 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.
- More 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 can be computed on C α 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)
```

8. If you first perturb the loop residues by setting all the residues to extended conformations ($\phi=\psi=180^\circ$), can your code close the two loops and find reasonable conformations? What is the loop rms? Submit your code.

High-resolution loop protocol

In high-resolution, loop optimization needs smaller perturbations such as that from small and shear moves. The classic Rosetta protocol is available as a Mover:

```
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.

9. Apply this mover to a few of your low-resolution loop models. How far does refinement move the loops? Do the loops remain closed?

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.

10. 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.

11. 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?