Scaffolding and Motif Grafting

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`)

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
```

alternatively, for bash shell users:

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

Rosetta is assumed to be installed at ${HOME}/rosetta_workshop/rosetta

Introduction

In this tutorial, we want to go through the most important methods in motif grafting and scaffold design. Motif grafting describes techniques to transplant the site of interest on a peptide or protein ("the motif") onto another protein ("the scaffold"). The motif can be the epitope on an antigen that is tageted by an antibody. Or in the case of this tutorial it will be small peptide ligand that will be transferred onto a larger protein. The difficulty of these techniques is to keep the motif in the original conformation, while not disrupting the overall fold of the scaffold protein. However, through the motif grafting especially residues at the connection between motif and scaffold will have to adjust either by a movement in the backbone to maintain secondary structures or also by exchanging the amino acid identity to avoid clashes of side chains. All scaffolding and grafting techniques are protein design protocols that require in-depth sampling and scoring and thorough evaluation, to enhace the likelihood of selecting expressable proteins.

Side Chain Grafting Tutorial

This tutorial is designed to graft functional motifs onto protein scaffolds. There are two different methods that are provided within this tutorial: *Side Chain Grafting* and *Backbone Grafting*. To begin with, a library of protein scaffolds is computationally scanned for possible graft sites. If the motif and scaffold backbones superimpose with very low root mean squared deviation (RMSD<0.5), then only the hot spot side chains need be transplanted from the motif to the corresponding positions in the matching site of the scaffold. This is known as *Side Chain Grafting*. Subsequently, surrounding residues on the scaffold surface that are in contact with the target, or binding partner that binds to the motif, are designed for favorable interactions.

Side Chain Grafting makes the minimal number of changes to the scaffold, increasing the chances of obtaining correctly folded designs during experimental validation. However, Side Chain Grafting often is not possible because the motif and scaffold structures are too dissimilar. In these cases, even though the motif and scaffold may have very different structures, it is still possible to use an alternative method known as *Backbone Grafting*. During Backbone Grafting, the algorithm looks for segments of the scaffold backbone that align closely to the termini of the motif (both N- and C-terminal sides), and then the scaffold segment between these alignment points is replaced by the motif. This technique is extremely versatile, for example, a loop in the scaffold might be replaced by a peptide motif with different secondary structure, or even with a different peptide length. Since the changes to the scaffold structure following Backbone Grafting can disrupt the overall fold, it is important to design the hydrophobic core to support the new backbone structure of the scaffold, followed by design of the protein-protein interface. The Backbone Grafting procedure often introduces many mutations to the scaffold, requiring careful filtering of designs to select those that present quality interfaces and high stability of the new scaffold.

In this tutorial we are going to use a co-crystal structure of Estrogen Receptor (ER) in complex with a helical peptide from a transcriptional co-activator (1GWQ.pdb). We will follow the steps described below to design a protein binder for ER.

1. Definition of the binding motif for interface design
2. Preparing a scaffold database
3. Matching for putative scaffolds (i.e., motif grafting)
4. Sequence design
5. Selection and Improvement of Designs

The first two steps are similar for both Side Chain and Backbone Grafting. Step 3 will be different because the two methods use different algorithms to search for putative scaffolds to graft the binding/functional motif. Last two steps are also similar but more rigorous for Backbone Grafting.

Create a directory in the SideChainGraft directory called my_files and switch to that directory. Although many files you need for the tutorial are located in the input_files directory, we will work from my_files for the rest of this section of the tutorial.

```
cd ~/rosetta_workshop/tutorials/scaffolding/SideChainGraft/
mkdir my_files
cd my_files
```

A. Prepare the input files for motif grafting.

In this step we will define the binding/functional motif that we want to graft onto a protein scaffold.

1. Download the co-complex from the Protein Databank (PDB). This complex is under the PDB ID "1gwq".

The 1GWQ.pdb file is provided in the input_files directory. However the instructions for downloading this PDB file are also provided below.

a. Go to http://www.rcsb.org and type 1gwq in the search bar. b. Click on "Download Files" on the right side of the page, then "PDB Format". c. Save the PDB file in the my_files directory as 1GWQ.pdb. d. Prepare the PDBs for running through Rosetta.

In general before running a PDB through Rosetta you should remove water molecules and all ligands that are non-essential to your protocol. We will use an automated script to do this processing.

The 1GWQ.pdb file contains a dimer of ER-alpha bound to helical peptides. We want to pull the target structure, i.e., ER-alpha (chain A, renamed as context.pdb) and the binding motif structure (chain C, renamed as motif.pdb) from the PDB 1GWQ. For future reference, context.pdb serves as the binding partner you would be trying to maintain binding affinity to, and motif.pdb would serve as the template for identifying scaffold targets.

