Shc Inhibitors Docking Protocol

February 10, 2021

1 Software

1.1 Rossetta

The Rosetta suite was used to preform the actual ligand docking simulations. Rosetta was downloaded and compiled to a remote cluster where all simulations were ran.

I often accessed the following resources when determining how to run Rosetta for ligand docking.

1. Rosetta ligand docking demo
2. Rosetta Ligand docking with flexible XML protocols
3. Rosetta Documentation

1.2 BioChemicalLibrary (BCL)

BCL was used for generating libraries of conformers (different possible conformations of a ligand) from one specific structure. These conformer libraries would then be given to Rosetta in order to simulate ligand flexibility.

1.3 Open Bable

Open Babel suite was used for one-off file conversions in cases where `sdf` files needed to be converted to `pdb` or similar operations.

1.4 RDBC

Rosetta ligand docking batch job submission control and organizer is a Python command line program I created to help me submit organize and analysize large number of Rosetta docking simulations to the remote cluster's workload manager, SLURM.

2 General workflow

This section describes the general procedure for running docking simulations to access the binding of a specific ligand to a specific protein target.

2.1 Prepare protein structure

First, a ligand free protein structure was repacked using the `ligand_rpkmin.static.linuxgccrelease` program of the Rosetta suite. Repacking was repeated for 100 structures and the lowest energy structure is selected as the docking target.

2.2 Prepare ligand

First, a file that describes the ligand needs to be aquired. For `RTX ...` drugs these was just a matter of using the `sdf` files. If the simulation involved docking a ligand / peptide from an existing co-crystal structure Pymol was used to create a `pdb` file containing only the ligand / peptide. Open Bable was then used to convert the `pdb` file to `sdf` format.

Next, the ligand conformer library and Rosetta `params` files was generated for the to-be-docked ligand. This simulates ligand flexibility during docking. This was usually completed with this Python script which wraps the BCL `molecule:ConformerGenerator` program and the Rosetta `molfile_to_params.py` located in `main/source/scripts/python/public/` of a standard Rosetta installation.

2.3 Prepare RDBC files

I almost always used RDBC to actually run the docking simulations on the remote cluster. RDBC works by using templates of files that would be required for individual jobs and filling them out based on the command line arguments to run many jobs. One of the most important is the Rosetta XML docking protocol which determines exactly what Rosetta does during the simulations. For random docking experiments (where the ligand is positioned at a random position around the protein before docking) I used the random_docker_template.xml.

2.4 Submit jobs with an RDBC command

Once all nessecary files are created the Rosetta docking simulations where submitted to SLURM using RDBC. Below is an example command I used to for randomly docking the NPEYp peptide to the 1OY2 shc structure.

```
python3 ~/software/RDBC/rh.py -l ~/jobs/dock_random_NPEYp/ligand \
-p ~/jobs/dock_random_NPEYp/protein/Shc1-PTB_1OY2_0061.pdb \
-o ~/jobs/dock_random_NPEYp/results \
-e ~/software/rosetta_bin_linux_2020.08.61146_bundle/main/source/bin/rosetta_scripts.static.li
-i 2000 \
-op ~/software/RDBC/handler/xml_templates/random_docker.xml \
-b ~/jobs/dock_random_NPEYp/templates/NPEYp.sbatch \
-mi 10
```

- -l: Location of my ligand preparation files produced during the *prepare ligand* step. This included
  - NPEYp_conformers.pdb: Conformer structures generated with BCL.
  - NPEYp.params: Rosetta params file generated by molfile_to_params.py
  - NPEYp.pdb: PDB structure of the NPEYp ligand converted from a provided sdf file.
- -p: Path to target protein. Selected during the *prepare protein structure* step.
- -o: Output path. Where I want the results of simulations to be written to.
- -e: Path to Rosetta scripts exe.
- -i: Number of individual simulations each job should run. This is equivlanet to the number of poses Rosetta will produce.
- -op: Path to Rosetta XML protocol file. In this case I used one designed for random docking.
- -b: Path to template `batch` file. This is filled out for each individual job in order to submit many smaller jobs instead of one larger one. This avoids issues if the reasrouces you can use

for one job are limited (as was my case).
- -mi: Tells RDBC to submit 10 copies of this job, which allows for simulating more poses when resources for individual jobs are constrained.

Then we just wait for our jobs to complete.

2.5 Aggregate run copies

For random docking that submits multible copies of the same job, I found it easer to aggregate the results into one large file that is easier to work with for plotting in programs like R.

If RDBC was used to submit such a job, it can also be used for aggregating the results using the `-mai` argument. If my jobs created by the example command had just finished they could be aggregated into one large results tab seperated file called `NPEYp.agg.tsv` with the command below.

```
RDBC/rh.py -o /home/ethollem/jobs/dock_random_NPEYp/results -mia NPEYp.agg.tsv
```