Open in Colab

De Novo Protein Design with PyRosetta

Keywords:

Overview

Make sure you are in the directory with the `.pdb` files:

```
cd google_drive/My\ Drive/student-notebooks/
```

Warning: This notebook uses `pyrosetta.distributed.viewer` code, which runs in `jupyter notebook` and might not run if you're using `jupyterlab`.

Note: This Jupyter notebook requires the PyRosetta distributed layer which is obtained by building PyRosetta with the `--serialization` flag or installing PyRosetta from the RosettaCommons conda channel (for more information, visit: http://www.pyrosetta.org/dow).

In [1]:
```
import Bio.Data.IUPACData as IUPACData
import Bio.SeqUtils
import logging
logging.basicConfig(level=logging.DEBUG)
import os
import pyrosetta
import pyrosetta.distributed
import pyrosetta.distributed.viewer as viewer
import site
import sys

# Notebook setup
if 'google.colab' in sys.modules:
    !pip install pyrosettacolabsetup
    import pyrosettacolabsetup
    pyrosettacolabsetup.mount_pyrosetta_install()
    print ("Notebook is set for PyRosetta use in Colab.  Have fun!")
```

```
DEBUG:pyrosetta.distributed.utility.pickle:ImportError loading blosc, falling back to un
compressed pickle archives.
DEBUG:pyrosetta.distributed.viewer:IPython.core.display expanding Jupyter notebook cell
width.
```

Initialize PyRosetta:

In [2]:
```
flags = """
-linmem_ig 10
-ignore_unrecognized_res 1
```

```
"""
pyrosetta.distributed.init(flags)
```

```
DEBUG:pyrosetta.distributed:with_lock call: <function maybe_init at 0x2aabff60f5e0>
DEBUG:pyrosetta.distributed:maybe_init normalizing 'options': '-linmem_ig 10 -ignore_unr
ecognized_res 1'
INFO:pyrosetta.distributed:maybe_init performing pyrosetta initialization: {'options':
'-linmem_ig 10 -ignore_unrecognized_res 1', 'extra_options': '-out:levels all:warning',
'set_logging_handler': 'interactive', 'silent': True}
INFO:pyrosetta.rosetta:Found rosetta database at: /home/konrad/miniconda3/envs/rosetta20
22/lib/python3.9/site-packages/pyrosetta/database; using it....
INFO:pyrosetta.rosetta:PyRosetta-4 2022 [Rosetta PyRosetta4.conda.linux.cxx11thread.seri
alization.CentOS.python39.Release 2022.12+release.a4d79705213bc2acd6b51e370eddbb2738df68
66 2022-03-20T21:59:37] retrieved from: http://www.pyrosetta.org
(C) Copyright Rosetta Commons Member Institutions. Created in JHU by Sergey Lyskov and P
yRosetta Team.
DEBUG:pyrosetta.distributed:with_lock finished: <function maybe_init at 0x2aabff60f5e0>
```

Let's setup the pose of a *de novo* helical bundle from the PDB for downstream design: https://www.rcsb.org/structure/5J0J

In [3]:

```
start_pose = pyrosetta.io.pose_from_file("inputs/5J0J.clean.pdb")
pose = start_pose.clone()
```

```
INFO:rosetta:core.conformation.Conformation: [ WARNING ] missing heavyatom:  OXT on resi
due ALA:CtermProteinFull 70
INFO:rosetta:core.conformation.Conformation: [ WARNING ] missing heavyatom:  OXT on resi
due ALA:CtermProteinFull 139
INFO:rosetta:core.conformation.Conformation: [ WARNING ] missing heavyatom:  OXT on resi
due ALA:CtermProteinFull 210
```

Design Strategy

Minimize the crystal structure coordinates, then convert chain "A" to poly-alanine, and then perform one-sided protein-protein interface design designing chain A while only re-packing the homotrimer interface residues of chains B and C! Therefore, we are designing a homotrimer into a heterotrimer. Furthermore, prevent backbone torsions from minimizing and only minimize the side-chains of the homotrimer interface and all of chain A using the `FastDesign` mover! After design, repack and minimize all side-chains.

Prior to and after design, we want to relax the protein with a scorefunction that packs the rotamers and minimizes the side-chain degrees of freedom while optimizing for realistic energies. If you were to allow backbone minimization (which you may optionally choose to), we would want use a Cartesian scorefunction (in this case, `ref2015_cart.wts`) which automatically sets `cart_bonded` scoreterm to a weight of 1.0, which helps to close chain breaks in the backbone. Similarly, you might want to also turn on the `coordinate_constraint` scoreterm to penalize deviations of the backbone coordinates from their initial coordinates during minimization. In this tutorial, we demonstrate that concept but will prevent backbone torsions from being minimized (however, feel free to turn on backbone minimization!):

In [4]:

```
relax_scorefxn = pyrosetta.create_score_function("ref2015_cart.wts")
relax_scorefxn.set_weight(pyrosetta.rosetta.core.scoring.ScoreType.coordinate_constrain
print("The starting pose total_score is {}".format(relax_scorefxn(start_pose)))
```

```
The starting pose total_score is 2113.581015295595
```

For *design*, if we allowed backbone minimzation we again would turn on the `coordinate_constraint` scoreterm to penalize deviations of the backbone coordinates from their initial coordinates during minimization. Additionally, we will use "non-pairwise decomposable" scoreterms to guide the packer trajectories (i.e. fixed backbone sequence design trajectories) in favor of our hypothetical design requirements to solve our biological problem. In this tutorial, we will make use of the following additional scoreterms during design:

- `aa_composition` : This scoring term is intended for use during design, to penalize deviations from a desired residue type composition. Applies to any amino acid composition requirements specified by the user within a ResidueSelector. This scoreterm also applies to the `AddHelixSequenceConstraints` mover which sets up ideal sequence constraints for each helix in a pose or in a selection.
- `voids_penalty` : This scoring term is intended for use during design, to penalize buried voids or cavities and to guide the packer to design solutions in which all buried volume is filled with side-chains.
- `aa_repeat` : This wholebody scoring term is intended for use during design, to penalize long stretches in which the same residue type repeats over and over (e.g. poly-Q sequences).
- `buried_unsatisfied_penalty` : This scoring term is intended for use during design, to provide a penalty for buried hydrogen bond donors or acceptors that are unsatisfied.
- `netcharge` : This scoring term is intended for use during design, to penalize deviations from a desired net charge in a pose or in a selection.
- `hbnet` : This scoring term is intended for use during design, to provide a bonus for hydrogen bond network formation.

