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 Author:
  - Cristina Elisa Martina (cristinaelisamartina@gmail.com)
- Program Author:
  - Jared Adolf_Bryfogle (jadolfbr@gmail.com)

Overview

In this tutorial we will be using the RosettaCarbohydrate framework to build and model glycans. The GlycanTreeModelerMover, which is our main method for modeling glycans, will be published soon.

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 epitope in order to use the 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.

In this tutorial we will:

1. Download, trim, clean and score the complex in pdb 2J88
2. Relax and score again the complex
3. Insert the sequon to add the glycosylation site (in two ways)
4. Add the glycan to the structure and model its conformation

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

Tutorial: Epitope Blocking, De-novo Glycan Modeling

As for the other tutorials, create your own working directory:

```
mkdir work_dir
cd work_dir
```

1. Download, trim, clean and score the complex in pdb 2J88

In this first step, we will download the pdb from RCSB-PDB (note that you can also copy it from the input_file folder). We will then inspect it with pymol: after coloring by chain, you will see three chains: chain A in green is the Bee Hyaluronidase; chain H in cyan is the heavy chain of the antibody and chain L in magenta is the light chain of the antibody. Take note of how the constant part of the antibody (the one not in contact with the Hyaluronidase) is missing some residues. We will remove this portion of the antibody to allow faster calculation and we will save the trimmed pdb as 2j88_trim.pdb (note that you can find this pdb in the input_file folder).

1. Go to the RCSB-PDB website (https://www.rcsb.org/) and search for 2J88.

2. Download the file in pdb format and save it in the work_dir folder. Note that you can also copy the file from the input_file folder with:

```
cp ../input_file/2j88.pdb .
```

3. Open the pdb with pymol from the terminal:

```
pymol 2j88.pdb
```

   If you can not open Pymol from the terminal, just open it as you usually do, and drag the 2j88.pdb file in the black windows of pymol. You will need to pay attention once we save the file in step 7.

4. With Pymol console, remove the solvent (paste the command in the terminal box of Pymol itself).

```
remove solvent
```

   Note that you can also use the GUI version instead of the command line: click on the "A" button (Action) on the right box, and select "remove water".

5. With Pymol, color the structure by chain, (again use the terminal from Pymol for this command).

```
util.cbc
```

   GUI version: click on the "C" button (Color), and select "by chain"

6. Select and remove the poorly solved portion of the antibody. We will remove from residue 111 to residue 180 for the heavy chain, and from residue 107 to residue 205 for the light chain:

```
select To_Remove, /2j88//H/111-180 + /2j88//L/107-205
remove (To_Remove)
```

   For the GUI version: click on the "S" button (Sequence) on the bottom-right corner to display the amino acid sequence. Manually select the residues that we want to discard. Click on the "A" button in the "(sele)" line and select "remove atoms".

7. Save the trimmed version of the pdb. If in Step 3 you were able to open pymol from the terminal, you can save directly the pdb without changing the path (but double check that the new file is created in the folder!)

```
save 2j88_trim.pdb
```

   If you had to drag the pdb in your pymol window instead, you will need to look for the correct path to the folder. From the pymol terminal write "save" and press the Tab-key to see in which folder you are. You will need to give the correct path for the tutorial folder and call the file "2j88_trim.pdb". It should be something like this:

```
save /YOUR/PATH/Workshop_2021/Glycan_Modeling/work_dir/2j88_trim.pdb
```

   If you are lost with the terminal, and you prefer the GUI version, click on "File" -> "Export Molecule" -> Selection "2j88" -> "Save". Remember to save it as "2j88_trim.pdb"

8. Close Pymol

Now, we clean and score the complex. After cleaning, you obtain 4 files: the 2j88_AHL.pdb and 3 fasta files containing the amino acid sequence for each chain in the complex. We will not use the fasta files in this tutorial. Notice that after cleaning, the complex is renumbered starting from 1 up to 536. You can open in pymol the structures 2j88_trim.pdb and 2j88_trim_AHL.pdb and compare the difference in the numbering. You will notice that the two structures are identical.