```
python2 ~/rosetta_workshop/rosetta/tools/protein_tools/scripts/clean_pdb.py 1GWQ A
python2 ~/rosetta_workshop/rosetta/tools/protein_tools/scripts/clean_pdb.py 1GWQ C
mv 1GWQ_A.pdb context.pdb
mv 1GWQ_C.pdb  motif.pdb
```

B. Preparing the Scaffold Database (A scaffold database is provided in the scaffolds/ directory, for this tutorial skip these steps!)

To prepare an inclusive scaffold database that can be searched for a variety of structural motifs, you can download structures from the PDB (http://www.rcsb.org) based on the following four criteria using advanced search module: (1) crystal structures with high-resolution x-ray diffraction data (< 2.5 A), (2) the proteins had been reported to be expressable in E. coli (this simplifies later experimental characterization), (3) a single protein chain in the asymmetric unit (MotifGraft only works with monomeric scaffolds as grafting targets), and (4) no bound ligands or modified residues.

In some circumstances, a focused scaffold library may produce more useful matches. For our particular example, the peptide motif has an alpha-helical conformation. Therefore, we prepared a small focused scaffold library of 18 helical proteins.

Further, online tools exist to search for suitable scaffolds (you can find to options in the appendix of this tutorial).

The scaffold PDB files were formatted for ROSETTA and subjected to an energy minimization step as described below:

1. Create a directory in my_files/ called scaffolds/

   ```
   mkdir scaffolds/
   ```

2. In my_files/scaffolds download and clean the scaffold structures you found using the same clean_pdb.py scripts as described above.

3. Make a list of all pdb files to be used as scaffold.

   ```
   command ls *.pdb > scaffolds.list
   ```

4. Relax the pdb structures while constraining the structure to its initial coordinates. Note that the "\" at the end of line 1 only marks the end of the line. If you copy-paste the command into the terminal, remove the "\" before running so that the command is one line.

   ```
   relax.linuxgccrelease -ignore_unrecognized_res \
    -relax:constrain_relax_to_start_coords -ex1 -ex2 -use_input_sc  -l  scaffolds.list
   ```

C. Motif Grafting and Interface Design

Motif grafting and interface design are distinct conceptual steps, but due to the flexibility of the RosettaScripts framework, both can be included in a single computational step.

Since scaffold matching is different for Side Chain and Backbone Grafting, we are first going to use the Side Chain grafting procedure. It is recommended that you attempt Side Chain Grafting before Backbone Grafting for your own functional motif, as it requires less changes in the protein scaffold, increasing the chances of obtaining correctly folded designs during experimental validation.

Now that we have our input pdb files for both the motif and context ready, along with the scaffolds database to scan and put the motif onto putative scaffolds, we will perform Side Chain Grafting using the MotifGraft_sc.xml script.

Copy the MotifGraft_sc.xml script from scripts/ directory to my_files/ directory.

```
cp ../scripts/MotifGraft_sc.xml .
```

Make a list of all pdb files being used as scaffold.

```
command ls ../scaffolds/*.pdb > scaffolds.list
```

Execute the MotifGraft_sc.xml rosettascripts using the following command. Again, if you copy-paste the command, make sure to remove any "\" at the end of each line so that the command is one line. It will take approximately 15 minutes to process all the 18 input protein strutcures.

