Glycan Modeling

Bold text means that these files and/or this information is provided.

Italicized text means that this material will NOT be conducted during the workshop

`fixed width text means you should type the command into your terminal`

If you want to try making files that already exist (e.g., input files), write them to a different directory! (mkdir my_dir)

Authors

- Tutorial and Program Author:
  - Jared Adolf_Bryfogle (jadolfbr@gmail.com)
- Corresponding PI:
  - Bill Schief (schief@scripps.edu)

Citation

Residue centric modeling and design of saccharide and glycoconjugate structures Jason W. Labonte Jared Adolf-Bryfogle William R. Schief Jeffrey J. Gray *Journal of Computational Chemistry*, 11/30/2016 - https://doi.org/10.1002/jcc.24679

Automatically Fixing Errors in Glycoprotein Structures with Rosetta Brandon Frenz, Sebastian Rämisch, Andrew J. Borst, Alexandra C. Walls Jared Adolf-Bryfogle, William R. Schief, David Veesler, Frank DiMaio *Structure*, 1/2/2019

Overview

We will be using the RosettaCarbohydrate framework to build and model glycans. The GlycanTreeModeler, which is our main method for modeling glycans will be published this summer. We will be using some custom glycan options to load pdbs. First, one needs the `-include_sugars` option, which will tell Rosetta to load sugars and add the sugar_bb energy term to a default scorefunction. This scoreterm is like rama for the sugar dihedrals which connect each sugar residue.

```
-include_sugars
```

When loading structures from the PDB that include glycans, we use these options. This includes an option to write out the structures in pdb format instead of the Rosetta format (which is actually better). Again, this is included in the config/flags files you will be using.

```
-maintain_links
-auto_detect_glycan_connections
-alternate_3_letter_codes pdb_sugar
-write_glycan_pdb_codes
```

More information on working with glycans can be found at this page: Working With Glycans

Backbone Torsions, Residue Connections, and side-chains

A glycan tree is made up of many sugar residues. Each residue a ring. The 'backbone' of a glycan is the connection between one residue and another. The chemical makeup of each sugar residue in this 'linkage' effects the propensity/energy of each bacbone dihedral angle. In addition, sugars can be attached via different carbons of the parent glycan. In this way, the chemical makeup and the attachment position effects the dihedral propensities. Typically, there are two backbone dihedral angles, but this could be up to 4+ angles depending on the connection.

In IUPAC, the dihedrals of N are defined as the dihedrals between N and N-1 (IE - the parent linkage). The ASN (or other glycosylated protein residue's) dihedrals become part of the first glycan residue that is connected. For this first first glycan residue that is connected to an ASN, it has 4 torsions, while the ASN now has none!

If you are creating a movemap for dihedral residues, please use the MoveMapFactory as this has the IUPAC nomenclature of glycan residues built in in order to allow proper DOF sampling of the backbone residues, especially for branching glycan trees. In general, all of our samplers should use residue selectors and internally will use the MoveMapFactory to build movemaps internally.

A sugar's side-chains are the constitutents of the glycan ring, which are typically an OH group or an acetyl group. These are sample together at 60 degree angles by default during packing. A higher granularity of rotamers cannot currently be handled in Rosetta, but 60 degrees seems adequete for our purposes.

Within Rosetta, glycan connectivity information is stored in the GlycanTreeSet, which is continually updated to reflect any residue changes or additions to the pose. If you are using PyRosetta or C++, this info is always available through the function

```
pose.glycan_tree_set()
```

Chemical information of each glycan residue can be accessed through the CarbohydrateInfo object, which is stored in each ResidueType object:

```
pose.residue_type(i).carbohydrate_info()
```

Algorithm

The `GlycanTreeModeler` essentially builds glycans from the root (The first residue of the Tree) out to the trees in a way that simulates a tree growing. It uses a notation of a 'layer' where the layer is defined as the number of residues to the glycan root (with the glycan root being layer 0). Within modeling, all glycan residues other than the ones being optimized are 'virtualized'. In Rosetta, the term 'Virtual' means that these residues are present, but not scored. (It should be noted that it is now possible to turn any residues Virtual and back to Real using two movers in RosettaScripts: `ConvertVirtualToRealMover` and `ConvertRealToVirtualMover`.)

Within the modeling application, sampling of glycan DOFs is done through the `GlycanSampler`. The sampler attempts to sample the large amount of DOFs available to a glycan tree. The GlycanSampler is a `WeightedRandomSampler`, which is a container of highly specific sampling strategies, where each strategy is weighted by a particular probability. At each apply, the mover selects one of these samplers using the probability set to it. This is the same way the SnugDock algorithm for antibody modeling works.