In [5]:

```
design_scorefxn = pyrosetta.create_score_function("ref2015_cart.wts")
design_scorefxn.set_weight(pyrosetta.rosetta.core.scoring.ScoreType.coordinate_constrai
design_scorefxn.set_weight(pyrosetta.rosetta.core.scoring.ScoreType.aa_composition, 1.0
design_scorefxn.set_weight(pyrosetta.rosetta.core.scoring.ScoreType.voids_penalty, 0.25
design_scorefxn.set_weight(pyrosetta.rosetta.core.scoring.ScoreType.aa_repeat, 1.0)
design_scorefxn.set_weight(pyrosetta.rosetta.core.scoring.ScoreType.buried_unsatisfied_
design_scorefxn.set_weight(pyrosetta.rosetta.core.scoring.ScoreType.hbnet, 1.0)
design_scorefxn.set_weight(pyrosetta.rosetta.core.scoring.ScoreType.netcharge, 1.0)
print("The starting pose total_score is {}".format(design_scorefxn(start_pose)))
```

```
The starting pose total_score is 23016.581015295596
```

By using the `relax_scorefxn` before and after the `design_scorefxn` , we ensure that these "non-pairwise decomposable" scoreterms are not forcing unrealistic rotamers that would otherwise not be held in place without these additional scoreterms.

Prior to any deviation from crystal structure coordinates, apply coordinate constraints to all of the backbone heavy atoms:

In [6]:

```
true_selector = pyrosetta.rosetta.core.select.residue_selector.TrueResidueSelector() # 

# Apply a virtual root onto the pose to prevent large lever-arm effects while minimizin
```

```
virtual_root = pyrosetta.rosetta.protocols.simple_moves.VirtualRootMover()
virtual_root.set_removable(True)
virtual_root.set_remove(False)
virtual_root.apply(pose)

# Construct the CoordinateConstraintGenerator
coord_constraint_gen = pyrosetta.rosetta.protocols.constraint_generator.CoordinateConst
coord_constraint_gen.set_id("contrain_all_backbone_atoms!")
coord_constraint_gen.set_ambiguous_hnq(False)
coord_constraint_gen.set_bounded(False)
coord_constraint_gen.set_sidechain(False)
coord_constraint_gen.set_sd(1.0) # Sets a standard deviation of contrained atoms to (an
coord_constraint_gen.set_ca_only(False)
coord_constraint_gen.set_residue_selector(true_selector)

# Apply the CoordinateConstraintGenerator using the AddConstraints mover
add_constraints = pyrosetta.rosetta.protocols.constraint_generator.AddConstraints()
add_constraints.add_generator(coord_constraint_gen)
add_constraints.apply(pose)
```

Prior to design, minimize with the `FastRelax` mover to optimize the pose within the `relax_scorefxn` scorefunction. Note: this takes ~1min 10s

In [7]:
```
tf = pyrosetta.rosetta.core.pack.task.TaskFactory()
tf.push_back(pyrosetta.rosetta.core.pack.task.operation.InitializeFromCommandline())
tf.push_back(pyrosetta.rosetta.core.pack.task.operation.IncludeCurrent())
tf.push_back(pyrosetta.rosetta.core.pack.task.operation.NoRepackDisulfides())
tf.push_back(pyrosetta.rosetta.core.pack.task.operation.OperateOnResidueSubset(
    pyrosetta.rosetta.core.pack.task.operation.PreventRepackingRLT(), true_selector)) #
mm = pyrosetta.rosetta.core.kinematics.MoveMap()
mm.set_bb(False) # Set to true if desired
mm.set_chi(True)
mm.set_jump(False)
fast_relax = pyrosetta.rosetta.protocols.relax.FastRelax(scorefxn_in=relax_scorefxn, st
fast_relax.cartesian(True)
fast_relax.set_task_factory(tf)
fast_relax.set_movemap(mm)
fast_relax.minimize_bond_angles(True)
fast_relax.minimize_bond_lengths(True)
fast_relax.min_type("lbfgs_armijo_nonmonotone")
fast_relax.ramp_down_constraints(False)

if not os.getenv("DEBUG"):
    %time fast_relax.apply(pose)

# Optionally, instead of running this you could reload the saved pose from a previously
#pose = pyrosetta.pose_from_file("minimized_start_pose.pdb")
```

```
INFO:rosetta:basic.thread_manager.RosettaThreadManager: [ WARNING ] A work vector of siz
e zero was passed to the RosettaThreadManager!  Duly returning without doing anything.
INFO:rosetta:basic.thread_manager.RosettaThreadManager: [ WARNING ] A work vector of siz
e zero was passed to the RosettaThreadManager!  Duly returning without doing anything.
INFO:rosetta:basic.thread_manager.RosettaThreadManager: [ WARNING ] A work vector of siz
e zero was passed to the RosettaThreadManager!  Duly returning without doing anything.
INFO:rosetta:basic.thread_manager.RosettaThreadManager: [ WARNING ] A work vector of siz
e zero was passed to the RosettaThreadManager!  Duly returning without doing anything.
CPU times: user 46.5 s, sys: 0 ns, total: 46.5 s
Wall time: 46.5 s
```

Let's check the delta `total_score` per residue after minimizing in the `relax_scorefxn` scorefunction:

In [8]:
```
if not os.getenv("DEBUG"):
    initial_score_res = relax_scorefxn(start_pose)/start_pose.size()
    final_score_res = relax_scorefxn(pose)/pose.size()
    delta_total_score_res = final_score_res - initial_score_res
    print("{0} kcal/(mol*res) - {1} kcal/(mol*res) = {2} kcal/(mol*res)".format(final_s
```

```
2.455328076903503 kcal/(mol*res) - 10.064671501407595 kcal/(mol*res) = -7.60934342450409
2 kcal/(mol*res)
```

We can see that the crystal structure coordinates were not quite optimal according to the `relax_scorefxn` scorefunction. So which model is correct?

By how many Angstroms RMSD did the backbone Cα atoms move?