For the scoring, we will use the Interface Analyzer Mover, that will give us information about the total score of the complex and about the binding energy between the antigen and the antibody. You will need 3 input files for the scoring: 1- the cleaned pdb (2j88_trim_AHL.pdb), the Interface_Analyzer.xml file and the Interface_Analyzer.options file. If you open the .xml file you can see how we are defining the interface in 'interface="A_HL" '. On one side of the interface we have chain A (the Bee Hyaluronidase) and on the other side we have chains H and L (the antibody).

9. Clean the pdb and remove the fasta files (we do not need them):

```
/PATH/TO/ROSETTA/main/tools/protein_tools/scripts/clean_pdb.py \
    2j88_trim.pdb AHL

rm *.fasta
```

10. Check the pdb in pymol:

```
pymol *.pdb
set seq_view
```

11. Close Pymol

12. Copy the files for scoring from the input_file folder

```
cp ../input_file/Interface_Analyzer.* .
```

13. Run the analysis, it will take ~1 minute:

```
/PATH/TO/ROSETTA/main/source/bin/rosetta_scripts.linuxgccrelease \
    -s 2j88_trim_AHL.pdb -parser:protocol Interface_Analyzer.xml \
    @Interface_Analyzer.options -out:file:scorefile Clean_Score.sc
```

14. Extract from the score file the total energy from column 2, the binding energy (dG_separated) from column 6 and the shape complementarity value (sc_value) from column 36. Here the command with awk, but you can also do it manually with excel or other programs.

```
cat Clean_Score.sc | tail -n +2 | awk '{print $NF, $2,$6,$36}' \
    > 0_Clean_Scores.txt
```

You will notice that the score is not optimal: for structures composed by ~500 aa we should expect a total energy around -1500 REU while the score we obtain is around -60 REU. The same is valid for the binding energy: expected binding energies for antibody binding range from -30 REU to -70 REU and here we have a positive value. The reason for these "bad" values is that the crystal structures obtained from RCSB-PDB are usually not optimal for Rosetta. They can contain clashes or bond stretches that not only give a bad score, but also are very unlikely to happen in proteins in solution. To fix this, we will refine the crystal structure with FastRelax in the next step.

The shape complementarity indicates how good the shape of the antibody complements with the shape of the antigen (or viceversa). Values range from 0 (the shapes do not match at all) to 1 (the shapes match perfectly and complement each other). A value of 0.65 or higher indicates reasonable interfaces. In our example with the Bee Hyaluronidase, we have a shape complementarity of ~0.75 that indicates a good fit between antibody and antigen.

2. Relax and score again the complex

In this section we will relax the structure, score it again with Interface Analyzer and inspect the interface with pymol. To relax the crystal structure we will use the FastRelax Mover. This protocol will perform multiple rounds of side-chains packing and whole structure minimization in order to find a low-energy conformation. You can find more info on this mover here: FastRelax.

For the sake of time, we will generate only a single output, that will then be used for the next step. It is however recommended to generate multiple outputs and select the best one in terms of total energy.

1. Copy the FastRelax xml and options file in the work_dir:

```
cp ../input_file/FastRelax.* .
```

2. Run the relax, it will take ~1 hour:

```
/PATH/TO/ROSETTA/main/source/bin/rosetta_scripts.linuxgccrelease \
    -s 2j88_trim_AHL.pdb -parser:protocol FastRelax.xml \
    @FastRelax.options
```

3. Run interface analyzer again, we will use the same .xml and .options files we used before but we give a different input structure (the relaxed one). It will take ~1 minute:

```
/PATH/TO/ROSETTA/main/source/bin/rosetta_scripts.linuxgccrelease \
    -s Rlx_2j88_trim_AHL_0001.pdb -parser:protocol \
    Interface_Analyzer.xml @Interface_Analyzer.options \
    -out:file:scorefile Relax_Score.sc
```