Sampling is always scaled with the number of glycan residues that you are modeling, so run-time will increase proportionally as well. If you are modeling a huge viral particle with lots of glycans, one can use quench mode, which will optimize each glycan individually. Tpyically for these cases, multiple rounds of glycan modeling is desired.

GlycanSampler Major components

1. Glycan Conformers

   These conformers have been generated through an in-depth bioinformatic analysis of the PDB using adaptive kernal density estimates and are unique for each linkage type including glycan residues connected to ASN residues. A conformer is a specific conformation of all of the dihedrals of a particular glycan linkage. Essentialy glycan 'fragments' for a particular type of linkage.

2. SugarBB Sampling

   This sampling is done through turning the sugar_bb energy term into a set of probabilities using the -log(e) function. This allows us to sample on the QM derived torsonal potentials during modeling.

3. Random Sampling and Shear Moves

   We sample random torsions +/- 15 , +/- 45, +/- 90 degrees each at decreasing probabilities at a 4:2:1 ratio of sampling Small,Medium,Large. Shear sampling is done where torsions are set for two residues in order to reduce downsteam effects and allow 'flipping' of the glycan torsions. The version that you are using in this tutorial does not include shear sampling.

4. Minimization and Packing

   1. Packing
      Of the residues set to optimize, chooses a random residue and packs that residue and all residues out to the tree that are not virtualized. We pack the sugar residues and any neighboring protein sidechains. TaskOperations may be set to allow design of protein residues during this.
   2. Minimization
      Minimize Sugar residues by selecting a residue in what is set to model, and selecting all residues out to the tree that are not virtualized.

General Setup and Inputs

You will be using a few different inputs. We will be designing in glycosylation spots in order to block antibody binding at a highly curved epitope, and we will be loading a human structure from the PDB that has internal glycans.

1. Notes for Tutorial Shortening

   Typically, the value of `-glycan_sampler_rounds` is set to 25 (which typically is enough) and nstruct is about 5-10k per input structure. You may increase glycan_sampler_rounds to 100 and then decrease output to 1-2500 nstruct in order to have the same level of sampling, which will result in very good models as well. Since this is denovo modeling of glycans, more nstruct is almost always better. For some tutorials, we may decrease this value below our optimal value in order to shorten the length of the tutorial.

2. General Notes

   This tutorial assumes that you have Rosetta added to your PATH variable. If you do not already have this done, add the rosetta applications to your path. For the Meilerlab workshop (tcsh shell), do this:

   ```
   setenv PATH ${PATH}:${HOME}/rosetta_workshop/rosetta/main/source/bin
   setenv PATH ${PATH}:${HOME}/rosetta_workshop/rosetta/main/source/tools
   ```

   We will be using JSON output of the scorefile, as this is much easier to work with in python and pandas. We use the option `-scorefile_format json`

   All of our common options for the tutorial are in the common file that you will copy to your working directory. Rosetta will look for this file in your working directory or your home folder in the directory `$HOME/.rosetta/flags`. See this page for more info on using rosetta with custom config files: https://www.rosettacommons.org/docs/latest/rosetta_basics/running-rosetta-with-options#common-options-and-default-user-configuration

   All tutorials have generated output in output_files and their approximate time to finish on a single (core i7) processor.

Tutorial

GlycanModeling is done through the RosettaScripts interface. Each tutorial has you copying a base XML and adding/modifying specific components to achieve a goal.

Tutorial A: Epitope Blocking, De-novo Glycan Modeling

Here, we will start with the antigen known as Bee Hyaluronidase, from PDB ID 2J88. The PDB file has an antibody bound to it as a HIGHLY immunogenic site. We would like to block this in order to use begin to use this enzyme for therapy as Hyaluronidase can be effective in breaking down sugars in the extracellular matrix, allowing certain larger drugs to get to regions of interest. The antibody is renumbered into the AHo numbering scheme that we use in the RAbD tutorial, and it has been relaxed with constraints into the Rosetta energy function.

We will be designing in at least one optimal glycan at the most immunogenic site. Note that a prototocol called SugarCoat is in development that will scan regions of interest for potential ideal glycosylation, however, one can certainly do this manually as we do below.