In [9]:
```
### BEGIN SOLUTION
pyrosetta.rosetta.core.scoring.CA_rmsd(start_pose, pose)
### END SOLUTION
```

Out[9]:
```
0.0
```

For downstream analysis, we want to save the pose:

In [10]:
```
minimized_start_pose = pose.clone()
pyrosetta.dump_pdb(minimized_start_pose, "outputs/minimized_start_pose.pdb")
```

Out[10]:
```
True
```

De Novo Protein Design

Prior to designing chain A, first let's make chain A poly-alanine so that we can re-design the sidechains onto the backbone:

In [11]:
```
# Since we will do direct pose manipulation, first remove the constraints
remove_constraints = pyrosetta.rosetta.protocols.constraint_generator.RemoveConstraints
remove_constraints.add_generator(coord_constraint_gen)
remove_constraints.apply(pose)
```

Obtain chain A and convert it to poly-alanine

In [12]:
```
keep_chA = pyrosetta.rosetta.protocols.grafting.simple_movers.KeepRegionMover(res_start
keep_chA.apply(pose)
polyA_chA = pyrosetta.rosetta.protocols.pose_creation.MakePolyXMover(aa="ALA", keep_pro
polyA_chA.apply(pose)
```

Obtain chains B and C

In [13]:
```
pose_chBC = minimized_start_pose.clone()
keep_chBC = pyrosetta.rosetta.protocols.grafting.simple_movers.KeepRegionMover(res_star
keep_chBC.apply(pose_chBC)
```

Append chains B and C onto the poly-alanine version of chain A

In [14]:
```
pyrosetta.rosetta.core.pose.append_pose_to_pose(pose1=pose, pose2=pose_chBC, new_chain=
```

Pose is now considered to have only 2 chains.

In [15]:
```
pose.num_chains()
```

Out[15]:
```
2
```

Let's re-establish that there are 3 chains.

In [16]:
```
switch_chains = pyrosetta.rosetta.protocols.simple_moves.SwitchChainOrderMover()
switch_chains.chain_order("12")
switch_chains.apply(pose)

print(pose.pdb_info())
print("Now the number of chains = {}".format(pose.num_chains()))
```

```
PDB file name:
 Pose Range  Chain    PDB Range  |   #Residues         #Atoms

0001 -- 0070    A 0001  -- 0070  |   0070 residues;    00703 atoms
0071 -- 0139    B 0071  -- 0139  |   0069 residues;    01187 atoms
0140 -- 0209    C 0140  -- 0209  |   0070 residues;    01212 atoms
                           TOTAL |   0209 residues;    03102 atoms

Now the number of chains = 3
```

Re-apply backbone coordinate constraints:

In [17]:
```
virtual_root.apply(pose)
add_constraints.apply(pose)
```

Have a look at the new pose, chain A in which is ready to be designed!

Next, we need to apply certain movers with our design specifications to activate certain non-pairwise decomposable scoreterms in the `design_scorefxn` scorefunction, that will be implemented when we run the `FastDesign` mover (or any downstream mover that calls the packer).

We will frequently be using the following residue selectors:

In [18]:
```
chain_A = pyrosetta.rosetta.core.select.residue_selector.ChainSelector("A")
chain_BC = pyrosetta.rosetta.core.select.residue_selector.NotResidueSelector(chain_A)
```

Apply the `AddCompositionConstraintMover` mover, which utilizes the `aa_composition` scoreterm. Applying the following mover to the pose imposes the design constraints that the ResidueSelector have 40% aliphatic or aromatic residues other than leucine (i.e. ALA, PHE, ILE, MET, PRO, VAL, TRP, or TYR), and 5% leucines. For documentation, see: https://www.rosettacommons.org/docs/latest/rosetta_basics/scoring/AACompositionEnergy

In [19]:
```
add_composition_constraint = pyrosetta.rosetta.protocols.aa_composition.AddCompositionC
add_composition_constraint.create_constraint_from_file_contents("""
PENALTY_DEFINITION
OR_PROPERTIES AROMATIC ALIPHATIC
NOT_TYPE LEU
FRACT_DELTA_START -0.05
FRACT_DELTA_END 0.05
PENALTIES 1 0 1 # The above two lines mean that if we're 5% below or 5% above the desir
FRACTION 0.4 # Forty percent aromatic or aliphatic, but not leucine
BEFORE_FUNCTION CONSTANT
AFTER_FUNCTION CONSTANT
END_PENALTY_DEFINITION

PENALTY_DEFINITION
TYPE LEU
DELTA_START -1
DELTA_END 1
PENALTIES 1 0 1
FRACTION 0.05 # Five percent leucine
BEFORE_FUNCTION CONSTANT
AFTER_FUNCTION CONSTANT
END_PENALTY_DEFINITION
""")
add_composition_constraint.add_residue_selector(chain_A)
add_composition_constraint.apply(pose)
```

Apply the `AddHelixSequenceConstraints` mover, which utilizes the `aa_composition` scoreterm. By default, this mover adds five types of sequence constraints to the designable residues in each alpha helix in the pose. Any of these behaviours may be disabled or modified by invoking advanced options, but no advanced options need be set in most cases. The five types of sequence constraints are:

- A strong sequence constraint requiring at least two negatively-charged residues in the first (N-terminal) three residues of each alpha-helix.
- A strong sequence constraint requiring at least two positively-charged residues in the last (C-terminal) three residues of each alpha-helix.
- A weak but strongly ramping sequence constraint penalizing helix-disfavoring residue types (by default, Asn, Asp, Ser, Gly, Thr, and Val) throughout each helix. (A single such residue is sometimes tolerated, but the penalty for having more than one residue in this category increases quadratically with the count of helix-disfavouring residues.)
- A weak sequence constraint coaxing the helix to have 10% alanine. Because this constraint is weak, deviations from this value are tolerated, but this should prevent an excessive abundance of alanine residues.
- A weak sequence constraint coaxing the helix to have at least 25% hydrophobic content. This constraint is also weak, so slightly less hydrophobic helices will be tolerated to some degree.

Note that alanine is not considered to be "hydrophobic" within Rosetta.

For documentation, see: https://www.rosettacommons.org/docs/latest/rosetta_basics/scoring/

In [20]:
```
add_helix_sequence_constraints = pyrosetta.rosetta.protocols.aa_composition.AddHelixSeq
add_helix_sequence_constraints.set_residue_selector(chain_A)
add_helix_sequence_constraints.apply(pose)
```

Note: the `aa_repeat` scoreterm works out-of-the-box, and does not need to be applied to the pose to work, it just needs to have a weight of >0 in the scorefunction used by the packer. It imposes a penalty for each stretch of repeating amino acids, with the penalty value depending nonlinearly on the length of the repeating stretch. By default, 1- or 2-residue stretches incur no penalty, 3-residue stretches incur a penalty of +1, 4-residue stretches incur a penalty of +10, and 5-residue stretches or longer incur a penalty of +100. Since the term is sequence-based, it is really only useful for design -- that is, it will impose an identical penalty for a fixed-sequence pose, regardless its conformation. This also means that the term has no conformational derivatives: the minimizer ignores it completely. The term is not pairwise-decomposible, but has been made packer-compatible, so it can direct the sequence composition during a packer run. For documentation, see: https://www.rosettacommons.org/docs/latest/rosetta_basics/scoring/Repeat-stretch-energy

