RosettaAntibodyDesign

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

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

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

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

Authors

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

Citation

Rosetta Antibody Design (RAbD): A General Framework for Computational Antibody Design, PLOS Computational Biology, 4/27/2018

Jared Adolf-Bryfogle, Oleks Kalyuzhniy, Michael Kubitz, Brian D. Weitzner, Xiaozhen Hu, Yumiko Adachi, William R. Schief, Roland L. Dunbrack Jr.

Overview

RosettaAntibodyDesign (RAbD) is a generalized framework for the design of antibodies, in which a user can easily tailor the run to their project needs. **The algorithm is meant to sample the diverse sequence, structure, and binding space of an antibody-antigen complex.** An app is available, and all components can be used within RosettaScripts for easy scripting of antibody design and incorporation into other Rosetta protocols.

The framework is based on rigorous bioinformatic analysis and rooted very much on our recent clustering of antibody CDR regions. It uses the **North/Dunbrack CDR definition** as outlined in the North/Dunbrack clustering paper. A new clustering paper will be out in the next year, and this new analysis will be incorporated into RAbD.

The supplemental methods section of the published paper has all details of the RosettaAntibodyDesign method. This manual serves to get you started running RAbD in typical use fashions.

Algorithm

Broadly, the RAbD protocol consists of alternating outer and inner Monte Carlo cycles. Each outer cycle consists of randomly choosing a CDR (L1, L2, etc.) from those CDRs set to design, randomly choosing a cluster and then a structure from that cluster from the database according to the input instructions, and optionally grafting that CDR's structure onto the antibody framework in place of the existing CDR (**GraftDesign**). The program then performs N rounds of the inner cycle, consisting of sequence design (**SeqDesign**) using cluster-based sequence profiles and structural constraints, energy minimization, and optional docking. Each inner cycle structurally optimizes the backbone and repacks side chains of the CDR chosen in the outer cycle as well as optional neighbors in order to optimize interactions of the CDR with the antigen and other CDRs.

Backbone dihedral angle (CircularHarmonic) constraints derived from the cluster data are applied to each CDR to limit deleterious structural perturbations. Amino acid changes are typically sampled from **profiles derived for each CDR cluster in PyIgClassify**. Conservative amino acid substitutions (according to the BLOSUM62 substitution matrix) may be performed when too few sequences are available to produce a profile (e.g., for H3). After each inner cycle is completed, the new sequence and structure are accepted according to the Metropolis Monte Carlo criterion. After N rounds within the inner cycle, the program returns to the outer cycle,

at which point the energy of the resulting design is compared to the previous design in the outer cycle. The new design is accepted or rejected according to the Monte Carlo criterion.

If optimizing the antibody-antigen orientation during the design (dock), SiteConstraints are automatically used to keep the CDRs (paratope) facing the antigen surface. These are termed **ParatopeSiteConstraints**. Optionally, one can enable constraints that keep the paratope of the antibody around a target epitope (antigen binding site). These are called **ParatopeEpitopeSiteConstraints** as the constraints are between the paratope and the epitope. The epitope is automatically determined as the interface residues around the paratope on input into the program, however, any residue(s) can be set as the epitope to limit unwanted movement and sampling of the antibody. See the examples and options below.

More detail on the algorithm can be found in the published paper.

General Setup and Inputs

1. Antibody Design Database

   This app requires the Rosetta Antibody Design Database. A database of antibodies from the original North Clustering paper is included in Rosetta and is used as the default . An updated database (which is currently updated bi-yearly) can be downloaded here: http://dunbrack2.fccc.edu/PyIgClassify/.

   It should be placed in `~/rosetta_workshop/rosetta/gmain/database/sampling/antibodies/` It is recommended to use this up-to-date database for production runs. For this tutorial, we will use the database within Rosetta.

2. Starting Structure

   The protocol begins with the three-dimensional structure of an antibody-antigen complex. Designs should start with an antibody bound to a target antigen (however optimizing just the antibody without the complex is also possible). Camelid antibodies are fully supported. This structure may be an experimental structure of an existing antibody in complex with its antigen, a predicted structure of an existing antibody docked computationally to its antigen, or even the best scoring result of low-resolution docking a large number of unrelated antibodies to a desired epitope on the structure of a target antigen as a prelude to de novo design.

   The program CAN computationally design an antibody to anywhere on the target protein, but it is recommended to place the antibody at the target epitope. It is beyond the scope of this program to determine potential epitopes for binding, however servers and programs exist to predict these. Automatic SiteConstraints can be used to further limit the design to target regions.

3. Model Numbering and Light Chain identification

   The input PDB file must be renumbered to the AHo Scheme and the light chain gene must be identified. This can be done through the PyIgClassify Server.

   On input into the program, Rosetta assigns our CDR clusters using the same methodology as PyIgClassify. The RosettaAntibodyDesign protocol is then driven by a set of command-line options and a set of design instructions provided as an input file that controls which CDR(s) are designed and how. Details and example command lines and instruction files are provided below.

   The gene of the light chain should always be set on the command-line using the option `-light_chain`, these are either lamda or kappa. PyIgClassify will identify the gene of the light chain.

   For this tutorial, the starting antibody is renumbered for you.

4. Notes for Tutorial Shortening

   Always set the option, `-outer_cycle_rounds` to 5 in order to run these examples quickly. The default is 25. We include this in our common options file that is read in by Rosetta at the start. We will only be outputting a single structure, but typical use of the protocol is with default settings of `-outer_cycle_rounds` and an `nstruct` of at least 1000, with 5000-10000 recommended for jobs that are doing a lot of grafting. For De-novo design runs, one would want to go even higher. Note that the Docking stage increases runtime significantly as well.

The total number of rounds is outer_cycle_rounds * nstruct.