1. Designing in a Glycosylation Site:

   `CreateGlycanSequonMover` and `CreateSequenceMotifMover`

   A sugar glycosylation site is known as a `Sequon`. The glycan sequon is made up of three protein residues which are recognized by the GlycosylTransferase Enzyme during translation in the ER. This enzyme adds the root of nascent glycan onto a protein. In this case, we use the sequon for ASN glycosylation. The sequon is as follows: `N[^P][S/T]`. The `[^P]` notation means that any residue other than P can be there. The `[S/T]` notation means that either S or T is recognized. This notation can be used to directly create Motifs in proteins using the `CreateSequenceMotifMover` and associated `SequenceMotifTaskOperation`. Documentation for these is available here:

   - https://www.rosettacommons.org/docs/wiki/scripting_documentation/RosettaScripts/xsd/mover_CreateSequenceMotifMover_type
   - https://www.rosettacommons.org/docs/wiki/scripting_documentation/RosettaScripts/xsd/to_SequenceMotifTaskOperation_type

   The create GlycanSequonMover can also be used for glycosylation of different AA than ASN.

   1. Design using a typical sequon

      ```
      mkdir work_dir
      cp ../input_files/common .
      cp ../input_files/tutA11.xml .
      cp ../input_files/2j88_complex.pdb .
      cp ../input_files/2j88_antigen.pdb .

      <CreateGlycanSequeonMover name="motif_creator" residue_selector="select"/>
      ```

      Before we begin, take a look at the complex. Where can we introduce a glycan to block binding? Where do you think the optimal glycan position would be for this particular antibody? Take a look at the xml. Is this the position we are targeting? Typically, we may want to allow some backbone movement in our sequon. The full glycan scanning protocol can be found in an input file, simple_glycan_scanner_manual.xml, where we relax the motif residues with constraints, add the sequon, and then relax again, comparing the energy between them to get the full energetic contributions of the sequon on the structure. In order to reduce the run time in these tutorials, we will be removing this going forward.
      Go ahead and run the xml (about 15 seconds)

      ```
      rosetta_scripts.linuxgccrelease -s 2j88_antigen.pdb -native 2j88_antigen.pdb \
          -parser:protocol tutA11.xml -parser:script_vars start=143A end=145A \
          -out:prefix tutA11_
      ```

      Take a look at the scorefile. Why do we have all these extra values here? These are the SimpleMetrics, and they have replaced filters for calculating useful values in Rosetta. In the xml, we define a few SimpleMetrics. We run a set before we actually create the sequon and then a set of metrics afterwards! In the XML, you see we use a prefix in the `RunSimpleMetrics` mover to denote any metrics run after the seqeon creation. Take a look at the protocol section and then at the RunSimpleMetrics movers we have defined. What is the prefix that is used post-sequon creation? Ok, now go back to the score file - what values have we output? Did we successfully design in our motif?

2. Design using the `N[^P][T]` motif

   This motif has been shown to have higher occupancy of the glycosation site with glycans in the resulting protein. Glycosylation is not 100% in some cases at some positions for (currently) unknown reasons, but this paper [] is a bioinformatic analysis that concludes that this motif has a higher occupancy. If we were creating a drug, we can use chromatography during protein isolation to choose peaks which include our glycan. Here, we are using the [-] notation as to not actually design the second position. We will use what is in the native protein here.

   ```
   cp ../input_files/tutA12.xml .

   <CreateSequenceMotifMover name="create_sequon" residue_selector="p1" motif="N[-]T"/>

   rosetta_scripts.linuxgccrelease -s 2j88_antigen.pdb -native 2j88_antigen.pdb \
       -parser:protocol tutA12.xml -parser:script_vars start=143A end=145A \
       -out:prefix tutA12_
   ```

   Was the sequon successfully designed? Take a look at the scorefile. Is the sequence that was designed different than the previous tutorial? (compare `sequence` to 'post-sequon_sequence). How is the energy difference from the native protein? Use the SimpleMetric output - look for the output that has native_delta in the name. Did we change the SASA?

2. Adding a man5 glycan:

   `SimpleGlycosylateMover`

   Now, we will expand on our first tutorial by glycosylating afterward. We will use the common name for a man5 sugar, which is a high-mannose branching glycan of 7 sugar residues (and 5 mannoses). You can use a few common names to make glycosylation easier, or an IUPAC string, or a file that has the IUPAC string in the first name of the file. Common names include man5,man7,man9 and a few others. You can find these in

   ```
   Rosetta/main/database/chemical/carbohydrates/common_glycans
   ```

   The IUPAC nomenclature of the man5 is as follows:

   ```
   a-D-Manp-(1->3)-[a-D-Manp-(1->3)-[a-D-Manp-(1->6)]-a-D-Manp-(1->6)]
                                             -b-D-Manp-(1->4)-b-D-GlcpNAc-(1->4)-b-D-GlcpNAc-
   ```

   More information on IUPAC nomenclature of sugar trees is here: http://www.chem.qmul.ac.uk/iupac/2carb. There is also a very detailed README in the common glycan directory for your reference.