Similarly, the `voids_penalty` scoreterm does not need to be applied to the pose to work, it just needs to have a weight of >0 in the scorefunction used by the packer. For documentation, see: https://www.rosettacommons.org/docs/latest/rosetta_basics/scoring/VoidsPenaltyEnergy

Similarly, the `buried_unsatisfied_penalty` scoreterm does not need to be applied to the pose to work, it just needs to have a weight of >0 in the scorefunction used by the packer. For documentation, see: https://www.rosettacommons.org/docs/latest/rosetta_basics/scoring/BuriedUnsatPenalty

Similarly, the `hbnet` scoreterm does not need to be applied to the pose to work, it just needs to have a weight of >0 (ideally between 1.0 to 10.0) in the scorefunction used by the packer. For documentation, see: https://www.rosettacommons.org/docs/latest/rosetta_basics/scoring/HBNetEnergy

Apply the `AddNetChargeConstraintMover` mover, which utilizes the `netcharge` scoreterm. In this case, we require that the net charge in chain A must be exactly 0.

In [21]:
```
add_net_charge_constraint = pyrosetta.rosetta.protocols.aa_composition.AddNetChargeCons
add_net_charge_constraint.create_constraint_from_file_contents("""
DESIRED_CHARGE 0 #Desired net charge is zero.
PENALTIES_CHARGE_RANGE -1 1 #Penalties are listed in the observed net charge range of -
PENALTIES 1 0 1 #The penalties are 1 for an observed charge of -1, 0 for an observed ch
BEFORE_FUNCTION QUADRATIC #Ramp quadratically for observed net charges of -2 or less.
AFTER_FUNCTION QUADRATIC #Ramp quadratically for observed net charges of +2 or greater.
""")
add_net_charge_constraint.add_residue_selector(chain_A)
add_net_charge_constraint.apply(pose)
```

Specify a custom `relaxscript` that optimizes the `ramp_repack_min` weights to prevent too many alanines from being designed (demonstrated at Pre-RosettaCON 2018):

Specify TaskOperations to be applied to chain A. In this case, let's use the latest LayerDesign implementation (Note: this is still being actively developed) using the XmlObjects class.

In [22]:
```
layer_design_task = pyrosetta.rosetta.protocols.rosetta_scripts.XmlObjects.create_from_
<RESIDUE_SELECTORS>
  <Layer name="surface" select_core="false" select_boundary="false" select_surface="tru
  <Layer name="boundary" select_core="false" select_boundary="true" select_surface="fal
  <Layer name="core" select_core="true" select_boundary="false" select_surface="false"
  <SecondaryStructure name="sheet" overlap="0" minH="3" minE="2" include_terminal_loops
  <SecondaryStructure name="entire_loop" overlap="0" minH="3" minE="2" include_terminal
  <SecondaryStructure name="entire_helix" overlap="0" minH="3" minE="2" include_termina
  <And name="helix_cap" selectors="entire_loop">
   <PrimarySequenceNeighborhood lower="1" upper="0" selector="entire_helix"/>
  </And>
  <And name="helix_start" selectors="entire_helix">
    <PrimarySequenceNeighborhood lower="0" upper="1" selector="helix_cap"/>
  </And>
  <And name="helix" selectors="entire_helix">
    <Not selector="helix_start"/>
  </And>
  <And name="loop" selectors="entire_loop">
    <Not selector="helix_cap"/>
  </And>
</RESIDUE_SELECTORS>
<TASKOPERATIONS>
  <DesignRestrictions name="layer_design">
    <Action selector_logic="surface AND helix_start" aas="EHKPQR"/>
    <Action selector_logic="surface AND helix" aas="EHKQR"/>
    <Action selector_logic="surface AND sheet" aas="DEHKNQRST"/>
    <Action selector_logic="surface AND loop" aas="DEGHKNPQRST"/>
    <Action selector_logic="boundary AND helix_start" aas="ADEIKLMNPQRSTVWY"/>
    <Action selector_logic="boundary AND helix" aas="ADEIKLMNQRSTVWY"/>
    <Action selector_logic="boundary AND sheet" aas="DEFIKLNQRSTVWY"/>
    <Action selector_logic="boundary AND loop" aas="ADEFGIKLMNPQRSTVWY"/>
    <Action selector_logic="core AND helix_start" aas="AFILMPVWY"/>
    <Action selector_logic="core AND helix" aas="AFILMVWY"/>
    <Action selector_logic="core AND sheet" aas="FILVWY"/>
    <Action selector_logic="core AND loop" aas="AFGILMPVWY"/>
    <Action selector_logic="helix_cap" aas="DNST"/>
  </DesignRestrictions>
</TASKOPERATIONS>
""").get_task_operation("layer_design")
```

Also prepare a ResidueSelector for the heterotrimer interface within chains B and C, and the rest of chains B and C. We will use these with `RestrictAbsentCanonicalAASRLT`, `RestrictToRepackingRLT`, and `PreventRepackingRLT` TaskOperations:

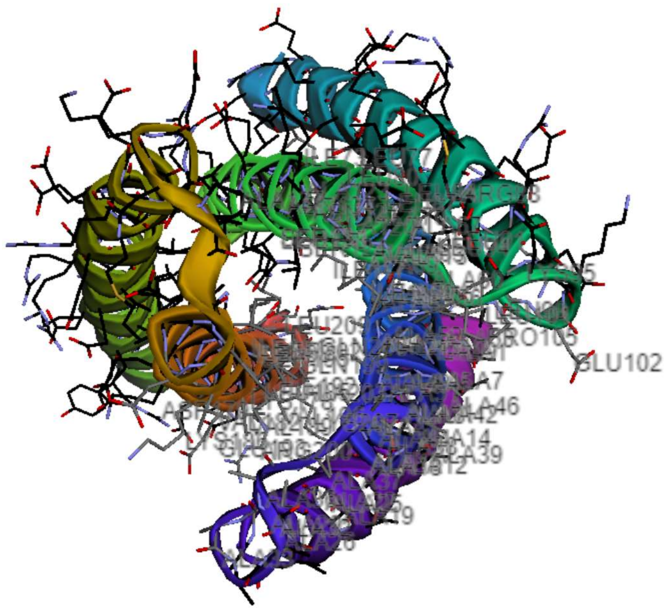
In [23]:
```
interface = pyrosetta.rosetta.core.select.residue_selector.InterGroupInterfaceByVectorS
interface.group1_selector(chain_A)
interface.group2_selector(chain_BC)
chain_BC_interface = pyrosetta.rosetta.core.select.residue_selector.AndResidueSelector(
not_chain_BC_interface = pyrosetta.rosetta.core.select.residue_selector.NotResidueSelec
chain_BC_not_interface = pyrosetta.rosetta.core.select.residue_selector.AndResidueSelec
```

