Tutorial 7: Structure-based design simulation of selective small molecule dopamine receptor antagonists using alchemistry or single-shot reactions

Author: Benjamin P. Brown (benjamin.p.brown17@gmail.com)

Date: 01-2022

Background

In Tutorial 4 we used ligand-based multi-component reaction design to create a library of dopamine receptor D4 (DRD4) antagonists. We then used ligand-based neural network quantitative structure activity relationship (QSAR) models to predict their activities and visualized a few of the molecules with the best predicted activity and selectivity over other dopamine receptors.

Frequently in small molecule drug discovery, you are looking for consensus between modeling approaches to determine what you will invest time/money/effort in synthesizing/purchasing/screening. The most straightforward approach would be to dock a subset of the molecules from our screen (along with suitable positive and negative controls) and rank their predicted binding affinities for each dopamine receptor using either the RosettaLigand docking score, a molecular dynamics (MD)-based free energy perturb (FEP) / thermodynamic integration (TI) approach, and/or a structure-based machine learning (ML) scoring system. Then we pursue the compounds that are predicted to be the best in all of the different simulations.

We are not going to do that. Here, we will pretend that we do not have enough data to build a robust QSAR model. In this scenario, we do have reliable methods to generate structural models of the receptors and we know the location of the binding pocket, making structure-based design an attractive approach.

We are also not going to use the multi-component reaction design strategy. We could use it – there is a "React" mutate derived from `FragmentMutateInterface` that we can define in the `object_data_label` of a `BCLFragmentMutateMover` (just make sure you set the ligand_based option to false) – but we will not for a couple of reasons.

One of the issues is that reactions with more than two reagents are generally assembled from very small fragments, which means we are starting from very little protein-ligand interaction pose data. In the best-case scenario, the chosen starting fragments have a well-defined position in the pocket and we can dock the remainder of the ligand where the only degrees of freedom are conformational – this greatly reduces the challenge of the docking problem, especially if we can also align the new molecule to a suitable reference molecule first. In the worst-case scenario, we're redocking every reaction product.

Another issue is that I'm not a medicinal chemist and cannot actually perform a split-Ugi-4-component reaction. I'll assume since you are doing this tutorial that you are also avoiding time in the wetlab. It is unlikely that we will find a company to perform a reaction like this one, and if they will you may also have to find and purchase the reagents. Luckily for us, today there are companies that will perform on-demand synthesis of potentially billions of molecules using a large collection of building blocks and a few simple one-shot two component reactions. Thus, if we restrict our chemical space to that which is accessible with these reagents and reactions, we can order the resultant compounds in much the same way we order compounds from typical screening libraries. Other members of the Meiler Lab (looking at you Rocco, Tracy, Paul) are actively doing exciting science aimed at designing algorithms to help navigate this vast chemical space, as are other research groups.

In this tutorial, we will use a structure-based design approach to build new molecules and rank them based on both their predicted affinity for DRD4 as well as their predicted selectivity over DRD2, DRD3, and DRD5. Broadly, we will split the tutorial into two sections. The first section will use alchemical mutates (e.g., `Halogenate`, `Alchemy`, `ExtendWithLinker`, `RingSwap`, etc.) to build structure-activity relationship (SAR) profiles on existing scaffolds. The goal with these simulations is to probe a scaffold to better understand the determinants of selectivity. The second section will demonstrate how to use the `AddMedChem` mutate to perform Enamine-style reactions.

Part 1: Preparing a small molecule scaffold for design

Our input scaffold can be prepared just as in Tutorial 5. Use files Tutorial_7/input/ligands/LIG_0001.fa.pdb instead. We will not walk through this step explicitly in this tutorial; please feel free to look back at Tutorial 5 Part 1 or use the prepared files in Tutorial_7/input/ligands/.

Recall that the Rosetta atom indices that we will reference in our XML scripts will differ from the atom indices in Tutorial_7/input/ligands/LIG_0001.fa.pdb. The "rosetta_indexed_LIG.pdb" file was generated from restype_converter using LIG.fa.params as input and will correspond to the atom indices referenced in this tutorial.