5. General Notes

   This tutorial assumes that you have Rosetta added to your PATH variable, as this is how Rosetta is generally run. 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

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

All of our common options for the tutorial are in the common file that you will copy to your working direc
Rosetta will look for this file in your working directory or your home folder in the directory `$HOME/.ros
See this page for more info on using rosetta with custom config files: <https://www.rosettacommons.org/doc

All tutorials have generated output in output_files and their approximate time to finish on a single (core
```

Tutorial

Tutorial A: General Design

In many of these examples, we will use the application for simplicity, however, ALL options are available using the AntibodyDesignMover. XML lines are given if you are more comfortable in RosettaScripts. A skeleton XML file is provided for you if you want to go the RosettaScripts route. Note that the mover does not run InterfaceAnalyzer to report interface scores at the end of the run in the tutorial version of Rosetta. This is changed in the most current version of Rosetta. AntibodyDesignMover

You must still use the `-light_chain` command-line option.

Lets copy the files we need first:

```
mkdir work_dir
cd work_dir
cp ../input_files/my_ab.pdb .
cp ../input_files/color_cdrs.pml .
cp ../input_files/rabd.xml .
cp ../input_files/common .
```

You are starting design on a new antibody, that is not bound to the antigen in the crystal. This is difficult and risky, but we review how one could go about this anyway. We start by selecting a framework. Here, we use the trastuzumab framework as it expresses well, is thermodynamically stable with a Tm of 69.5 degrees, and has been shown repeatedly that it can tolerate CDRs of different sequence and structure. Note that the energy of the complex is high as we are starting from a manual placement of the antibody to antigen. If we relax the structure too much, we will fall into an energy well that is hard to escape without significant sampling.

We are using an arbitrary protein at an arbitrary site for design. The PDB of our target is 1qaw. 1qaw is an oligomer of the TRP RNA-Binding Attenuation Protein from Bacillus Stearothermophilus. It is usually a monomer/dimer, but at its multimeric interface is a tryptophan residue by itself. As TRP rises, the probability of finding a fully oligomeric complex increases, which in this state helps to decrease TRP production through the TRP-operon regulatory mechanism.

It's a beautiful protein, with a cool mechanism. We will attempt to build an antibody to bind to two subunits to stabilize the dimeric state of the complex in the absence of TRP. Note that denovo design currently takes a large amount of processing power. Each tutorial below is more complex than the one before it. The examples we have for this tutorial are short runs to show HOW it can be done, but more outer_cycle_rounds and nstruct would produce far better models than the ones you will see here - as we will need to sample the relative orientation of the antibody-antigen complex through docking, the CDR clusters and lengths, the internal backbone degrees of freedom of the CDRs, as well as the sequence of the CDRs and possibly the framework. As you can tell, just the sampling problem alone is difficult. However, this will give you a basis for using RAbD on your own.

1. Sequence Design

   Using the application is as simple as setting the `-seq_design_cdrs` option. This simply designs the CDRs of the heavy chain using cdr profiles if they exist for those clusters during flexible-backbone design. If the clusters do not exist (as is the case for H3 at the moment), we use conservative design by default.

   ```
   antibody_designer.linuxgccrelease -s my_ab.pdb \
       -seq_design_cdrs L1 L3 -light_chain kappa -nstruct 5 -out:prefix tutA1_
   ```

   ```
   Main Mover: <AntibodyDesignMover name="RAbD" seq_design_cdrs="L1,L3">
   ```

   This will take a few minutes (227 seconds on my laptop). Ouptut structures and scores are in ../output_files if you wish to copy them over. Score the input pose using the InterfaceAnalayzer application

   ```
   InterfaceAnalyzer.linuxgccrelease -s my_ab.pdb -interface LH_ABCDEFGIJKZ -out:prefix native_
   ```

   Take a look at our scorefile (located in ~/rosetta_workshop/rosetta/gmain/source/tools), sort according to the interface energy, dG separated, which is calculated by InterfaceAnalyzerMover. The score is calculated by scoring the complex, separating the antigen from the antibody, repacking side-chains at the interface, and then taking the difference is score - i.e. the dG.

   ```
   scorefile.py --scores dG_separated --output tab tutA1_score.sc | sort  -k2 -k1
   ```

   Is it better than our input pose?

   ```
   scorefile.py --scores dG_separated --output tab tutA1_score.sc native_score.sc | sort -k2 -k1
   ```

2. Graft Design

   Now we will be enabling graft design AND sequence design on L1 and L3 loops. With an nstruct of 5, we are doing 25 design trials total.

   ```
   antibody_designer.linuxgccrelease -s my_ab.pdb -graft_design_cdrs L1 L3 \
       -seq_design_cdrs L1 L3 -light_chain kappa -nstruct 5 -out:prefix tutA2_ -overwrite
   ```

   ```
   Main Mover: <AntibodyDesignMover name="RAbD" seq_design_cdrs="L1,L3" graft_design_cdrs="L1,L3">
   ```

   This will take a about 2-3 times as long as sequence design, as grafting a non-breaking loop takes time. This was 738 seconds on my laptop. Ouptut structures and scores are in ../output_files if you wish to copy them over instead.

   Typically, we require a much higher -outer_cycle_rounds and nstruct to see anything significant. Did this improve energies?

   ```
   scorefile.py --scores dG_separated --summary tutA1_score.sc tutA2_score.sc
   scorefile.py --scores dG_separated --output tab native_score.sc \
       tutA1_score.sc tutA2_score.sc | sort -k2 -k1
   ```

   Take a look at the lowest (dG) scoring pose in pymol - do you see any difference in L1 and L3 loops there? Do they make better contact than what we had before?

   Lets take a look in pymol.

```
pymol my_ab.pdb tutA2_*
@color_cdrs.pml
center full_epitope
```

How different are the L1 and L3 loops? Have any changed length?

Open the PDB file of the lowest energy in a text editor. Go to the end of the file. Have the clusters changed from the native? To get the native cluster, open the PDB and look at the REMARK lines. THis was generated using the identify_cdr_clusters application.

3. Basic De-novo run

   Here, we want to do a denovo-run (without docking), starting with random CDRs grafted in instead of whatever we have in the antibody to start with (only for the CDRs that are actually undergoing graft-design). This is useful, as we start the design with very high energy and work our way down. Note that since this is an entirely new interface for our model protein, this interface is already at a very high energy - and so this is less needed, but it should be noted how to do this. (139 seconds on my laptop)

```
antibody_designer.linuxgccrelease -s my_ab.pdb -graft_design_cdrs L1 L3 \
    -seq_design_cdrs L1 L3 -light_chain kappa -random_start -nstruct 1 -out:prefix tutA3_