For minimization, we also need the following ResidueSelector:

In [24]:
```
chain_A_and_BC_interface = pyrosetta.rosetta.core.select.residue_selector.OrResidueSele
```

Make sure each ResidueSelector selects the regions as desired (in the following case, the `ResidueSelector` `interface` is visualized:

In [25]:
```
view = viewer.init(pose) \
    + viewer.setStyle() \
    + viewer.setStyle(residue_selector=interface, colorscheme="greyCarbon")
view()
```



Create TaskFactory to be used with the `FastRelax` mover (We use this instead of the FastDesign mover as the constructor allows us to set a relax script):

In [26]:
```
tf.clear()
tf = pyrosetta.rosetta.core.pack.task.TaskFactory()

tf.push_back(pyrosetta.rosetta.core.pack.task.operation.InitializeFromCommandline())
tf.push_back(pyrosetta.rosetta.core.pack.task.operation.IncludeCurrent())
tf.push_back(pyrosetta.rosetta.core.pack.task.operation.NoRepackDisulfides())

# Prevent repacking on chain_BC_not_interface
tf.push_back(pyrosetta.rosetta.core.pack.task.operation.OperateOnResidueSubset(
    pyrosetta.rosetta.core.pack.task.operation.PreventRepackingRLT(), chain_BC_not_inte
```

```python
# Repack only on chain_BC_interface
tf.push_back(pyrosetta.rosetta.core.pack.task.operation.OperateOnResidueSubset(
    pyrosetta.rosetta.core.pack.task.operation.RestrictToRepackingRLT(), chain_BC_inter

# Enable design on chain_A
aa_to_design = pyrosetta.rosetta.core.pack.task.operation.RestrictAbsentCanonicalAASRLT
aa_to_design.aas_to_keep("ACDEFGHIKLMNPQRSTVWY")
tf.push_back(pyrosetta.rosetta.core.pack.task.operation.OperateOnResidueSubset(
    aa_to_design, chain_A))

# Apply layer design
tf.push_back(layer_design_task)

# Convert the task factory into a PackerTask
packer_task = tf.create_task_and_apply_taskoperations(pose)
# View the PackerTask
print(packer_task)
```

```
#Packer_Task

Threads to request: ALL AVAILABLE

resid   pack?   design? allowed_aas
1       TRUE    TRUE    ASP:NtermProteinFull,ASN:NtermProteinFull,SER:NtermProteinFull,T
HR:NtermProteinFull
2       TRUE    TRUE    GLU,HIS,HIS_D,LYS,PRO,GLN,ARG
3       TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
4       TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
5       TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
6       TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
7       TRUE    TRUE    ALA,ASP,GLU,ILE,LYS,LEU,MET,ASN,GLN,ARG,SER,THR,VAL,TRP,TYR
8       TRUE    TRUE    ALA,ASP,GLU,ILE,LYS,LEU,MET,ASN,GLN,ARG,SER,THR,VAL,TRP,TYR
9       TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
10      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
11      TRUE    TRUE    ALA,PHE,ILE,LEU,MET,VAL,TRP,TYR
12      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
13      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
14      TRUE    TRUE    ALA,PHE,ILE,LEU,MET,VAL,TRP,TYR
15      TRUE    TRUE    ALA,ASP,GLU,ILE,LYS,LEU,MET,ASN,GLN,ARG,SER,THR,VAL,TRP,TYR
16      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
17      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
18      TRUE    TRUE    ALA,PHE,ILE,LEU,MET,VAL,TRP,TYR
19      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
20      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
21      TRUE    TRUE    ALA,PHE,ILE,LEU,MET,VAL,TRP,TYR
22      TRUE    TRUE    ALA,ASP,GLU,ILE,LYS,LEU,MET,ASN,GLN,ARG,SER,THR,VAL,TRP,TYR
23      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
24      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
25      TRUE    TRUE    ALA,PHE,ILE,LEU,MET,VAL,TRP,TYR
26      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
27      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
28      TRUE    TRUE    ALA,ASP,GLU,ILE,LYS,LEU,MET,ASN,GLN,ARG,SER,THR,VAL,TRP,TYR
29      TRUE    TRUE    ALA,ASP,GLU,ILE,LYS,LEU,MET,ASN,GLN,ARG,SER,THR,VAL,TRP,TYR
30      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
31      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
32      TRUE    TRUE    ALA,PHE,ILE,LEU,MET,VAL,TRP,TYR
33      TRUE    TRUE    ASP,GLU,GLY,HIS,HIS_D,LYS,ASN,PRO,GLN,ARG,SER,THR
34      TRUE    TRUE    ASP,GLU,GLY,HIS,HIS_D,LYS,ASN,PRO,GLN,ARG,SER,THR
35      TRUE    TRUE    ASP,GLU,GLY,HIS,HIS_D,LYS,ASN,PRO,GLN,ARG,SER,THR
```

```
36      TRUE    TRUE    ASP,GLU,GLY,HIS,HIS_D,LYS,ASN,PRO,GLN,ARG,SER,THR
37      TRUE    TRUE    ASP,GLU,GLY,HIS,HIS_D,LYS,ASN,PRO,GLN,ARG,SER,THR
38      TRUE    TRUE    ASP,GLU,GLY,HIS,HIS_D,LYS,ASN,PRO,GLN,ARG,SER,THR
39      TRUE    TRUE    ASP,ASN,SER,THR
40      TRUE    TRUE    ALA,ASP,GLU,ILE,LYS,LEU,MET,ASN,PRO,GLN,ARG,SER,THR,VAL,TRP,TYR
41      TRUE    TRUE    ALA,ASP,GLU,ILE,LYS,LEU,MET,ASN,GLN,ARG,SER,THR,VAL,TRP,TYR
42      TRUE    TRUE    ALA,ASP,GLU,ILE,LYS,LEU,MET,ASN,GLN,ARG,SER,THR,VAL,TRP,TYR
43      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
44      TRUE    TRUE    ALA,ASP,GLU,ILE,LYS,LEU,MET,ASN,GLN,ARG,SER,THR,VAL,TRP,TYR
45      TRUE    TRUE    ALA,PHE,ILE,LEU,MET,VAL,TRP,TYR
46      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
47      TRUE    TRUE    ALA,ASP,GLU,ILE,LYS,LEU,MET,ASN,GLN,ARG,SER,THR,VAL,TRP,TYR
48      TRUE    TRUE    ALA,ASP,GLU,ILE,LYS,LEU,MET,ASN,GLN,ARG,SER,THR,VAL,TRP,TYR
49      TRUE    TRUE    ALA,ASP,GLU,ILE,LYS,LEU,MET,ASN,GLN,ARG,SER,THR,VAL,TRP,TYR
50      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
51      TRUE    TRUE    ALA,PHE,ILE,LEU,MET,VAL,TRP,TYR
52      TRUE    TRUE    ALA,PHE,ILE,LEU,MET,VAL,TRP,TYR
53      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
54      TRUE    TRUE    ALA,ASP,GLU,ILE,LYS,LEU,MET,ASN,GLN,ARG,SER,THR,VAL,TRP,TYR
55      TRUE    TRUE    ALA,PHE,ILE,LEU,MET,VAL,TRP,TYR
56      TRUE    TRUE    ALA,ASP,GLU,ILE,LYS,LEU,MET,ASN,GLN,ARG,SER,THR,VAL,TRP,TYR
57      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
58      TRUE    TRUE    ALA,PHE,ILE,LEU,MET,VAL,TRP,TYR
59      TRUE    TRUE    ALA,PHE,ILE,LEU,MET,VAL,TRP,TYR
60      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
61      TRUE    TRUE    ALA,ASP,GLU,ILE,LYS,LEU,MET,ASN,GLN,ARG,SER,THR,VAL,TRP,TYR
62      TRUE    TRUE    ALA,ASP,GLU,ILE,LYS,LEU,MET,ASN,GLN,ARG,SER,THR,VAL,TRP,TYR
63      TRUE    TRUE    ALA,ASP,GLU,ILE,LYS,LEU,MET,ASN,GLN,ARG,SER,THR,VAL,TRP,TYR
64      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
65      TRUE    TRUE    ALA,PHE,ILE,LEU,MET,VAL,TRP,TYR
66      TRUE    TRUE    ALA,PHE,ILE,LEU,MET,VAL,TRP,TYR
67      TRUE    TRUE    GLU,HIS,HIS_D,LYS,GLN,ARG
68      TRUE    TRUE    ALA,ASP,GLU,ILE,LYS,LEU,MET,ASN,GLN,ARG,SER,THR,VAL,TRP,TYR
69      TRUE    TRUE    ALA,ASP,GLU,ILE,LYS,LEU,MET,ASN,GLN,ARG,SER,THR,VAL,TRP,TYR
70      TRUE    TRUE    ALA:CtermProteinFull,ASP:CtermProteinFull,GLU:CtermProteinFull,P
HE:CtermProteinFull,GLY:CtermProteinFull,ILE:CtermProteinFull,LYS:CtermProteinFull,LEU:C
termProteinFull,MET:CtermProteinFull,ASN:CtermProteinFull,PRO:CtermProteinFull,GLN:Cterm
ProteinFull,ARG:CtermProteinFull,SER:CtermProteinFull,THR:CtermProteinFull,VAL:CtermProt
einFull,TRP:CtermProteinFull,TYR:CtermProteinFull
71      FALSE   FALSE
72      FALSE   FALSE
73      FALSE   FALSE
74      FALSE   FALSE
75      FALSE   FALSE
76      FALSE   FALSE
77      TRUE    FALSE   LEU
78      FALSE   FALSE
79      FALSE   FALSE
80      TRUE    FALSE   LEU
81      FALSE   FALSE
82      FALSE   FALSE
83      FALSE   FALSE
84      TRUE    FALSE   LEU
85      FALSE   FALSE
86      FALSE   FALSE
87      TRUE    FALSE   LEU
88      TRUE    FALSE   ARG
89      FALSE   FALSE
90      FALSE   FALSE
91      TRUE    FALSE   LEU
```