```
rosetta_scripts.linuxgccrelease \
-l scaffolds.list  -use_input_sc  -ex1  -ex2  -nstruct 1 \
-parser:protocol MotifGraft_sc.xml
```

The option "-nstruct 1" is used to generate one design for each scaffold. However, if the scaffold and the motif are too different, Rosetta will not generate any output for that particular scaffold. Check the outputs to see how many scaffolds were not accepted for the grafting.

To generate more than one design, you will need to use the MultiplePoseMover. See the Rosetta Wiki for documentation.

The expected_output/ directory has designs from a previous run. One should look at the designs in Pymol.

For further explanation of the options used in the XML script, see this methods paper Silva et al, 2016.

D. Selection and Improvement of Designs

To date, no computational method has been developed that can predict with perfect accuracy which designs will be functional when challenged experimentally. Therefore, it is wise to proceed with designed sequences that have good metrics by multiple criteria.

1. Designs are initially filtered based on calculated metrics for interface quality, including a favorable binding energy (ddG < 0 ROSETTA energy units, ideally the energy should be lower than the native interface from which the motif was taken), high shape complementarity (Sc>0.65), and a low number of buried unsatisfied hydrogen-bonding atoms. In the XML scripts above, these filters report to a score file and will also be appended at the end of any ROSETTA output PDBs. If you are generating more than one designs per scaffold (for example, 10 designs per scaffold), you can select them based on the total score before looking for ddG, Sc and other metrics.

2. Once a set of designs has been selected based on the calculated metrics, it is important to perform human-guided inspection of the designed structures. There are many qualities of interfaces that are apparent to structural biologists that are not captured in standard metrics. Two common defects in ROSETTA-designed structures that are very important to avoid are: *i*) buried charged residues and *ii*) under-packed interfaces dominated by alanine residues.

3. Reverting designs back to native residues: It is also important to consider whether the designed scaffold will fold to its intended structure; having a spectacular interface on a computational model is irrelevant if the protein cannot fold in an experimental setting. This is particularly problematic for designed interfaces that have a large surface area dominated by hydrophobic residues. It is generally assumed that the probability of a designed sequence properly folding is inversely correlated with the number of mutations imposed on the scaffold during the design process. Therefore, it is beneficial to be conservative and make as few mutations as possible by reverting residues back to their native identities in a post-design stage. For this step, we will revert one of the designs (1ji6_0001_0001.pdb) back to its native sequence, 1ji6_0001.pdb in complex with the target (context.pdb).

   ```
   cat context.pdb ../scaffolds/1ji6_0001.pdb >nativecplx.pdb
   ```

   Note that ../scaffolds/1ji6_0001.pdb is the structure that was used to generate 1ji6_0001_0001.pdb, by grafting the motif.pdb onto a scaffold from the PDB.

   ```
   revert_design_to_native.linuxgccrelease \
   -revert_app:wt nativecplx.pdb -revert_app:design 1ji6_0001_0001.pdb -ex1 -ex2 \
   -use_input_sc > revert.log
   ```

   Read the final lines of revert.log to see what residues ROSETTA selected to revert, and which reversions were too high energy to implement. Additionally open the files in PyMol to see the differences yourself.

4. Manually adjusting designs: The user may wish to correct a number of frequent problematic features in ROSETTA designs, such as hydrophobic residues at the water-exposed interface edge, revert designed residues back to their native identities, mutate buried charged residues to hydrophobics, etc. There are no hard rules for manually improving designs; it is simply a matter of the designers preference and experience.

5. Filtering Designs based on folding probability: Many designed sequences will not fold correctly when experimentally tested. We have found structure prediction to be a powerful filter; the designed models when subjected to structure prediction calculations should yield similar structures to the designed models. If structure prediction returns an alternative conformation, or fails to converge on an energy minimum in a conformational landscape, then it is unlikely that the designed sequence will correctly fold.

Backbone Grafting Tutorial

All the steps in Backbone Grafting tutorial are same as in the Side Chain Grafting Tutorial. We have changed the scaffolds database for this part. The MotifGraft_bb.xml script incorporates the backbone grafting algorithm for scaffold matching.

1. Create my_files directory in BackboneGraft directory.

   ```
   cd ~/rosetta_workshop/tutorials/scaffolding/BackboneGraft
   mkdir my_files
   cd my_files/
   ```

2. Motif and Context pdb files

   ```
   cp ../input_files/motif.pdb .
   ```

```
cp ../input_files/context.pdb .
```

3. Make Scaffolds list.

```
command ls  ../scaffolds/*.pdb > scaffolds.list
```

4. Copy Motif Grafting and Sequence Design XML

```
cp ../scripts/MotifGraft_bb.xml .
```

5. Run Backbone Grafting

   execute the backbone grafting script using following commandline.

```
rosetta_scripts.linuxgccrelease \
-l scaffolds.list -use_input_sc -nstruct 1 -parser:protocol MotifGraft_bb.xml
```

   Note: running MotifGraft_bb.xml takes longer than MotifGraft_sc.xml (8 scaffolds, with 1 output model each takes about ~15 minutes).

6. Selection and Improvement of Designs

   Designs from backbone grafting require extra attention, as the engineering of a protein core to support the grafted motif can be challenging. Therefore, one should check to see how motif placement has changed the structure of initial scaffold using PyMol or other protein visualization tool.

FunFolDes Tutorial

The **FunFolDes** protocol aims to tackle a very specific issue of motif grafting: the need for a relatively high structural similarity between the structural motif of interest and the region in the scaffold into which it has to be grafted. To do so, instead of moving the motif from one context to another, **FunFolDes** generates an unfolded pose around the motif, folding it back using the scaffolds's topology as guide. Through this process, the protocol forces the new topology to adapt to the structural restrictions imposed by the motif while keeping the overall topology of the scaffold. Basically, FunFolDes grafts the motif in the fold of the scaffold (as in comparison to its structure). A binding target (i.e. the antibody) can also be added during the folding process, ensuring that the final designs will display the motif in a way compatible with the target of interest. The major power of FunFolDes lies in its ability to graft motifs of different sequence length and to swap motif sequence order, which is very useful for motifs of higher complexity.

FunFolDes was designed as a flexible protocol capable of interacting with a wide variety of RosettaScript components. As such, in this tutorial we will focus on understanding the key rules of the protocol as well as its two nuclear components: then `NubInitioMover` and the `NubInitioLoopClosureMover`. To do so, we will graft graft RSVF's site II epitope (motif - PDB ID: **3IXT**) into the structure of the A6 protein of the Antennal Chemosensory system from the moth *Mamestra brassicae* (scaffold - PDB ID: **1KX8**) and comment on the different parts of the script.

First, we will discuss how `residue labeling` is used to communicate the intrinsic rules of FunFolDes between the different RosettaScripts components. Then we will list the necessary files to execute the protocol. After that, we will walk through the prepared FunFolDes script and show how to properly execute it. Finally, we will show some critical score terms for evaluating the generated results.

1. Set-up files and inputs

For Running FunFolDes, a motif from a donor is required. The file itself can contain other structural data. Additionally, the file can also contain the structural data of a binding target. However, a target is not necessary to execute the protocol. If a target is added, it has to be in the same PDB file as the donor. This ensures that the positioning between the two structures will remain intact during the process. As of the writing of this tutorial, FunFolDes is capable of processing a target as long as it is a protein, DNA/RNA or small molecule. Through the application of the `ResidueSelector`, the donor can come from one or more segments belonging to one or more chain identifiers, the only limitation being that donor and target cannot share the same chain identifier.

A PDB file containing the structure of the carrier scaffold is required. Because FunFolDes depends on the *ab-initio* protocol, the carrier scaffold will also be used to create the necessary fragment files to guide the folding process. In the following section we will comment on how to search for putative carrier scaffolds.

Change to the FunFolDes folder in your scaffolding tutorial by using:

```
cd ../../FunFolDes/
mkdir my_files
cd my_files
```

First, we will take a look at the input structures, motif, context and scaffold. The motif (residues 255-276 of chain C in 3IXT.pdb) will be grafted on the residues 79-100 in 1KX8.pdb.

Therefore, copy the pdbs in your folder and open them in PyMOL.

```
cp ../input_files/3IXT.pdb .
cp ../input_files/1KX8.pdb .
pymol *pdb
```

2. The FunFolDes protocol

The key step of the FunFolDes protocol, the folding of unfolded protein around the motif, is contained inside a single mover called `NubInitioMover`. After the folded pose is created the protocol needs to communicate with other movers on the rules that must be followed to ensure the proper placement and residue types of the motif. This is important when considering that the protocol is capable of grafting motifs of different sequence lengths as well as swapping their sequence order when dealing with multi-segment motifs. To do so, `NubInitioMover` takes and expands upon the residue labeling system introduced by `MotifGraftMover`, as we have seen above in the Sidechain and Backbone grafting. Each label is assumed to define a behavior in terms of backbone and side chain movement and residue type variation. Rosetta can interpret these labels as ResidueSelectors. ResidueSelectors allow us to specify which exact action Rosetta will perform on the specified residues. For example, the `MOVE_MAP_FACTORIES` controls actions that are performed on the backbone movement and the chi angles, while `TASKOPERATIONS` can be applied to Movers (and other Taskoperations) to specifiy on which residue subset the action will be performed.

Copy the FunFolDes.xml from the input folder into your working folder and then open it in a text-editor of your choice.

```
cp ../input_files/FunFolDes.xml .
```

In Table 1, you will find the labels we are interested in. There are two main categories of Residue_Selectors in the FunFolDes protocol, which control 1) general behavior of the designated parts of the proteins and 2) execute specific tasks on selected subsets of the protein, e.g. design. Try to identify each label and then go through the script and track, which Mover calls which Taskoperation or ResidueSelector. Does the script capture the expected behavior in table 1?

label	targeted residues	expected behavior
MOTIF	Residues from the motif	None by itself
TEMPLATE	Residues from the template	Residues are allowed bb/chi movement and design
HOTSPOT	key residues in the motif	Residues can not move or be designed
COLDSPOT	Non-key residues in the motif	Residues have chi movement and can be designed
FLEXIBLE	Movable residues in the motif	Residues have bb/chi movement but cannot be designed
CONTEXT	Residues from the target (e.g. antibody)	Residues are not allowed to move or be designed

The NubInitioMover represents the central keystone of the FunFolDes protocol and is responsible for building the unfolded pose aroung the donor and generates the folded pose. The **Nub**, which is a static segment (the donor) around which the pose will be folded, holds information on the order of the inserted segments and allows for the rearrangement of the motif's chains. It further takes care of the N and C terminal adjustment to the backbone and controls which residues in the motif are allowed to change during design. The execution of the NubInitioMover is followed up by a design step, for which we use FastDesign in order to generate the most stable version of the folded protein.

The `NubInitioMover` is in its core an *ab initio* folding protocol in Rosetta and requires fragments for this purpose. Contrary to normal fragment picking, in the context of FunFolDes the aim is not sampling the structural space of the

query sequence but ensuring that the final structure is as close as possible to the original scaffold, while conforming to the required changes to fit the donor motif. As such, the fragments are expected to be obtained from the structure of the scaffold itself, not from its sequence. The FunFoldDes protocol normally uses the StructFragmentMover, which automatically picks fragments on the fly. However, due to the limited time we have for this tutorial, we will provide fragments in the tutorial here. (In the appendix, you can find a stand-alone RosettaScripts xml-protocol, which can be used for Fragment picking.)

Copy the provided fragment into your working directory:

```
cp ../input_files/1kx8.200.3mers .
cp ../input_files/1kx8.200.9mers .
```

Now, we are ready to execute FunFolDes:

```
rosetta_scripts.linuxgccrelease \
    -s 1KX8.pdb -nstruct 5 -ignore_waters -ignore_unrecognized_res \
    -parser:script_vars donor=3IXT.pdb frags3=1kx8.200.3mers frags9=1kx8.200.9mers \
    -out:prefix FFL_ -parser:protocol FunFolDes.xml
```

You will produce five models. Each run will approximatly last for 30 mintues, therefore we will continue with the evaluation using pregenerated models. If you use protocols like FunFolDes in a research project you will want to make sure that you generate at least 5000-10000 models.

3. Evaluation

It is very important to ensure that the protein is correctly folded and fulfills all criteria that we have for a protein from nature. This is even more important for FunFolDes in comparison to the SideChain and Backbone grafting protocols that we already worked through, as FunFolDes contains an *ab inito* mover and samples more conformational and sequence space then the other protocols. Therefore, a high number of decoys are needed to obtain a number of designs with acceptable confidence. The protocol that we executed for the workshop already contained a number of filters, which evaluated certain aspects that might influence decisions on designs, for example:

- driftRMSD: compares the final design with the folded pose, saved just after the folding process. This measure provides an insight of the backbone changes applied to the scaffold in the post-process stage of the protocol.(Note: this will throw an error, when the sequence length changed.)
- design_score: individualized score of the designed pose (especially of interest when working with a target).
- PackStat: a packing score

Further, the classical scores that we used in other protocols are important, too. You can check the binding energy to your target structure by calculating the dG_separated value and the total_score, which gives you an idea of the overall stability of your protein.

For the purpose of this tutorial, you will find 30 pre-generated models in the ../output_files/. Copy the pdb-files and the corresponding score.sc file into your working directory using:

```
cp ../output_files/*pdb .
cp ../output_files/*sc .
```

You can check for the best scoring model by total_score using:

```
awk '{print $2,$35}' score.sc | sort -nk1
```

You should also check for other score terms like the design_score, driftRMSD and packstat score by modifying the command above, so it outputs other score terms. In the ensemble there should be two models that are profoundly worse than the others. Which scores help you identifing them? Take a look at them in PyMOL.

Further you should take a look at the score range, for example how many REU difference do you find for the total_score?

When your runs are finished, you can open them and compare them to the pre-generated results.

Appendix

Additional information

Methods for scaffold searches

Smotif geometric loop definition

This would be an approach when working with continuous super-secondary structure motifs (i.e. two secondary structures linked by a single loop). A simple way to obtain multiple scaffold candidates is submitting the donor motif to the Frag'R'Us server http://www.bioinsilico.org/FRAGRUS.

MASTER search

Actual RMSD searches can be performed with **MASTER**. This approach is especially useful when dealing with multi-segment motifs. **MASTER** not only performs quite fast searches but matches are independent of the order of each individual segment in sequence. This increases the expected number of putative scaffolds and, as we will see, is not an issue for FunFolDes.

MASTER download and install instructions (Linux only) can be found at https://grigoryanlab.org/master/. Briefly, after installation, to prepare a MASTER search, the necessary steps will be: (a) download the PDB database, (b) format it in MASTER's format, (c) convert the query structure to MASTER's format, (d) perform the search.

Download the PDB database:

```
cd $DATABASES
mkdir pdb
rsync -rlpt -v -z --delete --files-from=sample.txt --port=33444 \
        rsync.rcsb.org::ftp_data/structures/divided/pdb/pdb/
```

Format the database

```
find pdb -name "*ent.gz" > db.list
createPDS --type target --pdbList db.list
find pdb -name "*pds" > master.list
```

Format the query structure

```
createPDS --type query --pdb motif.pdb
```

Search

```
master --query motif.pds --targetList master.list --rmsdCut 5 --matchOut motif.master.list
```

For simplicity the example code directly transforms the structures downloaded from the PDB into MASTER's PDS format. However, separating the individual chains of each file would be more useful in some scenarios.

Keep in mind that, at the end of a search, the only thing you'll need (and get) is (a) the PDB files of the most promising scaffold matches and (b) the residue range of the acceptor region, MASTER will provide those ranges in Rosetta count.

Fragment picking stand alone for FunFolDes

This is a fragment picking stand-alone script. Fragment picking can be time intensive and you not necessarily want to do it inside your FunFolDes.xml. However, you have the option to include the fragment picking in your FunFolDes run, when you don't care about the increased time. As a resouce we provide a stand alone script:

