

Non-canonical amino acid (NCAA) parameterization using Rosetta

When designing peptides, one can find the vocabulary of the twenty canonical amino acids to be restrictive in designing tight, specific binders to the receptor. Therefore, a framework needs to exist for modeling amino acids which can have any sidechain chemistry, i.e., non-canonical amino acids (NCAAs). In order to include these NCAAs into Rosetta design simulations, you must first parameterize them, which involves listing atoms and bonds and their respective types, recording the initial geometry, assigning rotamers, etc. Thankfully, most of this process can be handled automatically by the `molfile_to_params_polymer.py` script, but rotamer assignment still remains a challenge for which Rosetta proposes multiple solutions. This tutorial will detail how to use each of these tools in combination with `molfile_to_params_polymer.py` to go from a molfile (.sdf) of a NCAA to a parameter file (.params) which is useable in Rosetta design simulations.

1. Using existing canonical parameters for non-canonicals

The simplest way of assigning rotamers to a NCAA is to use a set of rotamers which already exist and simply attach them to your NCAA. However, this requires that your NCAA be quite similar to an existing canonical AA. For this step, we will be parameterizing a 3,4,5-trifluorophenylalanine (abbreviated in this tutorial as TFF), which highly resembles a phenylalanine. First, make a directory for your params files and `cd` into that directory.

```
mkdir NCAA_params
cd NCAA_params
```

The molfile for TFF is located at `../input_files/TFF.sdf`. There are a few things to note about this input which are required for proper parameterization by Rosetta. First of all, you should notice that the backbone is in a dipeptide form where each end of the amino acid is extended to a methyl group representing the adjacent C-alpha atoms of neighboring amino acids. This is necessary for Rosetta to understand how to connect this AA to adjacent AA's. In the file itself, there is also a set of lines which inform Rosetta which atoms correspond to the various atoms of the backbone, which atoms connect to the upper and lower AA's in the sequence, and properties such as charge, aromaticity, and chirality. Because of these instructions, to parameterize this NCAA, all we have to run is:

```
python <RosettaDir>/main/source/scripts/python/public/molfile_to_params_polymer.py \
--clobber --polymer --no-pdb --name TFF --use-parent-rotamers PHE \
-i ../input_files/TFF.sdf
```

Note the usage of `--use-parent-rotamers` in this command, as this is what establishes which canonical AA rotamers you want to use. If you look at the generated parameter file with `cat TFF.params`, you should see a line which says `ROTAMER_AA PHE`, indicating phenylalanine's rotamers are being used for this AA.

2. Rigorous rotamer calculation using MakeRotLib

In 2012, Renfrew et. al. developed MakeRotLib, for generating NCAA rotamers through minimization of iterated initial conformational states using a hybrid Rosetta/CHARMM energy function. This protocol remains the most rigorous calculation of NCAA rotamers that exists in Rosetta, but due to its rigor, its runtime is not suitable for large libraries of NCAAs. In addition, the runtime scales exponentially with the number of chi angles and caps at 4 chis, so this protocol is also not suitable for highly flexible sidechains. Finally, MakeRotLib is not capable of handling anything other than monosubstituted alpha amino acids, so if your amino acid structure is exotic, it won't be able to be processed by MakeRotLib. To demonstrate the functionality of MakeRotLib, you will parameterize an amino acid with a methyl ether group as a sidechain (abbreviated as EAA in this tutorial). Similar to the previous case, `EAA.sdf` is in dipeptide form and has the necessary instructions for Rosetta to parameterize the molecule. Run `molfile_to_params_polymer.py` to get the parameter file:

```
python <RosettaDir>/main/source/scripts/python/public/molfile_to_params_polymer.py \
--clobber --polymer --no-pdb --name EAA -i ../input_files/EAA.sdf
```

However, since we excluded `--use_parent_rotamers`, this parameter file is not ready for use in Rosetta yet. In order to run MakeRotLib and generate the rotamers, you will need an options file, supplied at `../input_files/EAA_makerotlib_options.in`. This file specifies the angle ranges over which MakeRotLib should iterate, the number of chi angles in the sidechain, and initial guesses as to how many chi angle bins there are and where they lie. For this tutorial, the options file will consider all possible phi and psi angle values at increments of 10 degrees, every value of the single chi angle at 30 degree increments, and assume there are 3 chi rotamer bins each spaced 120 degrees apart (which is reasonable since the canonicals generally show this pattern as well). Now to run MakeRotLib:

```
<RosettaDir>/main/source/bin/MakeRotLib.default.linuxgccrelease \
  -extra_res_fa ./EAA.params -options_file ../input_files/EAA_makerotlib_options.in
```