92	FALSE	FALSE	
93	FALSE	FALSE	
94	TRUE	FALSE	LEU
95	TRUE	FALSE	LYS
96	FALSE	FALSE	
97	FALSE	FALSE	
98	TRUE	FALSE	LEU
99	FALSE	FALSE	
100	FALSE	FALSE	
101	TRUE	FALSE	LEU
102	TRUE	FALSE	GLU
103	FALSE	FALSE	
104	FALSE	FALSE	
105	TRUE	FALSE	PRO
106	FALSE	FALSE	
107	FALSE	FALSE	
108	FALSE	FALSE	
109	FALSE	FALSE	
110	TRUE	FALSE	ILE
111	FALSE	FALSE	
112	FALSE	FALSE	
113	TRUE	FALSE	VAL
114	FALSE	FALSE	
115	FALSE	FALSE	
116	FALSE	FALSE	
117	TRUE	FALSE	ILE
118	FALSE	FALSE	
119	FALSE	FALSE	
120	TRUE	FALSE	ALA
121	FALSE	FALSE	
122	FALSE	FALSE	
123	TRUE	FALSE	ALA
124	FALSE	FALSE	
125	FALSE	FALSE	
126	FALSE	FALSE	
127	FALSE	FALSE	
128	FALSE	FALSE	
129	FALSE	FALSE	
130	FALSE	FALSE	
131	TRUE	FALSE	SER
132	FALSE	FALSE	
133	FALSE	FALSE	
134	FALSE	FALSE	
135	FALSE	FALSE	
136	FALSE	FALSE	
137	TRUE	FALSE	ALA
138	TRUE	FALSE	LEU
139	FALSE	FALSE	
140	FALSE	FALSE	
141	FALSE	FALSE	
142	FALSE	FALSE	
143	FALSE	FALSE	
144	FALSE	FALSE	
145	FALSE	FALSE	
146	FALSE	FALSE	
147	FALSE	FALSE	
148	FALSE	FALSE	
149	FALSE	FALSE	
150	FALSE	FALSE	
151	FALSE	FALSE	