Congratulations! Move on to Part 2.

Part 2: *In silico* SAR profile creation

For this example, we will start with a difluoropiperadine DRD4 antagonist scaffold from Jeffries et al. 2016[1]. Protein-ligand docking is beyond the scope of this tutorial. If you are unfamiliar with docking or want a refresher on docking in Rosetta, I recommend you look at the Rosetta tutorials linked on the Meiler Lab website, read through Gordon & Meiler 2012 (https://pubmed.ncbi.nlm.nih.gov/22183535/), and/or check out the protocol capture in Smith & Meiler 2020[2].

In preparation for this tutorial, I have docked one of the difluoropiperadine scaffolds into DRD4.

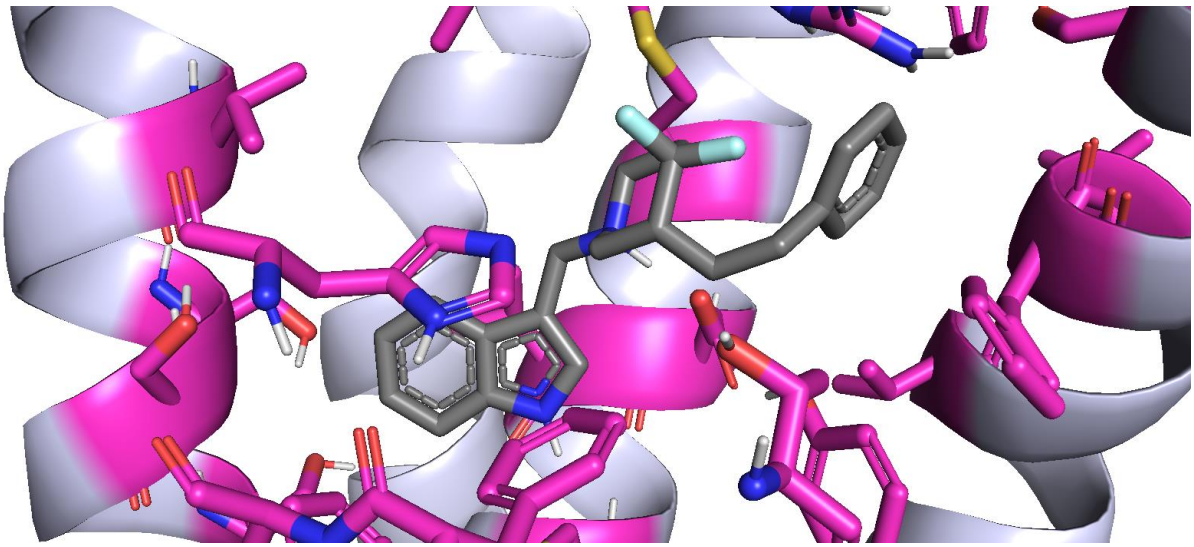

Figure 1. Docked pose of our difluoropiperadine scaffold from Jeffries et al. 2016[1].

There is another variant of the scaffold in which the indole ring is replaced by a benzene. I specifically chose the indole variant because it has less symmetry. Molecules with a high degree of symmetry are more likely to "flip" binding modes when substituents are added to either side, making it challenging to infer SAR. Moreover, recent work by Zhou et al. 2019[3] suggests a potential mode of binding for molecules with approximately this topology, which allowed me to initialize the docking simulation with a pose created with BCL::MolAlign and more rapidly converge on a likely binding mode.

On that note, let's begin design.