Main Mover: <AntibodyDesignMover name="RAbD" graft_design_cdrs="L1,L3" seq_design_cdrs="L1,L3"
                                                                        random_start="1"/>
```

   Would starting from a random CDR help anywhere? Perhaps if you want an entirely new cluster or length to break a patent or remove some off target effects? We will use it below to start de novo design with docking. Since the CDRs are native, these are already fairly energetically favorable.

4. RosettaScripts RAbD Framework Components

   This tutorial will give you some exprience with your own XML antibody design protocol using the RosettaAntibdyDesign components. We will take the light chain CDRs from a malaria antibody and graft them into our antibody. In the tutorial we are interested in stabilizing the grafted CDRs in relation to the whole antibody, instead of interface design to an antigen.

   We will graft the CDRs in, minimize the structure with CDR dihedral constraints to not purturb the CDRs too much, and then design the framework around the CDRs while designing the CDRs and neighbors. The result should be mutations that better accomodate our new CDRs. This can be useful for humanizing potential antibodies or framework switching, where we want the binding properties of certain CDRs, but the stability or immunological profile of a different framework.

   1. Copy the Files

      ```
      cp ../input_files/ab_design_components.xml .
      cp ../input_files/malaria_cdrs.pdb .
      ```

      Take a look at the xml. We are using the *AntibodyCDRGrafter* to do the grafting of our CDRs. We then add constraints using _CDRDihderalConstraintMover_s for each CDR. Next, we do a round of pack/min/pack using the *RestrictToCDRsAndNeighborsOperation* and the CDRResidueSelector. This task operation controls what we pack and design. It first limits packing and design to only the CDRs and its neighbors. By specifying the *design_framework=1* option we allow the neighbor framework residues to design, while the CDRs and antigen neighbors will only repack. If we wanted to disable antigen repacking, we would pass the *DisableAntibodyRegionOperation* task operation. Using this, we can specify any antibody region as *antibody_region*, *cdr_region*, or *antigen_region* and we can disable just design or both packing and design.

      These task operations allow us to chisel exactly what we want to design in antibody, sans a residue-specific resfile (though we could combine these with one of them!). These TaskOperations will eventually also become *ResidueSelectors*, to increase their utility further.

      Finally, we use the new SimpleMetric system to obtain our final sequence of the CDRs to compare to our native antibody as well as pymol selections of our CDRs.

      - SimpleMetrics

More Documentation is available here:

- RosettaScripts
- Antibody and CDR specific operations
- Antibody modelling and design movers

2. Run the protocol or copy the output (357 seconds).

```
rosetta_scripts.linuxgccrelease -parser:protocol ab_design_components.xml \
    -input_ab_scheme AHO_Scheme -cdr_definition North \
    -s my_ab.pdb -out:prefix tutA4_ \
    -nstruct 5 -score:weights ref2015_cart -check_cdr_chainbreaks false
```

3. Look at the score file. Are the sequences different between what we started with? How about the interaction energies?

Tutorial B: Optimizing Interface Energy (opt-dG)

1. Optimizing dG

   Here, we want to set the protocol to optimize the interface energy during Monte Carlo instead of total energy. The interface energy is calculated by the InterfaceAnalyzerMover through a specialized MonteCarlo called *MonteCarloInterface*. This is useful to improve binding energy and generally results in better interface energies . Resulting models should still be pruned for high total energy. This was benchmarked in the paper, and has been used for real-life designs to the HIV epitope (825 seconds).

```
antibody_designer.linuxgccrelease -s my_ab.pdb -graft_design_cdrs L1 L3 \
    -seq_design_cdrs L1 L3 -light_chain kappa -mc_optimize_dG  -nstruct 5 \
    -out:prefix tutB1_

Main Mover: <AntibodyDesignMover name="RAbD" seq_design_cdrs="L1,L3" graft_design_cdrs="L1,L3"
                                                                    mc_optimize_dG="1" />
```

   Use the scorefile.py script to compare this to tutorial A2. Are the interface energies better? Has the Shape Complementarity improved (sc score) improved?

2. Optimizing Interface Energy and Total Score (opt-dG and opt-E)

   Here, we want to set the protocol to optimize the interface energy during Monte Carlo, but we want to add some total energy to the weight. Because the overall numbers of total energy will dominate the overall numbers, we only add a small weight for total energy. This has not been fully benchmarked, but if your models have very bad total energy when using opt-dG - consider using it. (892)

```
antibody_designer.linuxgccrelease -s my_ab.pdb  -graft_design_cdrs L1 L3 \
    -seq_design_cdrs L1 L3 -light_chain kappa -mc_optimize_dG \
    -mc_total_weight .01 -mc_interface_weight .99 -nstruct 5 -out:prefix tutB2_
```

   Use the scorefile.py script to compare total energies (`total_score`) of this run vs the one right before it. Are the total scores better?

Tutorial C: Towards DeNovo Design: Integrated Dock/Design

This tutorial takes a long time to run as docking is fairly slow - even with the optimizations that are part of RAbD. PLEASE USE THE PREGENERATED OUTPUT. The top 10 designs from each tutorial and associated scorefiles of a 1000 nstruct cluster run are in the output directory. Note that we are starting these tutorials with a pre-relaxed structure in order to get more reliable rosetta energies. Since we are running a large nstruct, we will escape the local energy well that this leads us into.

1. RosettaDocking

   In this example, we use integrated RosettaDock (with sequence design during the high-res step) to sample the antibody-antigen orientation, but we don't care where the antibody binds to the antigen. Just that it binds. IE - No Constraints. The RAbD protocol always has at least Paratope SiteConstraints enabled to

make sure any docking is contained to the paratope (like most good docking programs). This takes too long to run, so PLEASE USE THE OUTPUT GENERATED FOR YOU. We will use opt-dG here and for these tutorials, we will be sequence-designing all cdrs to begin to create a better interface. Note that sequence design happens whereever packing occurs - Including during high-resolution docking.

```
antibody_designer.linuxgccrelease -s my_ab_relaxed.pdb -graft_design_cdrs L1 L2 L3 H1 H2 \
    -seq_design_cdrs L1 L2 L3 H1 H2 H3 -light_chain kappa \
    -mc_optimize_dG -do_dock -nstruct 1000 -random_start -out:prefix tutC1_

Main Mover: <AntibodyDesignMover name="RAbD" mc_optimize_dG="1" do_dock="1"
                seq_design_cdrs="L1,L2,L3,H1,H2,H3" graft_design_cdrs="L1,L2,L3,H1,H2"/>
```

Use pymol to load the files

```
cp -r ../output_files/top10_C1/ .
cp ../output_files/tutC1_score.sc .
pymol my_ab.pdb top10_C1/*
@color_cdrs.pml
center full_epitope
```

Where is the antibody in the resulting designs? Are the interfaces restricted to the Paratope? Has the epitope moved relative to the starting interface?

2. Auto Epitope Constraints

   Allow Dock-Design, incorporating auto-generated SiteConstraints to keep the antibody around the starting interface residues. These residues are determined by being within 6A to the CDR residues (This interface distance can be customized). Again, these results are provided for you.