   Note that within the `SimpleGlycosylateMover` you may also give multiple glycans using the `glycans` option, which will randomly choose a glycan tree to use for glycosylation from the list given. Glycosylation is not deterministic in that you always get a man5 at a particular position and is influenced by a great deal of structural biology that is not yet fully determined. For now, since we are aiming to create a drug and purifying our result, using a man5 is sufficient. This takes about 15 seconds.

   ```
   cp ../input_files/tutA2.xml .

   <SimpleGlycosylateMover name="glycosylate" residue_selector="select" glycan="man5" />

   rosetta_scripts.linuxgccrelease -s 2j88_antigen.pdb -native 2j88_antigen.pdb \
       -parser:protocol tutA2.xml -parser:script_vars start=143A end=145A \
       -out:prefix tutA2_
   ```

   We built the glycan and have not done any modeling, so lets model some glycans!

3. Modeling glycans

1. `GlycanResidueSelector` and the `GlycanTreeModeler`

   We will run the previous tutorials in a single rosetta script where we end with modeling the glycan residues. We use a very short run time and nstruct, so results will not be as clean as they would otherwise, but this should give you an idea of how all this works. Typically, we would model different positions of potential glycosylations, but here to save time, we will simply continue to build and model the glycan position we started with. Output files have been provided for you if you wish to use these. We will not be giving the mover a residue selector as it uses all glycans by default, but you can use the GlycanResidueSelector to choose specific trees or even glycan residues within those trees to model. This takes about 380 seconds to run.

   ```
   cp ../input_files/tutA3.xml .

   <GlycanTreeModeler name="model" layer_size="2" window_size="1" rounds="1" refine="false" />

   rosetta_scripts.linuxgccrelease -s 2j88_antigen.pdb -native 2j88_antigen.pdb \
       -parser:protocol tutA3.xml -parser:script_vars start=143A end=145A \
       -out:prefix tutA3_ -nstruct 10
   ```

   Use the scorefile.py script to get the lowest energy model.

   ```
   scorefile.py --scores total_score --output tab tutA3_score.sc | sort  -k2 -k1
   ```

   How does it look? Load the native into pymol as well. Would this glycan block this particular antibody? Where else could we place a glycan?

Tutorial B: Using Glycan Density

In this tutorial we will load a pdb directly into Rosetta with sugars already present. The config for this has been provided for you.

```
cp ../input_files/pdb_flags .
cp ../input_files/4do4_refined.pdb.gz .
cp ../input_files/4do4_crys.symm .
cp ../input_files/4do4_symm.pdb .
```

The glycan tree that we will be working with is 5 residues long. I use coot to look at density maps. Density maps were generated by downloading the cif file from the PDB and using PHENIX maps and default `maps.params`. This command was used to generate them:

```
phenix.maps 4do4.pdb 4do4-sf.cif
```

The density map generated is too large to be distributed with the rest of the tuorial, so I have uploaded it to Google Drive for you to download. https://drive.google.com/open?id=1h569jpwLxyHu7iHLG8eu2Q9_B-Q9e_C9 Please download and place it in your working directory, if it's not already there.

1. Calculating Density Fit

   Although a structure may be solved with high resolution, not all solved residues may fit the density well. A structure from the PDB is still a model afterall, informed through experimentation. This is especially true of glycan residues, which are fairly mobile. Crystal contacts of neighboring proteins help to reduce the movemment of glycans and may help to induce a state that can be solved more easily given high-resolution density. In this tutorial, we will be using Rosetta to determine how well a residue fits into the given density. There are methods to do this in the coot program, but we want to be able to do this for any structure in a streamlined way - especially if we need to calculate RMSDs on only well-fitting glycan residues. The methods we will be employing in Rosetta are based on Frank Dimaio's work with Rosetta density.