152	FALSE	FALSE	
153	FALSE	FALSE	
154	FALSE	FALSE	
155	FALSE	FALSE	
156	FALSE	FALSE	
157	FALSE	FALSE	
158	FALSE	FALSE	
159	FALSE	FALSE	
160	FALSE	FALSE	
161	FALSE	FALSE	
162	FALSE	FALSE	
163	FALSE	FALSE	
164	FALSE	FALSE	
165	FALSE	FALSE	
166	FALSE	FALSE	
167	FALSE	FALSE	
168	FALSE	FALSE	
169	FALSE	FALSE	
170	FALSE	FALSE	
171	FALSE	FALSE	
172	FALSE	FALSE	
173	FALSE	FALSE	
174	FALSE	FALSE	
175	FALSE	FALSE	
176	FALSE	FALSE	
177	FALSE	FALSE	
178	FALSE	FALSE	
179	FALSE	FALSE	
180	FALSE	FALSE	
181	TRUE	FALSE	ILE
182	TRUE	FALSE	VAL
183	FALSE	FALSE	
184	FALSE	FALSE	
185	TRUE	FALSE	LEU
186	TRUE	FALSE	LYS
187	FALSE	FALSE	
188	TRUE	FALSE	ILE
189	TRUE	FALSE	VAL
190	FALSE	FALSE	
191	FALSE	FALSE	
192	TRUE	FALSE	ILE
193	TRUE	FALSE	GLU
194	FALSE	FALSE	
195	FALSE	FALSE	
196	TRUE	FALSE	VAL
197	FALSE	FALSE	
198	FALSE	FALSE	
199	FALSE	FALSE	
200	TRUE	FALSE	ARG
201	FALSE	FALSE	
202	TRUE	FALSE	SER
203	TRUE	FALSE	ALA
204	FALSE	FALSE	
205	FALSE	FALSE	
206	FALSE	FALSE	
207	TRUE	FALSE	LYS
208	FALSE	FALSE	
209	TRUE	FALSE	LEU
210	TRUE	FALSE	VRT

The PackerTask looks as intended. Now setup the `MoveMapFactory` :

In [27]:

```
# Set up a MoveMapFactory
mmf = pyrosetta.rosetta.core.select.movemap.MoveMapFactory()
mmf.all_bb(setting=False)  # Set to true if needed
mmf.all_bondangles(setting=False)
mmf.all_bondlengths(setting=False)
mmf.all_chi(setting=True)
mmf.all_jumps(setting=False)
mmf.set_cartesian(setting=False)

# Set movemap actions to turn on or off certain torsions, overriding the above defaults
enable = pyrosetta.rosetta.core.select.movemap.move_map_action.mm_enable
disable = pyrosetta.rosetta.core.select.movemap.move_map_action.mm_disable

# Set custom minimizable torsions
mmf.add_bondangles_action(action=enable, selector=chain_A_and_BC_interface)
mmf.add_bondlengths_action(action=enable, selector=chain_A_and_BC_interface)
mmf.add_chi_action(action=enable, selector=chain_A_and_BC_interface)

mmf.add_bondangles_action(action=disable, selector=chain_BC_not_interface)
mmf.add_bondlengths_action(action=disable, selector=chain_BC_not_interface)
mmf.add_chi_action(action=disable, selector=chain_BC_not_interface)
```

Now let's double-check some more `pose` information to verify that we are ready for `FastDesign` :

In [28]:

```
display_pose = pyrosetta.rosetta.protocols.fold_from_loops.movers.DisplayPoseLabelsMove
display_pose.tasks(tf)
display_pose.movemap_factory(mmf)
display_pose.apply(pose)
```

We are ready to setup the `FastRelax` mover:

In [29]:

```
#fast_design = pyrosetta.rosetta.protocols.denovo_design.movers.FastDesign(scorefxn_in=
fast_design = pyrosetta.rosetta.protocols.relax.FastRelax(scorefxn_in=design_scorefxn,
fast_design.cartesian(False)
fast_design.set_task_factory(tf)
fast_design.set_movemap_factory(mmf)
fast_design.min_type("lbfgs_armijo_nonmonotone")
fast_design.ramp_down_constraints(False)
```

Note that this takes ~37min 13s:

In [31]:

```
fast_design.apply(pose)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [31], in <cell line: 1>()
----> 1 fast_design.apply(pose)

IndexError: map::at
```

Save this pose for downstream reference:

In [ ]:
```
designed_pose = pose.clone()
pyrosetta.dump_pdb(designed_pose, "outputs/designed_pose.pdb")
```

Now that we have re-designed chain A, it is strongly recommended to use `FastRelax` with a Cartesian scorefunction to repack and minimize with a realistic scorefuction. Note: this takes ~6min 27s

In [ ]:
```
tf.clear()
tf = pyrosetta.rosetta.core.pack.task.TaskFactory()
tf.push_back(pyrosetta.rosetta.core.pack.task.operation.InitializeFromCommandline())
tf.push_back(pyrosetta.rosetta.core.pack.task.operation.IncludeCurrent())
tf.push_back(pyrosetta.rosetta.core.pack.task.operation.NoRepackDisulfides())
tf.push_back(pyrosetta.rosetta.core.pack.task.operation.OperateOnResidueSubset(
    pyrosetta.rosetta.core.pack.task.operation.RestrictToRepackingRLT(), true_selector)
mm = pyrosetta.rosetta.core.kinematics.MoveMap()
mm.set_bb(False) # Set to true if desired
mm.set_chi(True)
mm.set_jump(False)
fast_relax = pyrosetta.rosetta.protocols.relax.FastRelax(scorefxn_in=relax_scorefxn, st
fast_relax.cartesian(True)
fast_relax.set_task_factory(tf)
fast_relax.set_movemap(mm)
fast_relax.minimize_bond_angles(True)
fast_relax.minimize_bond_lengths(True)
fast_relax.min_type("lbfgs_armijo_nonmonotone") # Cartisian scorefunction
fast_relax.ramp_down_constraints(False)

# To run the FastRelax trajectory.
if not os.getenv("DEBUG"):
    %time fast_relax.apply(pose)

#Or for speed, we will load the pose from a previous trajectory.
pose = pyrosetta.pose_from_file("expected_outputs/designed_relaxed_pose.pdb")
```

Save the pose for downstream analysis:

In [ ]:
```
designed_relaxed_pose = pose.clone()
pyrosetta.dump_pdb(designed_relaxed_pose, "expected_outputs/designed_relaxed_pose.pdb")
```

Let's compare sequences to see what happened:

In [ ]:
```
print(minimized_start_pose.chain_sequence(1))
print(designed_relaxed_pose.chain_sequence(1))
```

View the resulting design!