4. Extract from the score file the total energy, the binding energy and the shape complementarity value as we did before:

```
cat Relax_Score.sc | tail -n +2 | awk '{print $NF,$2,$6,$36}' \
    > 0_Relax_Scores.txt
```

5. Compare the energies before and after relax.

6. Compare the structures before and after relax:

```
pymol 2j88_trim_AHL.pdb Rlx_2j88_trim_AHL_0001.pdb
```

7. Identify residues of the antigen that are in contact with the antibody. You can manually select the residues on the antigen that are in contact with the antibody, they should be residues 127 to 132. For the commandline version use:

```
sele Interface, resi 127-132
show sticks, Interface
color red, Interface
```

8. Close pymol

After relax, the energies drop down to more reasonable Rosetta values: around -1660 REU for the total energy and -45 REU for the binding energy. There are no significant changes in shape complementarity and this is expected since there are no major conformational changes in the structures and in the interface.

From Pymol, we can notice that there are three key residues at the tip of the antigen involved in the interaction with the antibody: F129, W130 and D131. Each one of these positions can be a good candidate for glycosylation and in a normal protocol they should all be tested. For this tutorial however we will consider only the first one: F129.

3. Insert the sequon to add the glycosylation site (in two ways)

In this section we will first remove the antibody chains from the structure, and then use two different methods to mutate the protein and insert a sugar glycosylation site, known as a 'sequon'. The N-glycosylation sequon is made up of three residues which are recognized by the GlycosylTransferase Enzyme during translation in the ER. The sequon is composed by the NxT/S triplet, in which the first residue has to be an Asparagine (this residue is the one that will be glycosylated); the second residue can be anything but Proline and the third residue has to be a Threonine or a Serine.

Before inserting the mutations for the sequon, we need to remove the antibody chains from the pdb. We will use DeleteChainMover (more info: DeleteChain). In the Delete_Chain.xml file you can see how chains H and L are specified in the "chain" parameter.

The first method we will use to insert the sequon is CreateGlycanSequonMover. This mover was designed to explicitely insert glycosylation sites in proteins. The default setting creates a sequon for N-linked glycans, but it can be used also for C-linked glycans (more info: CreateGlycanSequence).

We will then test the total energy and the Solvent-Accessible Surface Area (SASA) before changing the sequence (pre_metrics in the Create_Glycan_Sequon.xml), and we will calculate total energy, SASA, and Sequence after sequon insertion (post_metrics).

1. Copy the Delete_Chain.xml:

```
cp ../input_file/Delete_Chain.xml .
```

2. Remove chains H and L, it will take few seconds:

```
/PATH/TO/ROSETTA/main/source/bin/rosetta_scripts.linuxgccrelease \
    -s Rlx_2j88_trim_AHL_0001.pdb -parser:protocol Delete_Chain.xml \
    -out:prefix Del_
```

3. Check the new pdb file, then close pymol:

```
pymol Del_Rlx_2j88_trim_AHL_0001_0001.pdb
```

4. Let's rename the pdb. This step is not necessary for the protocol itself, but just to shorten the file name and have a better readability:

```
cp Del_Rlx_2j88_trim_AHL_0001_0001.pdb 2j88_antigen.pdb
```

5. Copy the Create_Glycan_Sequon.xml:

```
cp ../input_file/Create_Glycan_Sequon.xml .
```

6. Mutate the protein with the sequon motif, it will take few seconds:

```
/PATH/TO/ROSETTA/main/source/bin/rosetta_scripts.linuxgccrelease \
    -s 2j88_antigen.pdb -native 2j88_antigen.pdb -parser:protocol \
    Create_Glycan_Sequon.xml -parser:script_vars \
    start=129A end=131A -out:prefix Sqn1_
```

7. Compare the proteins with pymol:

```
pymol 2j88_antigen.pdb Sqn1_*.pdb
sele Interface, resi 129-131
show sticks, Interface
set seq_view
```

8. Close pymol

The second method we will use to insert the sequon is CreateSequenceMotifMover. This mover is much more general and it is used to insert any kind of motifs (more info: CreateSequenceMotif).