```
antibody_designer.linuxgccrelease -s my_ab_relaxed.pdb -graft_design_cdrs L1 L3 \
    -seq_design_cdrs L1 L2 L3 H1 H2 H3 -light_chain kappa \
    -mc_optimize_dG -do_dock -use_epitope_constraints -nstruct 1000 -out:prefix tutC2_

Main Mover: <AntibodyDesignMover name="RAbD" mc_optimize_dG="1" do_dock="1" use_epitope_csts="1"
                seq_design_cdrs="L1,L2,L3,H1,H2,H3" graft_design_cdrs="L1,L2,L3,H1,H2"/>
```

   Use pymol to load the files

```
cp -r ../output_files/top10_C2/ .
cp ../output_files/tutC2_score.sc .
pymol my_ab.pdb top10_C2/*
@color_cdrs.pml
center full_epitope
```

   How do these compare with with the previous tutorial? Are the antibodies closer to the starting interface? Are the scores better?

3. Specific Residue Epitope Constraints

   Allow Dock-Design, as above, but specify the Epitope Residues and Paratope CDRs to guide the dock/design to have these interact. For now, we are more focused on the light chain. We could do this as a two-stage process, where we first optimize positioning and CDRS of the light chain and then the heavy chain or simply add heavy chain CDRs to the paratope CDRs option.

```
antibody_designer.linuxgccrelease -s my_ab_relaxed.pdb -graft_design_cdrs L1 L3 \
    -seq_design_cdrs L1 L2 L3 H1 H2 H3 -light_chain kappa -do_dock -use_epitope_constraints \
    -paratope L1 L3 -epitope 38J 52J 34K 37K -nstruct 1000 -out:prefix tutC3_

Main Mover: <AntibodyDesignMover name="RAbD" mc_optimize_dG="1" do_dock="1" use_epitope_csts="1"
                epitope_residues="38J,52J,34K,37K" paratope_cdrs="L1,L3"
                seq_design_cdrs="L1,L2,L3,H1,H2,H3" graft_design_cdrs="L1,L2,L3,H1,H2"/>
```

   Again, load these into Pymol.

```
cp -r ../output_files/top10_C3 .
cp ../output_files/tutC1_score.sc .
pymol my_ab.pdb top10_C3/*
@color_cdrs.pml
center full_epitope
```

Now that we have specified where we want the interface to be and are additionally designing more CDRS, how do the enegies compare? Are we starting to get a decent interface with the lowest energy structure?

How do these compare with the previous runs?

Tutorial D: Advanced Settings and CDR Instruction File Customization

Once again, all output files are in ../output_files. Please use these if you want as many of these take around 10 minutes to run.

1. CDR Instruction File

   More complicated design runs can be created by using the Antibody Design Instruction file. This file allows complete customization of the design run. See below for a review of the syntax of the file and possible customization. An instruction file is provided where we use conservative design on L1 and graft in L1, H2, and H1 CDRs at a longer length to attempt to create a larger interface area. More info on instruction file syntax can be found at the end of this tutorial. (764 seconds on my laptop)

   ```
   cp ../input_files/my_instruction_file.txt .
   cp ../input_files/default_cdr_instructions.txt .
   ```

   Take a look at the default CDR instructions. These are loaded by default into Rosetta. There is syntax examples at the end of the file.

   ```
   antibody_designer.linuxgccrelease -s my_ab.pdb \
       -graft_design_cdrs L1 H2 H1 -seq_design_cdrs L1 L3 H1 H2 H3 -light_chain kappa \
       -cdr_instructions my_instruction_file.txt -random_start -nstruct 5 -out:prefix tutD1_

   Main Mover: <AntibodyDesignMover name="RAbD" instruction_file="my_instruction_file.txt"
            seq_design_cdrs="L1,L3,H1,H2,H3" graft_design_cdrs="L1,H2,H1" random_start="1"
   ```

2. Dissallow AAs and the Resfile

   Here, we will disallow ANY sequence design into Proline residues and Cysteine residues, while giving a resfile to further LIMIT design and packing as specific positions. These can be given as 3 or 1 letter codes and mixed codes such as PRO and C are accepted. Note that the resfile does NOT turn any residues ON, it is simply used to optionally LIMIT design residue types and design and packing positions. Resfile syntax can be found here: Resfiles Note that specifying a resfile and dissalowing aa are only currently available as cmd-line options that are read by RAbD.

   Runtime is approximately 200 seconds.

   ```
   cp ../input_files/my_resfile.resfile .
   ```

   Take a look at the resfile. Can you describe what it is we are doing with it?

   ```
   antibody_designer.linuxgccrelease -s my_ab.pdb -seq_design_cdrs L1 L3 H1 H2 H3 \
       -light_chain kappa -resfile my_resfile.resfile -disallow_aa PRO CYS \
       -nstruct 5 -out:prefix tutD2_

   Main Mover: <AntibodyDesignMover name="RAbD" seq_design_cdrs="L1,L3,H1,H2,H3" />
   ```

3. Mintype

   Here, we will change the mintype to relax. This mintype enables Flexible-Backbone design. Our default is to use min/pack cycles, but relax typically works better. However, it also takes considerably more time! This tutorial takes about 339 seconds for one struct!

```
antibody_designer.linuxgccrelease -s my_ab.pdb -seq_design_cdrs L1 L3 H1 H3 \
    -light_chain kappa -resfile my_resfile.resfile -disallow_aa PRO CYS -mintype relax \
    -nstruct 1 -out:prefix tutD3_

Main Mover: <AntibodyDesignMover name="RAbD" seq_design_cdrs="L1,L3,H1,H3" mintype="relax"/>
```

4. Framework Design

   Finally, we want to allow the framework residues AROUND the CDRs we will be designing and any interacting antigen residues to design as well here. In addition, we will disable conservative framework design as we want something funky (this is not typically recommended and is used here to indicate what you CAN do. Note that we will also design the interface of the antigen using the `-design_antigen` option. This can be useful for vaccine design. Note that these design options are cmd-line ony options currently. Approx 900 second runtime.

   ```
   antibody_designer.linuxgccrelease -s my_ab.pdb -seq_design_cdrs L1 L3 H1 H3 \
       -light_chain kappa -resfile my_resfile.resfile -disallow_aa PRO CYS \
       -mintype relax -design_antigen -design_framework \
       -conservative_framework_design false -nstruct 1 -out:prefix tutD4_