```
<ROSETTASCRIPTS>
  <RESIDUE_SELECTORS>
    # Selectors to clean up what we will not use of the scaffold.
    <Index name="TMP" resnums="10A-105A" />
    <Not name="!TMP" selector="TMP" />
    <ProteinResidueSelector name="PROTEIN" />
    <Not name="!PROTEIN" selector="PROTEIN" />
    <Or name="!TMP_OR_PROT" selectors="!TMP,!PROTEIN" />
  </RESIDUE_SELECTORS>
```

```
  <MOVERS>
    <DeleteRegionMover name="delete" residue_selector="!TMP_OR_PROT" />
    <StructFragmentMover name="FragmentPicker"
        prefix="wauto"
        vall_file="vall.jul19.2011.gz"
        output_frag_files="1"
        small_frag_file="wauto.200.3mers"
        large_frag_file="wauto.200.9mers"
        frag_weight_file="scores.cfg" />
  </MOVERS>
  <PROTOCOLS>
    <Add mover="delete" />
    <Add mover="FragmentPicker" />
  </PROTOCOLS>
</ROSETTASCRIPTS>
```

Rosetta Remodel Tutorial

The remodel application is a Rosetta flexible loop-modeling tool that is tailor-made for protein design -- in this case, motif design. It uses a simple interface, the *blueprint*, to coordinate various protein modeling tasks, which can include backbone building, sidechain design, disulfide pairing, and constraint assignments during run-time, making the application a versatile tool for motif design. For this tutorial, we will reuse the previous example, ER bound to a helical peptide from a transcriptional co-activator, but this time we will perform design on the binding pocket on chain A of 1GWQ to search for stabilizing mutations within the binding interface. This is a rather limited example of Rosetta Remodel, but given it's flexible design capabilites, the purpose of this tutorial is to orient you with the application input files and limitations.

1. Create my_files directory in RosettaRemodel directory.

   ```
   cd ~/rosetta_workshop/tutorials/scaffolding/RosettaRemodel/
   mkdir my_files
   cd my_files/
   ```

2. Prepare the input PDBs. Like our previous examples we must first relax our input structure, **1gwq.pdb**. Both **1gwq.pdb** and the relaxed structure **1gwq_0001.pdb** are provided for you in the ../input_files/ directory. It is important to note that Rosetta Remodel can only design one chain during each simulation and the chain considered for design must be consecutively numbered, starting from 1. Even though our relaxed structure contains two chains, we will renumber the pdb file so that chain A begins with 1 using the option flag "-out:file:renumber_pdb". If we were to consider chain C for design, we would first want to remove all information for chain A, and the renumber chain C starting with 1. *Skip this step*, but for reference, the following was used to prepare the input files:

   ```
   cp ../input_files/1gwq.pdb .
   relax.default.linuxgccrelease \
   -s 1gwq.pdb -ignore_unrecognized_res -use_input_sc -constrain_relax_to_start_coords \
   -relax:fast -out:file:renumber_pdb
   ```

3. Prepare the blueprint file. In general, a blueprint file contains three columns, where the first column is the residue position, the second column is the residue identity -- in one letter codes, such a A for alanine, and the third is the design option to be performed on that residue. To fully understand all the design options included in a blueprint file, it is highly recommended that you refer to Huang, P-S et al., 2011 (Reference 1 at the end of this section). In our example blueprint file **1gwqA.blueprint**, we will only consider residues for design that are within 5 Angstoms of the helical peptide ligand, which includes residues 50, 52, 53, 56, 57, 62, 67, 68, 70, 71, 74, 75, 233, 234, 237 and 238 on chain A with the design specification ". ALLAA" in the third column, which means that the backbone is fixed and all amino acid substitutions are allowing during design. In addition, we will also specify to repack flanking resides with the design specification ". NATAA". Otherwise, all other residues will be fixed.

   ```
   cp ../input_files/1gwqA.blueprint .
   ```

If you were to make the blueprint file yourself, first make a general blueprint file, specifying all positions as fixed.

```
~/rosetta_workshop/rosetta/tools/remodel/getBluePrintFromCoords.pl \
  -pdbfile 1gwq_0001.pdb > test.blueprint
```

Now specify the correct design options. The **1gwqA.blueprint** has already been amended to include the correct design options. However, it you were to make the changes yourself, you would need to change the third column option "." in the test.blueprint file to include the design options "ALLAA" or "NATAA" on the lines corresponding to the residue number:

```
...
49 L . NATAA
50 V . ALLAA
51 H . NATAA
52 M . ALLAA
53 I . ALLAA
54 N . NATAA
55 W . NATAA
56 A . ALLAA
57 K . ALLAA
58 R . NATAA
...
61 G . NATAA
62 F . ALLAA
63 V . NATAA
...
66 T . NATAA
67 L . ALLAA
68 H . ALLAA
69 D . NATAA
70 Q . ALLAA
71 V . ALLAA
72 H . NATAA
73 L . NATAA
74 L . ALLAA
75 E . ALLAA
76 C . NATAA
...
232 Y . NATAA
233 D . ALLAA
234 L . ALLAA
235 L . NATAA
236 L . NATAA
237 E . ALLAA
238 M . ALLAA
239 L . NATAA
...
```

4. Run RosettaRemodel

   Executing the command below will take much longer than the length of this session, so please don't try to run it as is. All output files have been generated for the following analysis step. However, if you do want to run it, change the options "-num_trajectory 50" to ""-num_trajectory 1" and "-save_top 10" to "-save_top 1". Since our input pdb contains two chains, it is important to note that Rosetta Remodel will work on the first chain unless the flag "-chain" specifies the chain to remodel. Moreover, Rosetta Remodel can only work on one chain at a time (at least by default), and depending on which version of Rosetta you use, it is possible to get an error regarding missing residues. Even if you only have one chain, it is suggested you use the "-run::chain" option.

```
remodel.default.linuxgccrelease \
-s 1gwq_0001.pdb -remodel:blueprint 1gwqA.blueprint \
-run::chain A -extrachi_cutoff 1 -ex1 -ex2 -use_input_sc \
-num_trajectory 50 -save_top 10 -use_clusters false -find_neighbors
```

For those who are specificially interested in protein design techniques for motif transplanation onto another backbone without docking the ligand to the binding site, this can be achieved using the "0 x I" (insertion) notation in the blueprint file, indicating the location and length of the motif graft site to insert into your backbone of choice, and the specified option "-remodel:domainFusion:insert_segment_from_pdb" which specfies the PDB file of the motif sequence you would like to insert. For the motif PDB file, the PDB does not need to be renumbered so that it starts with 1, but the PDB file that serves as the graft site will need to be renumbered.

If you are using manual mode (which most likely you will be), it is recommended that you assign all positions included in the rebuilt segment, or else they will be turned into valines since valine is the default residue during the centroid phase. The blueprint file is essentially like a resfile, where you can declare your design specifications to guide the all-atom phase -- and Rosetta Remodel works best when you explicitly tell it what to do!

If you would like to have more examples of use cases for Rosetta Remodel, refer to the Rosetta Remodel wiki documentation (listed below) for additional information and documentation.

5. Analyze results

Rosetta Remodel handles its own file I/O and only uses the job_distributor to launch processes. Therefore, typically you would expect all output in the format of XXXX_0001.pdb with the 0001 suffix increasing incrementally to match the total number of requested output models. That is not the case in Rosetta Remodel, which outputs 1.pdb, 2.pdb, etc., for the total number of models you requested with "-save_top ". Rosetta Remodel will only output a single file as XXXX_0001.pdb, which represents the lowest scoring model from the "-save_top" models.

The type of analysis you would want to perform is dependent on what your desired outcome is. Since Rosetta Remodel automatically chooses the lowest energy model, you could just trust this model for your future experiments. You should, however, do further analysis of your models to understand what changes Rosetta made to the original structure/sequence. Since we performed design on the ligand binding pocket of ER, we should start looking at two things: i) The RMSD of the models to the starting model, and ii) the sequence identity of the designed models. Calculating the RMSD for design models is likely not very informative, but for loop insertion or motif grafting, it is very important to check if your starting backbone retains its original geometry to some degree. To calculate the RMSD of the output models to the input model 1gwq_0001.pdb:

```
~/rosetta_workshop/rosetta/tools/protein_tools/scripts/score_vs_rmsd_full.py \
```

-n 1gwq_0001.pdb -d ../expected_output/*.pdb

cat score_vs_rmsd_align_all_model.tsv

Since this tutorial only used design, a better way to assess the output models is look at the type of designed mutations Rosetta introduced in the ER binding pocket. You can do this either by generating a weblogo or by performing a multiple sequence alignment of the output sequences. To generate a weblogo:

cp ../input_files/1gwqA.resfile .

```
~/rosetta_workshop/rosetta/tools/protein_tools/scripts/deep_analysis \
--format pdf --title "Designed residue frequencies" --labels sequence --debug \
```

--prefix designs_ --native 1gwq_0001.pdb --seq --res 1gwqA.resfile -s nd ../expected_output/*pdb

Or to do a multiple sequence alignment:

```
cat *_A.fasta > output.fasta
clustalw output.fasta
```

By looking at the weblogo of the designed positions, **weblogo.png** in the expected_output/ directory, Rosetta Remodel quickly converges onto a single sequence, with the exception of positions 71 and 75, which helps explain why all designs have an 0.11 Angstrom RMSD to the starting relaxed model. With the multiple sequence alignment file, **output.aln**, you can see that 6.pdb is the only model with a differing sequence. However, if you notice in position 233, Rosetta Remodel completely replaces the native aspartic acid for a glutamic acid, which may be due to Rosetta's sampling bias. For this tutorial, it is likely best to visualize changes to side chain contacts to see why Rosetta replaced the native sequence. To do this open PyMOL and load the output PDB files (1.pdb - 10.pdb) and the starting PDB, 1gwq_0001.pdb. You can compare similarities of the side chain contacts by selecting the designed residue and the residue it makes contact with, followed by "Show -> side chain -> as sticks" and then "Action -> find -> any contacts -> within 3 (or 4 for some) Angstroms", which should illustrate the number and length of each of the side chain contacts. By comparing the designed and native side chain contacts, it is generally observed that Rosetta favors increased number of side chain interactions, which presumably correlates with a higher binding interface stability. To go one step further, you should look at individual residue score breakdowns. To do this:

cp ../input_files/compare.list cp ../expected_output/remodel_pdbs/*pdb .