In [ ]:
```
chA = pyrosetta.rosetta.core.select.residue_selector.ChainSelector("A")
chB = pyrosetta.rosetta.core.select.residue_selector.ChainSelector("B")
chC = pyrosetta.rosetta.core.select.residue_selector.ChainSelector("C")
view = sum(
    [
        viewer.init(designed_relaxed_pose),
        viewer.setStyle(cartoon_color="lightgrey", radius=0.25),
        viewer.setSurface(residue_selector=chA, colorscheme="orangeCarbon", opacity=0.5
        viewer.setSurface(residue_selector=chB, color="greenCarbon", opacity=0.5, surfa
        viewer.setSurface(residue_selector=chB, color="violetCarbon", opacity=0.5, surf
        viewer.setDisulfides(radius=0.25),
        viewer.setZoom(factor=1.5)
    ]
)
view()
```

Analysis:

By how many Angstroms RMSD did the backbone Cα atoms move?

In [ ]:

```

```

What is the delta `total_score` from `minimized_start_pose` to `designed_relaxed_pose` ?

In [ ]:

```

```

What is the per-residue energy difference for each mutated position between `minimized_start_pose` and `designed_relaxed_pose` ?

In [ ]:

```

```

Are the sequence constraints imposed by the `aa_repeat` scoreterm satisfied? Re-write the following python code that counts the number of residue types in chain A to check for the longest stretch of each residue type in the primary amino acid sequence in chain A:

In [ ]:

```
for aa in IUPACData.protein_letters:
    aa_selector = pyrosetta.rosetta.core.select.residue_selector.ResidueNameSelector(st
    aa_and_chain_A = pyrosetta.rosetta.core.select.residue_selector.AndResidueSelector(
    sel_res_count_metric = pyrosetta.rosetta.core.simple_metrics.metrics.SelectedResidu
    sel_res_count_metric.set_residue_selector(aa_and_chain_A)
    print(aa, int(sel_res_count_metric.calculate(designed_relaxed_pose)))
```

Does chain A have 40% percent aromatic or aliphatic (but not leucine) and 5% leucine, satisfying the `aa_composition` scoreterm? For additional practice, re-write the following python implementation using PyRosetta ResidueSelectors and SimpleMetrics:

In [ ]:

```
num_aro_ali = 0
num_leucine = 0

for aa in pose.chain_sequence(1):
    if aa in "WYFAVIMP":
        num_aro_ali += 1
    if aa == "L":
        num_leucine += 1

print("The % aromatic residues in chain A is {0}%".format((num_aro_ali*100)/len(pose.ch
print("The % leucine in chain A is {0}%".format((num_leucine*100)/len(pose.chain_sequen
```

Are the sequence constraints imposed by the `AddHelixSequenceConstraints` mover with the `aa_composition` scoreterm satisfied?

Uses sasa (solvent-accessible surface area) to asses whether there are there more or less voids in `designed_relaxed_pose` as compared to `minimized_start_pose` , satisfying the `voids_penalty` scoreterm.

In [ ]:

```
tf.clear()
tf = pyrosetta.rosetta.core.pack.task.TaskFactory()
tf.push_back(pyrosetta.rosetta.core.pack.task.operation.OperateOnResidueSubset(
    pyrosetta.rosetta.core.pack.task.operation.RestrictToRepackingRLT(), true_selector)
sasa = pyrosetta.rosetta.protocols.simple_filters.TaskAwareSASAFilter()
sasa.task_factory(tf)
print("The sasa of minimized_start_pose is {}".format(sasa.score(minimized_start_pose))
print("The sasa of designed_relaxed_pose is {}".format(sasa.score(designed_relaxed_pose
```

Is the net charge of chain A equal to exactly zero, satisfying the `netcharge` scoreterm?

In [ ]:

```
# One method to calculate net charge of chain A
num_negative = 0
num_positive = 0
for aa in pose.chain_sequence(1):
    if aa in "DE":
        num_negative += 1
    if aa in "KR":
        num_positive += 1
print("The net charge of chain A = {0}".format(num_positive - num_negative))

# Another method to calculate net charge of chain A
test_pose = pose.clone()
keep_chA = pyrosetta.rosetta.protocols.grafting.simple_movers.KeepRegionMover(res_start
keep_chA.apply(test_pose)
net_charge = pyrosetta.rosetta.protocols.simple_filters.NetChargeFilter()
print("The net charge of chain A = {0}".format(net_charge.score(test_pose)))
```

Are there any buried unsatisfied polar atoms, satisfying the `buried_unsatisfied_penalty` scoreterm?

In [ ]:

```
uhb = pyrosetta.rosetta.protocols.rosetta_scripts.XmlObjects.create_from_string("""
<SCOREFXNS>
  <ScoreFunction name="fa_default" weights="ref2015"/>
</SCOREFXNS>
<FILTERS>
  <BuriedUnsatHbonds name="uhb_sc" use_reporter_behavior="true" use_hbnet_behavior="fal
  <BuriedUnsatHbonds name="uhb_bb" use_reporter_behavior="true" use_hbnet_behavior="fal
</FILTERS>
""")
```

In [ ]:

```
print("The number of unsatisfied side-chain heavy atoms in minimized_start_pose is {}".
print("The number of unsatisfied backbone heavy atoms in minimized_start_pose is {}".fo

print("The number of unsatisfied side-chain heavy atoms in designed_relaxed_pose is {}"
print("The number of unsatisfied backbone heavy atoms in designed_relaxed_pose is {}".f
```

Inspect the tracer output from the `BuriedUnsatHbonds` filter, and inspect `designed_relaxed_pose` very closely in PyMol or py3Dmol. Would you agree with the `BuriedUnsatHbonds` filter? How does the number of buried unsatisfied heavy atoms compare to `minimized_start_pose` ?

Packer results are stochastic based on a random number generator, that is after running `pyrosetta.init()` you see:

> rosetta:core.init.random: RandomGenerator:init: Normal mode, seed=937978431 RG_type=mt19937

How do your results compare with your neighbors' results? Ideally you would run the same protocol hundreds of times, and filter them down using Rosetta filters that recaptiulate your design requirements to a number of designs that you can then experimentally validate to answer your biological question.

See Also

Note these may not be available in PyRosetta through code or even by xml (remodel), but they are extremely useful tools when doing denovo protein design - and you should be aware of them.

- **RosettaRemodel**
  - https://www.rosettacommons.org/docs/latest/application_documentation/design/rosettaremod
- **Sewing**
  - https://www.rosettacommons.org/docs/latest/scripting_documentation/RosettaScripts/compos
- **Parametric Design**
  - Previous Workshop!

In [ ]:

```

```

< *De Novo* Parametric Backbone Design | Contents | Index | **Point Mutation Scan** >

Open in Colab