   Main Mover: <AntibodyDesignMover name="RAbD" seq_design_cdrs="L1,L3,H1,H3" mintype="relax"/>
   ```

5. H3 Stem, kT, and Sequence variablility.

   Finally, we want increased variability for our sequence designs. So, we will increase number of sampling rounds for our lovely cluster profiles using the `-seq_design_profile_samples` option, increase kT, and allow H3 stem design.

   We will enable H3 Stem design here, which can cause a flipping of the H3 stem type from bulged to non-bulged and vice-versa. Typically, if you do this, you may want to run loop modeling on the top designs to confirm the H3 structure remains in-tact. Note that once again, these sequence-design specific options must be set on the cmd-line.

   Description of the `seq_design_profile_samples` option (default 1): "If designing using profiles, this is the number of times the profile is sampled each time packing done. Increase this number to increase variability of designs - especially if not using relax as the mintype."

   This tutorial takes approx 450 seconds.

   ```
   antibody_designer.linuxgccrelease -s my_ab.pdb -seq_design_cdrs L1 L2 H3 \
       -graft_design_cdrs L1 L2 -light_chain kappa -design_H3_stem -inner_kt 2.0 \
       -outer_kt 2.0 -seq_design_profile_samples 5 -nstruct 5 -out:prefix tutD5_

   Main Mover: <AntibodyDesignMover name="RAbD" seq_design_cdrs="L1,L3,H1,H3" mintype="relax"
                                                          inner_kt="2.0" outer_kt="2.0"/>
   ```

   How different is the sequence of L1,L2, and H3 from our starting antibody?

You should now be ready to explore and use RosettaAntibodyDesign on your own. Congrats! Thanks for going through this tutorial!

Reference Manual

Antibody Design CDR Instruction File

The Antibody Design Instruction File handles CDR-level control of the algorithm and design. It is used to create the CDRSet for sampling whole CDRs from the PDB, as well as fine-tuning the minimization steps and sequence design strategies.

For example, in a redesign project, we may only want to design a particular CDR and explore the local conformations within its starting CDR cluster, or we may simply want to optimize the sequence of a CDR or set of CDRs using our cluster profiles. These examples can be accomplished using the CDR Instruction File. This file uses a simple

syntax where each CDR is controlled individually in the first column, and then a rudimentary language controls each part of the antibody design machinery. The file controls which CDRs are sampled, what the final CDRSet will be, whether sequence design is performed and how, and which minimization type will be used to optimize the structure and sequence for each CDR. Some of these commands can also be controlled via command-line options for simple-to-setup runs. Any option set via command-line (such as `-graft_design_cdrs` and `-seq_design_cdrs` will override anything set in the instruction file)

Syntax

The CDR Instruction file is composed of tab or white-space delimited columns. The first column on each line specifies which CDR the option is for. For each option, 'ALL' can be given to control all of the CDRs at once. ALL can also be used in succession to first set all CDRs to something and then one or more CDRs can be set to something else in succeeding lines. Commands are not lower- or upper-case-sensitive. A '#' character at the beginning of a line is a file comment and is skipped.

First column - *CDR*

Second Column - *TYPE*

Other columns can be used to specify lists, etc.

Instruction Types

Type	Description
GraftDesign	Instructions for the Graft-based Design stage
SeqDesign	Instructions for Sequence Design
CDRSet	Instructions for which structures end up in the set of structures from which to sample during GraftDesign. option follows the syntax: `CDR CDRSet option`
MinProtocol	Instructions for Minimization

Example

```
#H3 no design; just flexible motion of the loop

#SET All to design first
ALL GraftDesign ALLOW
ALL SeqDesign ALLOW

#Fix H3 and disallow GraftDesign for H1 and L2
L2 GraftDesign FIX
H1 GraftDesign FIX

#Disallow GraftDesign and SeqDesign for H3
H3 FIX

ALL MinProtocol MINTYPE relax

ALL CDRSet EXCLUDE PDBIDs 1N8Z 3BEI

#ALL CDRSet EXCLUDE Clusters L1-11-1 L3-9-cis7-1 H2-10-1
```

General Design Syntax

Instruction	Description
`L1 Allow`	Allow L1 to Design in both modes

Instruction	Description
`L1 GraftDesign Allow`	Allow L1 to GraftDesign
`L1 SeqDesign Allow`	Allow L1 to SequenceDesign
`L1 Fix`	Disable L1 to Design in both modes
`L1 GraftDesign Fix`	Disable L1 to GraftDesign
`L1 SeqDesign Fix`	Disable L1 to SequenceDesign
`L1 Weights Z`	Weight a particular CDR when choosing which CDR to design. By default, all CDRs have an equal weight.

CDRSet Syntax (General)

Instruction	Description
`L1 CDRSet Length Max X`	Maximum length of CDR
`L1 CDRSet Length Min Y`	Minimum length of CDR
`L1 CDRSet Cluster_Cutoffs Z`	Limits the CDRSet to include only CDRs of clusters that have Z or more members. Used in conjunction with Sequence Design cutoffs to sample clusters where profile data is sufficient or simply common clusters of a particular species.
`L1 CDRSet ONLY_CURRENT_CLUSTER`	Limits the CDRSet to include only CDRs belonging to the identified cluster of the starting CDR. Overrides other options. Used for sequence and structure sampling within CDR clusters. Useful for redesign applications.
`L1 CDRSet Center_Clusters_Only`	Limits the CDRSet to include only CDRs that are cluster centers. Overrides other options. Used for broadly sampling CDR structure space. Useful to get an idea of what CDRs can fit well or for a first-pass run for de novo design.

CDRSet Syntax (Include and Excludes)

This set of options either includes only the items specified or it excludes specific items. The syntax is thus:

```
L1 CDRSet INCLUDE_ONLY option list of items
```

OR

```
L1 CDRSet EXCLUDE option list of items
```

The options for this type of CDRSet syntax are listed below

Option	Description
`PDBIDs`	Include only or leave out a specific set of PDBIds. This is very useful for benchmarking purposes.
`Clusters`	Include only or leave out a specific set of Clusters. Useful if the user already knows which clusters are able to interact with antigen, whether this is from previous runs of the program or via homologues. This is also useful for benchmarking.
`Germline`	Include only or leave out specific human/mouse germlines.
`Species`	Include only or leave out specific species. Very useful limiting possible immune reactions in the final designs. 2 Letter designation as used by IMGT. Hu and Mo for Human and Mouse.

Sequence Design Instructions

The Sequence Design Instructions, used with the keyword SeqDesign after the CDR specifier, is used to control the sequence design aspect of the algorithm. In addition to controlling which CDRs undergo sequence design, the Sequence Design options control which strategy to use when doing sequence design (primary strategy), and which strategy to use if the primary strategy cannot be used for that CDR (fallback strategy).

Currently, the fallback strategy is used if the primary strategy does not meet statistical requirements. For example,

using cluster-based profiles for sequence design (which is the default), it is imperative that the particular cluster has at least some number of sequences in the database for the strategy to be successful. By default, this cutoff is set at 10 and uses the command-line option, `-seq_design_stats_cutoff Z`. This means that if a cluster has less than 10 members and the primary strategy is to use cluster-based profiles, then the fallback strategy will be used. The default fallback strategy for all of the CDRs is the use of a set of conservative mutations with data coming from the BLOSUM62 matrix by default, with the set of mutations allowed consisting of those substitutions with positive or zero scores in the matrix. For example, if we have a D at some position for which we are using this fallback strategy, the set of allowed mutants would be N, Q, E, and S. Further, the set of conservative mutations can be controlled using the command-line option, `-cons_design_data_source` source, where other BLOSUM matrices can be specified. All BLOSUM matrices can be used for conservative design, where the numbers (40 vs. 62 vs. 80 etc.) indicate the sequence similarity cutoffs used to derive the BLOSUM matrices - with higher numbers being a more conservative set of mutations.

General Syntax

Option	Description
`L1 SeqDesign Strategy Z`	Set Primary Sequence Design Strategy
`L1 SeqDesign Fallback_Strategy Z`	Set Fallback Sequence Design Strategy

Strategy Types

Z	Strategy Use	Description
`Profiles`	Primary	Use cluster-based profiles based on sequence probabilities as described above
`Profile_Sets`	Primary	Randomly sample a full CDR sequence from the CDR cluster each time packing is done
`Profiles_Combined`	Primary	Randomly sample a full CDR sequence from the CDR cluster and sample from the cluster-based profiles each time packing is done. Used to increase variability. Helpful in conjunction with center cluster member CDRSet sampling.
`Conservative`	Both	Conservative design operation as described above
`Basic`	Both	Basic design - no profiles, just enabling all amino acids at those positions
`Disable`	Fallback	Turn off design for that CDR if primary strategy does not meet cutoffs.

Etc

Option	Description
`L1 SeqDesign DISALLOWED`	Disable specific amino acid sampling for this CDR (One or Three letter codes. Can be mixed) Ex. ARG C S ASN PRO

Minimization

Many minimization types are implemented; however, they each require a different amount of time to run. See the MinType descriptions below for a list of acceptable options and their use.

Option	Description
`L1 MinProtocol MinType Z`	Set the minimization type for this CDR
`L1 MinProtocol MinOther CDRX, CDRY, CDRZ`	Set other CDRs to minimize during the optimization/design of this particular CDR. A packing shell around the CDRs to be minimized within a set distance (default 6 Å) is created. Any CDRs or regions (framework/antigen) set to design within this shell will be designed. These regions will use all set sequence design options (profiles,

Option	Description
	conservative, etc.). This option is useful for creating CDR-CDR interactions for loop and antibody structural stability. By default, we reasonably minimize other CDRs during the optimization step. (More CDRs = Lower Speed)

Default Settings

These settings are overridden by your set instruction file/command-line options

```
#General
ALL FIX
DE FIX

ALL WEIGHTS 1.0

#Minimization - Options: min, relax, ds_relax cartesian, backrub, repack, none
ALL MinProtocol MINTYPE min

#Neighbors.  These are conservative values used in benchmarking.  Limit these to speed up runs.
L1 MinProtocol Min_Neighbors L2 L3
L2 MinProtocol Min_Neighbors L1
L3 MinProtocol Min_Neighbors L1 H3
H1 MinProtocol Min_Neighbors H2 H3
H2 MinProtocol Min_Neighbors H1
H3 MinProtocol Min_Neighbors L1 L3

#Length
ALL CDRSet LENGTH MAX 25
ALL CDRSet LENGTH MIN 1


#Profile Design
ALL SeqDesign STRATEGY PROFILES
ALL SeqDesign FALLBACK_STRATEGY CONSERVATIVE