```
residue_energy_breakdown.linuxgccrelease \
-in:file:l compare.list -out:file:silent per_res.sc
```

The output file **expected_output/per_res.sc** is a table of all onebody and pairwise interaction scores for each residue in each model. The score file was converted to a csv file, **per_res.csv** so that you can create a plot of all of the designed positions and the onebody total score values of each of the models. A plot of the per-residue total scores has been provided for you, **expected_output/per_res_scores.png**, but if you would like to generate/change the plot in anyway, you can start with this:

```
cp expected_output/per_res.csv .
R
> install.packages("ggplot2","cowplot")
> library(ggplot2, cowplot)
> setwd("~/rosetta_workshop/tutorials/scaffolding/Rosetta_Remodel/")
> data <- read.csv("per_res.csv", header = T, sep = ",")
> data$resi1 <- as.factor(data$resi1)
> designs = subset(data, resi1=="50"|resi1=="52"|resi1=="53"|resi1=="56"|resi1=="57"| \
  resi1=="62"|resi1=="67"|resi1=="68"|resi1=="70"|resi1=="71"|resi1=="74"|resi1=="75"| \
  resi1=="233"|resi1=="234"|resi1=="237"|resi1=="238")
> onebody = subset(designs, restype2=="onebody")
> ggplot(onebody, aes(x = description, y = total, fill = pose_id)) + geom_col() + \
  facet_grid(~ resi1) + labs(x = "Designed residue position", \
  y = "Rosetta Total Score per residue (in REU)") + theme(axis.text.x = element_blank(), \
  strip.background = element_rect(color = "black", fill = "white"))
```

Note: "> " represents the R environment, and all commands must be typed in an R shell. After you have finished making your plot, type "q()" to exit out of the R shell.

By looking at the individual onebody score terms, it becomes apparent that Rosetta Remodel per residue scores are the same, or at least very similar to the native structure, with some key exceptions. For position 68 on chain A, Rosetta scores the native histidine as highly unfavorable, whereas the models have a much lower-scoring aspartic acid. If you were to visualize the side chain interactions of 1gwq_0001.pdb and one of the output models (say 1.pdb), you can see that in the original sequence the native histidine does not make any contacts with its neighboring side chains. By replacing the histidine with an aspartic acid, the now negatively charged residue is able to make a polar contact with the histine in the ligand (H247). However, by replacing the native histidine with a negatively charged aspartic acid, there are now three negatively charged residues in close vicinity, that is D68, S158, and S159, in the model sequence, which may or may not be an issue. This is where also looking at the pairwise per residue scores would be useful to see if the native or designed sequence is likely to be favorable or not. For position 233, even though Rosetta Remodel replaces the native aspartic acid everytime, the native sequence scores more favorably than any of the designed E233 positions, and in this case you should probably assume that the D233E mutation is not a functionally relevant mutation. Time permitting, go back and take a look at positions 71 and 75, which Rosetta does not converge

on a single sequence or score.

Useful References

For Scaffold Design:

1. Silva, D., Correia, B.E., and Procko, E. (2016) Motif-driven Design of Protein-Protein Interactions. *Methods Mol. Biol.* 1414:285-304.
2. Azoitei, M.L., Ban, Y.A., Julien, J., Bryson, S., Schroeter, A., Kalyuzhniy, O., Porter, J.R., Adachi, Y., Baker, D., Pai, E.F., and Schief, W.R. (2012) Computational Design of High-Affinity Epitope Scaffolds by Backbone Grafing of a Linear Epitope. *J Mol. Biol.* 415:175-192.
3. Azoitei, M.L., Correia, B.E., Ban, Y.A., Carrico, C., Kalyuzhniy, O., Chen, L., Schroeter, A., Huang, P., McLellan, J.S., Kwong, P.D., Baker, D., Strong, R.K., Schief, W.R. (2011) Computation-Guided Backbone Grafting of a Discontinuous Motif onto a Protein Scaffold. *Science* 334:373-376.

For Rosetta Remodel:

1. Huang, P.S., Ban, Y.E., Richter, F., Andre, I., Vernon, R., Schief, W.R., Baker, D. (2011) RosettaRemodel: A Generalized Framework for Flexible Backbone Protein Design. *PLoS One* 6(8):e24109.
2. https://www.rosettacommons.org/docs/latest/application_documentation/design/rosettaremodel