We will define the motif as 'N[-]T', where '[-]' indicates the wild type residue for that particular position. As done in the previous step, we will then test the total energy and the SASA before and after changing the sequence with the sequon motif.

9. Copy the Create_Sequence_Motif.xml file:

```
cp ../input_file/Create_Sequence_Motif.xml .
```

10. Mutate the protein with the sequon motif, it will take few seconds:

```
/PATH/TO/ROSETTA/main/source/bin/rosetta_scripts.linuxgccrelease \
    -s 2j88_antigen.pdb -native 2j88_antigen.pdb -parser:protocol \
    Create_Sequence_Motif.xml -parser:script_vars \
    start=129A end=131A -out:prefix Sqn2_
```

11. Compare the proteins with pymol:

```
pymol 2j88_antigen.pdb Sqn*.pdb
sele Interface, resi 129-131
show sticks, Interface
set seq_view
```

12. Close pymol

13. Let's compare the energies and the SASA; from both the score files we will extract the pdb name (last column), the motif sequence (column 26), the total energy post and pre-sequon (columns 19 and 21), the SASA post- and pre-sequon (columns 18 and 20). We will directly calculate the delta (model - native) with the awk command:

```
echo "Name Motif ETot_post ETot_pre ETot_delta SASA_post \
    SASA_pre SASA_delta" > 0_Sequon_scores.txt

cat Sqn1_score.sc | tail -n+3 | awk '{print $NF,$26,$19,$21, \
    $19-$21,$18,$20,$18-$20}' >> 0_Sequon_scores.txt

cat Sqn2_score.sc | tail -n+3 | awk '{print $NF,$26,$19,$21, \
    $19-$21,$18,$20,$18-$20}' >> 0_Sequon_scores.txt
```

Both methods mutated the protein to insert the sequon, NWS with the first one and NWT for the second one. Differences in energy, in structure and in SASA are minimal and the two models can be considered equally good according to Rosetta. However it is experimentally known that the sequon NxT is more efficient than NxS, so we will continue the tutorial with the NWT sequon (the second one).

4. Add the glycan to the structure and model its conformation

In this last step we are adding the glycan to the Asparagine residue that we just created and we will model its conformation. To add the glycan, we will use the SimpleGlycosylateMover. We will need to include the option "-include_sugars" to our Rosetta command from now on (see the options files). This option will tell Rosetta to load sugars and add the sugar_bb energy term to a default scorefunction, and will take into account the dihedral angles that connect each sugar residue, otherwise unknown for Rosetta. We will add 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

```
/PATH/TO/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: Nomenclature.

Note that within the `SimpleGlycosylateMover` you may also specify multiple glycan types (e.g. man5, man7 and man9) using the `glycans` option, which will randomly choose a glycan tree to use for glycosylation from the list given. Inside the cell, glycosylation is not deterministic and you will not always get a man5 at a particular position. Sometimes glycolylation does not happen at all and it is not clear why. However, for this tutorial, man5 is a good representative for glycosylation.

1. Copy Glycosylate xml and options files:

```
cp ../input_file/Glycosylate.* .
```

2. Run the glycosylation protocol, it will take ~1 minute:

```
/PATH/TO/ROSETTA/main/source/bin/rosetta_scripts.linuxgccrelease \
    -s Sqn2_*.pdb -native Sqn2_*.pdb -parser:protocol Glycosylate.xml \
    @Glycosylate.options -parser:script_vars start=129A -out:prefix Glyc_
```

3. Analyze the structure:

```
pymol Sqn2_*.pdb Glyc_*.pdb
sele N129, resi 129
show sticks, N129
```

4. Close pymol

5. Analyze the score and the SASA:

```
cat Glyc_score.sc | awk '{print $NF,$4,$3}' > 0_Glycosylation_scores.txt
```

As you can see the score is really bad because the glycan was added without any minimization. We will now model it to find more favorable conformations with GlycanTreeModelerMover.

6- Copy Model_Glycan xml and options files

```
cp ../input_file/Model_Glycan.* .
```

7- Model the glycan, this step will take ~2 hours:

```
/PATH/TO/ROSETTA/main/source/bin/rosetta_scripts.linuxgccrelease \
    -s Glyc_*.pdb -native Glyc_*.pdb -parser:protocol Model_Glycan.xml \
    @Model_Glycan.options -parser:script_vars start=129A -out:prefix Mdl_
```

8. Check the scores obtained from glycan modeling:

```
cat Mdl_score.sc | awk '{print $NF,$2}' > 0_Modeling_scores.txt
```

9. Check the structures:

```
pymol Mdl_*.pdb
sele N129, resi 129
show sticks, N129
```

10. Load the antigen-antibody complex:

```
load Rlx_2j88_trim_AHL_0001.pdb
```

11. Close pymol

After modeling, the total energy of the glycosylated antigen went back to normal (~ -850 REU). The 10 models show all different low energy conformations of the glycan chain, and mimic the glycan flexibility in solution. In this tutorial we generated only 10 models due to time limits, but in a normal protocol a minimum of 500 models should be generated.

In the pymol session you can see how the generated glycan chains occupy the same space previously occupied by the antibody. In an experimental setting such an ELISA, the antibody should be unable to bind the antigen due to the glycan shield.

I hope you enjoyed the tutorial and do not hesitate to contact me for questions!

Citations

Rosetta Carbohydrates:

1. J. W. Labonte, J. Adolf-Bryfogle, W. R. Schief, and J. J. Gray, **"Residue-Centric Modeling and Design of Saccharide and Glycoconjugate Structures"**, J Comput Chem, vol. 38, no. 5, pp. 276–287, Feb. 2017, doi: 10.1002/jcc.24679.

2. B. Frenz et al., **"Automatically Fixing Errors in Glycoprotein Structures with Rosetta"**, Structure, vol. 27, no. 1, pp. 134-139.e3, Jan. 2019, doi: 10.1016/j.str.2018.09.006.

3. M. L. Nance, J. W. Labonte, J. Adolf-Bryfogle, and J. J. Gray, **"Development and Evaluation of GlycanDock: A Protein-Glycoligand Docking Refinement Algorithm in Rosetta"** J Phys Chem B, Jun. 2021, doi: 10.1021/acs.jpcb.1c00910.

Rosetta Scripts:

4. S. J. Fleishman et al., **"RosettaScripts: A Scripting Language Interface to the Rosetta Macromolecular Modeling Suite"**, PLOS ONE, vol. 6, no. 6, p. e20161, Jun. 2011, doi: 10.1371/journal.pone.0020161.

Rosetta FastRelax:

5. F. Khatib et al., **"Algorithm discovery by protein folding game players"**, PNAS, vol. 108, no. 47, pp. 18949–18953, Nov. 2011, doi: 10.1073/pnas.1115898108.

6. J. B. Maguire et al., **"Perturbing the energy landscape for improved packing during computational protein design"**, Proteins: Structure, Function, and Bioinformatics, vol. 89, no. 4, pp. 436–449, 2021, doi: 10.1002/prot.26030.

Rosetta DeleteChain:

7. J. Adolf-Bryfogle et al., **"RosettaAntibodyDesign (RAbD): A general framework for computational antibody design"**, PLoS Comput. Biol., vol. 14, no. 4, p. e1006112, 2018, doi: 10.1371/journal.pcbi.1006112.

More details on this protocol:

Carbohydrate Backbone Torsions, Residue Connections, and Side-Chains

A glycan tree is made up of many sugar residues. Each residue is 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 Overview

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.

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 sugar 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.