##DE Loops can be designed.  They use conservative design by default since we have no profile data!
DE SeqDesign STRATEGY CONSERVATIVE
```

GraftDesign Sampling Algorithms

These change the way CDRs are sampled from the antibody design database. They can be specified using the -design_protocol flag.

-design_protocol	Description
even_cluster_mc	Evenly sample clusters during the GraftDesign stage by first choosing a cluster from all the clusters set to design for the chosen Primary CDR and then choosing a structure within that cluster. (**DEFAULT**)
even_length_cluster_mc	Evenly sample lengths and clusters during the GraftDesign stage by first choosing a length from the set of lengths for the chosen Primary CDR and the a cluster from the set of clusters, and then finally a structure within that cluster. Useful to broaden set of lengths sampled during the protocol.
gen_mc	Sample CDRs to GraftDesign according to their distribution in the database. This results in common clusters and lengths being sampled more frequently. However, these lengths/clusters may not be those regularly seen in nature vs

-design_protocol	Description
	regularly crystalized. AKA - they are biased towards crystals, however, they have more profile data associated with them.
deterministic_graft	Deterministic Graft is meant to try every CDR combination from the CDRSet (the set of clusters and structures). The outer loop is done deterministically for each CDR in a set. It is very useful for trying small numbers of combinations - such as all loop lengths >=12 for H2 or all CDRs of a particular cluster. Note that there is no outer monte carlo, so the final designs are the best found by the protocol, and each sampling is independent from the others. If you have too many structures in your CDRSet (such as all L1) and you try combos that are beyond a certain limit (AKA - they will never finish), you will error out. Once a Multi-Threaded Rosetta is working, trying all possibilities is certainly something that is more plausible. If you are interested in using something like this, please email the author.

Structure Optimization Types

These 'Mintypes' can be independently set for each CDR through the instruction file or generally set using the command line option `-mintype`. The default is `min`, as this has some optimization and does not take a very long time (and has been shown to be comparable to relax). Although we refer to 'design' we mean side-chain packing, with any residues/CDRs set to design as designing. For further information on the algorithm and strategies used for sequence design, please see the instruction file overview and the methods section of the paper.

Circular Harmonic Dihedral Constraints are added to each CDR according the cluster of the CDR or the starting dihedrals if this is a rare cluster that has no data. These ensure that minimization and design does not destroy the loop.

-mintype	Description
min	**Cycle of design->min->design->min**. Results in good structures, however not as good as relax in recovering native physical characteristics. Significantly faster. (**DEFAULT**)
cartmin	**Cycle of design->min->design->min** Cartesian Space Minimization. Automatically adds cart_bonded term if not present and turns off pro_close
relax	Flexible Backbone design using `RelaxedDesign`, which is neighbor-aware design during FastRelax where the packing shell to the designing CDRs is updated at every packing iteration. Results in lower energies and closer physical characteristics to native, but takes significantly longer. It is recommended to first run min and then relax mintype on the top resulting models. (Relax Protocol Paper)
dualspace_relax	Flexible Backbone design using 'RelaxedDesign' while optimizing both Dihedral and Cartesian space Dualspace Relax Protocol Paper
backrub	**Cycle of backrub->design**. Uses backrub to optimize the Backbone of the CDRs set to minimize. Use the `-add_backrub_pivots 11A 12A 12A:B` option to add additional sets of back rub pivots, such as to add flexibility to the antigen interface. Flexibility is extremely minimal, but in some cases may be useful. (Original Backrub Protocol Paper)

RosettaScripts and PyRosetta

The Full protocol that is the application is available to RosettaScripts as the AntibodyDesignProtocol. This just has a few extra options before and after design such as running fast relax and or snug dock.

The Configurable main Mover is available as the AntibodyDesignMover.

Individual components of RAbD can be used to create your own custom antibody modeling and design protocols in RosettaScripts (or PyRosetta).

Command-line Options

General Options

Option	Description
`-view`	Enable viewing of the antibody design run. Must have built using extras=graphics and run with antibody_designer.graphics executable (**Default=False**)
`-cdr_instructions`	Path to CDR Instruction File
`-antibody_database`	Path to the current Antibody Database, updated weekly. Download from http://dunbrack2.fccc.edu/PyIgClassify/ (**Default=/sampling/antibodies/antibody_database_rosetta_design.db**)
`-paper_ab_db`	Force the use the Antibody Database with data from the North clustering paper. This is included in Rosetta. If a newer antibody database is not found, we will use this. The full ab db is available at Through PyIgClassify (**Default = false**)

Basic settings for an easy-to-setup run

Option	Description
`-seq_design_cdrs`	Enable these CDRs for Sequence-Design. (Can be set here or in the instructions file. Overrides any set in instructions file if given) Ex -seq_design_cdrs L1 L2 L3 h1
`-graft_design_cdrs`	Enable these CDRs for Graft-Design. (Can be set here or in the instructions file. Overrides any set in instructions file if given) Ex -graft_design_cdrs L1 L2 L3 h1
`-primary_cdrs`	Manually set the CDRs which can be chosen in the outer cycle. Normally, we pick any that are sequence-designing.
`-mintype`	The default mintype for all CDRs. Individual CDRs may be set via the instructions file (*Options = min, cartmin, relax, backrub, pack, dualspace_relax, cen_relax, none*) (**Default=min**)
`-disallow_aa`	Disallow certain amino acids while sequence-designing (could still be in the graft-designed sequence). Useful for optimizing without, for example, cysteines and prolines. Applies to all sequence design profiles and residues from any region (cdrs, framework, antigen). You can control this per-cdr (or only for the CDRs) through the CDR-instruction file. A resfile is also accepted if you wish to limit specific positions directly. (Three or One letter codes)
`-top_designs`	Number of top designs to keep (ensemble). These will be written to a PDB and each move onto the next step in the protocol. (**Default=1**)
`-do_dock`	Run a short lowres + highres docking step in the inner cycles. (dock/min). Recommended 2 inner cycles for better coverage. (dock/min/dock/min). Inner/Outer loops for highres are hard coded, while low-res can be changed through regular low_res options. If sequence design is enabled, will design regions/CDRs set during the high-res dock (**Default=false**)

Energy Optimization settings (opt-dG)

Option	Description
`-mc_optimize_dG`	Optimize the dG during MonteCarlo. It is not possible to do this within overall scoring, but where possible, do this during MC calls. This option does not globally-use the MonteCarloInterface object, but is protocol-specific. This is due to needing to know the interface it will be used on. dG is measured by the InterfaceAnalyzerMover. (**Default=false**)
`-mc_interface_weight`	Weight of interface score if using MonteCarloInterface with a particular protocol. (**Default=1.0**)
`-mc_total_weight`	Weight of total score if using MonteCarloInterface with a particular protocol. (**Default=0.0**)

Protocol Rounds

Option	Description
`-outer_cycle_rounds`	Rounds for outer loop of the protocol (not for deterministic_graft). Each round chooses a CDR and designs. One run of 100 cycles with relax takes about 12 hours. If you decrease this number, you will decrease your run time significantly, but your final decoys will be higher energy. Make sure to increase the total number of output structures (nstruct) if you use lower than this number. Typically about 500 - 1000 nstruct is more than sufficient. Full DeNovo design will require significantly more rounds and nstruct. If you are docking, runs take about 30 percent longer. (**Default=25**)