To do this, we will once again be employing the SimpleMetric system. In this case, we use the `PerResidueDensityFitMetric`, which is a PerResidueRealMetric. This type of SimpleMetric calculates a particular value for each residue given a residue selector. Very useful here. We will also be employing the DensityFitResidueSelector, which uses the metric. Since this is a fairly slow metric, we will use in-built functionality for using our calculated values from the metric, which are stored in the pose. We will then use the SelectedResidueCountMetric to determine how many residues have great fit. In later tutorials, we will be using the RMSDMetric with this selector in order to calculate RMSD on well-fitting glycan residues.

Residues higher than .8 are great fit to density. Residues between .6 - .8 are good fit to density Residues below .4 fit to density are BAD fits

```
cp ../input_files/tutB1.xml .
```

```
<PerResidueDensityFitMetric name="fit_native" residue_selector="tree" output_as_pdb_nums="1"
                                                   sliding_window_size="1" match_res="1"/>
<DensityFitResidueSelector name="fits8" den_fit_metric="fit_native" cutoff=".8" use_cache="1"
                                                                fail_on_missing_cache="1"/>
<SelectedResidueCountMetric name="n_fits8" custom_type="fit8" residue_selector="fits8"/>
```

```
rosetta_scripts.linuxgccrelease -fconfig common pdb_flags \
    -s 4do4_refined.pdb.gz -native 4do4_refined.pdb.gz -parser:protocol tutB1.xml \
    -parser:script_vars branch=177A map=4do4_2mFo-DFc_map.ccp4 symmdef=4do4_crys.symm \
    -out:prefix tutB1_
```

Run the xml and while it is running, take a look at the XML (runtime is about 80 seconds). It is fairly complicated and we will be building on it during the rest of these tutorials. Note that we first define the density metric, and then we use it within the selector. At the bottom, we add these to our set of native_metrics. What other metrics are we using?

Ok, take a look at the scorefile. You can use the scorefile.py script to output as tabs if you would like. How many residues have great fit to density (hint, look for fit6_selection_count and fit8_selection_count data terms)? Are there any residues that fit poorly into the density?

2. Refinement into density

   Here, we will be doing a short refinement protocol into the density, with its crystal symmetry. This is a short protocol, but will work for our purposes. For a much longer (albeit very similar) refinement protocol of the glycan and whole protein, see Frenz et al (referenced at the top of the page). The full protocol used in this paper is included in the input files as cryoem_glycan_refinement.xml. Take a look and see how it compares to what we are doing here. As usual, output files are available. Runtime for all 10 structures is about 2 hours.

```
cp ../input_files/tutB2.xml .

rosetta_scripts.linuxgccrelease -fconfig common pdb_flags map_flags \
    -s 4do4_refined.pdb.gz -native 4do4_refined.pdb.gz -parser:protocol tutB2.xml \
    -parser:script_vars branch=177A map=4do4_2mFo-DFc_map.ccp4 symmdef=4do4_crys.symm \
    -out:prefix tutB2_ -nstruct 10
```

   Are the density fit scores higher? How different are the RMSDs of the glycan residues? Take a look at the structure of the lowest energy - how different does it look? Are any new contacts created? Were we able to improve the density fit for some of those residues?

3. Denovo building into Density

   In this tutorial, we will be once again loading our crystal structure with density and symmetry. However, we will be randomizing the bb torsions and building the glycan out from scratch. In reality, we would have some idea of what glycan we are building and we would glycosylate the protein with the chemical motif we have figured out from means such as mass spec. We would then model the glycan to solve the crystal structure. With the new PackerPalette machinery in Rosetta and the ability to design glycans, we could actually build a protocol to sample chemical motifs of the glycans we are building out into the density, however, since this a very very large combinatorial problem, we should have some idea of what exists in the structure.

We will first rebuild the glycan tree using the density as a guide, and then refine it further using what we learned in the previous tutorial. Note that like tutorial B2, this one takes a good long while (4 hours)

```
cp ../input_files/tutB3.xml.

rosetta_scripts.linuxgccrelease -fconfig common pdb_flags map_flags \
    -s 4do4_refined.pdb.gz -native 4do4_refined.pdb.gz -parser:protocol tutB3.xml \
    -parser:script_vars branch=177A map=4do4_2mFo-DFc_map.ccp4 symmdef=4do4_crys.symm \
    -out:prefix tutB3_ -nstruct 10
```

How are our RMSDs? Were we able to do enough sampling to get close to the native structure? Are the energies acceptable? Are there parts of the glycan that are closer to native than others? Why might this be? Is an nstruct of 10 enough??

Thank you for doing this tutorial! I hope you learned a lot and are ready to work with these crazy carbohydrates! Cheers!