This calculation should take a few minutes, after which you should have a ton of `EAA_*` files in your current directory. This directory contains logs from running MakeRotLib as well as a `.rotlib` file for each pair of phi/psi angle values. The final objective is to consolidate all of these files into a single `.rotlib`, and add a reference to this rotlib into the parameter file:

```
for i in `seq -170 10 180`; do
  for j in `seq -170 10 180`; do
    cat EAA_180_${i}_${j}_180.rotlib >> EAA.rotlib
  done
done
echo "NCAA_ROTLIB_PATH $PWD/EAA.rotlib" >> EAA.params
echo "NCAA_ROTLIB_NUM_ROTAMER_BINS 1 3" >> EAA.params
rm -rf EAA_*
```

Now that the file `EAA.params` has been assigned rotamers, it is now ready to use in Rosetta. Note that the `NCAA_ROTLIB_PATH` is hardcoded, so if you use the example `.params` file in `output_files`, you will need to change this path to match your environment.

3. Small molecule conformers as NCAA rotamer libraries (FakeRotLib)

While MakeRotLib is the most accurate method for rotamer construction in Rosetta, it does not apply in many contexts, as was previously discussed. To address some of these shortcomings, we consider the NCAA as a small molecule and define the rotamers of the NCAA as low energy conformers of the “small molecule”. The implementation of this idea is the `fake_rotlib.py` script, which uses RDKit to generate conformations of the NCAA, score the conformations using the UFF forcefield, and utilize the *N* lowest energy conformations in the parameter file as “PDB rotamers”. The distinction between this implementation of the rotamer library and the previous methods is that the previous methods define the *distribution* of rotamers and then score a given conformation according to its position in that distribution, whereas PDB rotamers store a set of acceptable conformations and randomly draws from these conformations when modeling the residue. Since PDB rotamers don’t have to fit into the distribution parameters accepted by Rosetta, pretty much any NCAA can be accommodated by PDB rotamers. On the other hand, PDB rotamers inherently discretize the conformational space, are not compatible with some movers, and generally require more compute time and memory in modeling. To allow both types of rotamer libraries to be built, `fake_rotlib.py` also has functionality to generate a rotamer distribution file (`.rotlib`) from the PDB rotamers (as long as the NCAA has four or less chi angles). In addition to rotamer modeling, `fake_rotlib.py` automates a few other steps of the process, including dipeptide capping, writing the params instructions, and running `molfile_to_params_polymer.py`.

As a demonstration of `fake_rotlib.py`, we parameterize another phenylalanine derivative (with an attached Bis(2-chloroethyl)amine group), abbreviated as MFF in this tutorial. Since this molecule has far more chi angles than any other we’ve parameterized before, we will be using PDB rotamers in lieu of a `.rotlib` file. `fake_rotlib.py` depends on RDKit to generate conformers, so we need a python with it installed. We also need to move the `fake_rotlib.py` script into the Rosetta source:

```
cp ../scripts/fake_rotlib.py <RosettaDir>/main/source/scripts/python/public/
```

With a python interpreter with RDKit active, simply run:

```
python <RosettaDir>/main/source/scripts/python/public/fake_rotlib.py \  
  --input '../input_files/MFF.sdf' --dip -n 100  
mv ../input_files/MFF* ./
```

Note that the `--dip` flag is used here because the input `MFF.sdf` is already in dipeptide form and has parameterization instructions pre-generated. If instructions need to be generated, run without this flag and ensure that the input is NOT in dipeptide form (either neutral or zwitterionic backbone is acceptable). This causes two important files to be generated: the `MFF.params` which possesses a reference to the PDB rotamers file `MFF_rotamer.pdb`. `MFF_rotamer.sdf` is also here, but this is an intermediate file used input to `molfile_to_params_polymer.py`. Regardless, as long as `MFF_rotamer.pdb` remains in the same directory as `MFF.params`, the file is ready to be used in Rosetta simulations.