`-inner_cycle_rounds`	Number of times to run the inner minimization protocol after each graft. Higher (2-3) rounds recommended for pack/min/backrub mintypes or if including dock in the protocol. (**Default = 1**)
`-dock_cycle_rounds`	Number of rounds for any docking. If you are seeing badly docked structures, increase this value. (**Default=1**)

Distance Detection

Option	Description
`-interface_dis`	Interface distance cutoff. Used for repacking of interface, epitope detection, etc. (**Default=8.0**)
`-neighbor_dis`	Neighbor distance cutoff. Used for repacking after graft, minimization, etc. (**Default=6.0**)

Paratope + Epitope

Option	Description
`-paratope`	Use these CDRs as the paratope. Default is all of them. Currently only used for SiteConstraints. Note that these site constraints are only scored docking unless *-enable_full_protocol_atom_pair_cst* is set (Ex -paratope L1 h1)
`-epitope`	Use these residues as the antigen epitope. Default is to auto-identify them within the set interface distance at protocol start if epitope constraints are enabled. Currently only used for constraints. PDB Numbering. Optional insertion code. Example: 1A 1B 1B:A. Note that these site constraints are only used during docking unless -enable_full_protocol_atom_pair_cst is set.
`-use_epitope_constraints`	Enable use of epitope constraints to add SiteConstraints between the epitope and paratope. Note that paratope constraints are always used. Note that these site constraints are only used during docking unless -global_atom_pair_cst_scoring is set.(**Default = false**)

Regional Sequence Design

Option	Description
`-design_antigen`	Design antigen residues during sequence design. Intelligently. Typically, only the neighbor antigen residues of designing cdrs or interfaces will be co-designed. Useful for specific applications.(**Default = false**)
`-design_framework`	Design framework residues during sequence design. Typically done with only neighbor residues of designing CDRs or during interface minimization. (**Default = false**)
`-conservative_framework_design`	If designing Framework positions, use conservative mutations instead of all of them. (**Default=true**)

Seq Design Control

Option	Description
`-resfile`	Use a resfile to further limit which residues are packed/designed, and can further limit residue types for design.
`-design_H3_stem`	Enable design of the first 2 and last 3 residues of the H3 loop if sequence designing H3. These residues play a role in the extended vs kinked H3 conformation. Designing these residues may negatively effect the overall H3 structure by potentially switching a kinked loop to an extended and vice versa. Rosetta may get it right. But it is off by default to err on the cautious side of design. Sequence designing H3 may be already risky. (**Default=false**)
`-design_proline`	Enable proline design. Profiles for proline are very good, but designing them is a bit risky. Enable this if you are feeling daring. (**Default=false**)
`-sample_zero_probs_at`	Value for probabilstic design. Probability that a normally zero prob will be chosen as a potential residue each time packer task is called. Increase to increase variablility of positions. (**Default=0**)

Profile Stats

Option	Description
`-seq_design_stats_cutoff`	Value for probabilistic -> conservative sequence design switch. If number of total sequences used for probabilistic design for a particular cdr cluster being designed is less than this value, conservative design will occur. More data = better predictability. (**Default=10**)
`-seq_design_profile_samples`	If designing using profiles, this is the number of times the profile is sampled each time packing done. Increase this number to increase variability of designs - especially if not using relax as the mintype. (**Default=1**)

Constraint Control

Option	Description
`-dihedral_cst_weight`	Weight to use for CDR CircularHarmonic cluster-based or general constraints that are automatically added to each structure and updated after each graft. Set to zero if you don't want to use these constraints. Note that they are not used for the backrub mintype. Overrides weight/patch settings.(**Default = .3**)
`-atom_pair_cst_weight`	Weight to use for Epitope/Paratope SiteConstraints. Paratope Side contraints are always used. Set to zero to completely abbrogate these constraints. Overrides weight/patch settings.'Real' (**Default = 0.01**)
`-global_dihedral_cst_scoring`	Use the dihedral cst score throughout the protocol, including final scoring of the poses instead of just during minimization step (**Default = false**)
`-global_atom_pair_cst_scoring`	Use the atom pair cst score throughout the protocol, including final scoring of the poses instead of just during docking. Typically, the scoreterm is set to zero for scorefxns other than docking to decrease bias via loop lengths, relax, etc. It may indeed help to target a particular epitope quicker during monte carlo design if epitope constraints are in use, as well for filtering final models on score towards a particular epitope if docking. (**Default = true**)

Fine Control

Option	Description
-idealize_graft_cdrs	Idealize the CDR before grafting. May help or hinder. (**Default = false**)
-add_backrub_pivots	'Additional backrub pivot residues if running backrub as the MinType. PDB Numbering. Optional insertion code. Example: 1A 1B 1B:A. Can also specify ranges: 1A-10:A. Note no spaces in the range.
-inner_kt	KT used in the inner min monte carlo after each graft. (**default = 1.0**),
-outer_kt	KT used for the outer graft Monte Carlo. Each graft step will use this value (**Default = 1.0**),

Outlier Control

Option	Description
-use_outliers	Include outlier data for GraftDesign, profile-based sequence design stats, and cluster-based dihedral constraints. Outliers are defined as having a dihedral distance of > 40 degrees and an RMSD of >1.5 A to the cluster center. Use to increase sampling of small or rare clusters. (**Default=false**)
-use_H3_graft_outliers	Include outliers when grafting H3. H3 does not cluster well, so most structures have high dihedral distance and RMSD to the cluster center. Due to this, cluster-based dihedral constraints for H3 are not used. Sequence profiles can be used for clusters, but not usually. (**Default = true**)
-use_only_H3_kinked	Remove any non-kinked CDRs from the CDRSet if grafting H3. For now, the match is based on the ramachandran area of the last two residues of the H3. Kinked in this case is defined as having AB or DB regions at the end. Will be improved for detection (**Default = false**)

Protocol Steps

Option	Description
-design_protocol	Set the main protocol to use. Note that deterministic is currently only available for the grafting of one CDR. (Options = gen_mc, even_cluster_mc, even_length_cluster_mc, deterministic_graft) (**Default=even_cluster_mc**)
-run_snugdock	Run snugdock on each ensemble after designing. (**Default=false**)
-run_relax	Run Dualspace Relax on each ensemble after designing (after snugdock if run). Also output pre-relaxed structures (**Default = false**)
-run_interface_analyzer	Run the Interface Analyzer and add the information to the resulting score function for each top design output. (**Default = true**)

Memory management / CDRSet caching

Option	Description
-high_mem_mode	If false, we load the CDRSet (CDRs loaded from the database that could be grafted) on-the-fly for a CDR if it has more than 50 graft-design members. If true, then we cache the CDRSet before the start of the protocol. Typically, this will really only to come into play when designing all CDRs. For de-novo design of 5/6 CDRs, without limiting the CDRSet in the instructions file, you will need 3-4 gb per process for this option. (**default = false**)
-cdr_set_cache_limit	If high_mem_mode is false, this is the limit of CDRSet cacheing we do before we begin load them on-the-fly instead. If high_mem_mode is true, then we ignore this setting. If you have extremely low memory per-process, lower this number (**default = 300**)

Benchmarking

Option	Description
`-random_start`	Start graft design (currently) with a new set of CDRs from the CDRSets as to not bias the run with native CDRs. (**Default=false**)
`-remove_antigen`	Remove the antigen from the pose before doing any design on it (**Default = false**)
`-add_graft_log_to_pdb`	Add the full graft log to the output pose. Must also pass -pdb_comments option. (**Default = 'true'**)