I have nominally chosen four design strategies for illustrative purposes and assigned them to a `RandomMover` with equal probability:

```
<RandomMover name="bcl_design"
movers="bcl_swap_core,bcl_swap_east,bcl_swap_west,bcl_opti"
weights="0.25,0.25,0.25,0.25"
/>
```

Let's look more deeply at a couple of the mutates. Generally, you will find these mutates mirror much of what we saw in the previous two tutorials. The salient point for this particular example is related to the different RingSwap strategies we use for different parts of the molecule.
Compare the RingSwap mutate used in `bcl_swap_east` with `bcl_swap_core`:

East:

```
<BCLFragmentMutateMover name="ringswap_conservative_aro_east"
ligand_chain="X"
object_data_label="RingSwap(
ring_library=%%rings%%/alternative_rings.east.sdf.gz,
druglikeness_type=IsConstitutionDruglike,
conservative=1,restricted=1,
atom_comparison=ElementType,
bond_comparison=BondOrderOrAromaticWithRingness,
mutable_atoms=0 1 2 3 4 5,
ring_initiation_probability=0.0)"
/>
```

Core:

```
<BCLFragmentMutateMover name="ringswap_conservative_core"
ligand_chain="X"
object_data_label="RingSwap(
ring_library=%%rings%%/alternative_rings.core.sdf.gz,
ring_initiation_probability=0.0,
restricted=0,conservative=1,
refine_alignment=1,
atom_comparison=CouldHaveSubstituents,
bond_comparison=BondOrderOrAromaticWithRingness,
druglikeness_type=IsConstitutionDruglike,
mutable_atoms=8 9 10 11 23)"
/>
```

The strategy for the east ring swap is consistent with what we have seen previously: We specify a ring library to draw from (`ring_library`), reduce the probability of growing or collapsing a ring to zero (`ring_initiation_probability`), restrict ring size changes to be comparable to the current ring (restricted=1), and request a substructure-based alignment with the current ring (`conservative`=1) prior to attachment (because attaching the ring at any available random ring atom may perturb the structure more liberally than desired). We do the ring substructure alignment at the resolution of `ElementType` for atoms and `BondOrderOrAromaticWithRingness` for bonds.

The strategy for the core swap is a bit different for three reasons: (1) changes to the core ring conformation propagate outward such that even small changes to the core (e.g., dropping from a 6-membered ring to a 5-membered ring) can notably shift the protein-ligand interaction pose; (2) the core ring and its potential alternatives are not aromatic and generally contain at least one stereocenter, so there are more configurational degrees of freedom associated with sampling this ring than we have worried about so far; and (3) the core ring is heavily substituted, and we if we cannot match the substitution pattern then we must lose either our fluorine decorations or the entirety of one or more of the attached rings.

So how do we address these challenges? (In-development feature)

First, maintain the `conservative`=true option. This will perform an initial alignment of the new ring to the old ring to bias the attachment site. Second, however, **we change the atom comparison type to `CouldHaveSubstituents`**, which compares atoms based on their ability to have substitutions. This will align atoms to maximize attachments that mimic the attachments of the original core ring.

Third, we enable the `refine_alignment` flag. This will perform a quick flexible alignment of the new molecule to the starting structure using substructure superimpositions of ligand conformers scored with the BCL::MolAlign method[4] where the comparisons are done at the `ElementType` and `BondOrderOrAromaticWithRingness` levels (I'll make this customizable some point soon). Importantly, **`refine_alignment` performs the alignment across the entire molecule and not just the ring being substituted**. This is important because it contrasts with the role and alignment resolution of the `conservative` flag. It means that the starting alignment of the core ring will be optimized to place substituents, but the final refinement will be done to match the original geometry of the whole molecule.

This may seem abstract. Let's run a few examples. Focus your XML to modify just the core by changing the `RandomMover` probabilities.

```
<RandomMover name="bcl_design"
movers="bcl_swap_core,bcl_swap_east,bcl_swap_west,bcl_opti"
weights="1.0,0.0,0.0,0.0"
/>
```

Run the protocol with the original `RingSwap` settings.

```
bash scripts/AffinityDesignD4.sh scripts/AffinityDesignD4.xml
input/receptors/DRD4.LIG.receptor.pdb input/ligands/LIG_0001.fa.pdb
input/ligands/LIG.fa.params output/AffinityDesignD4/RS_CHS_
```

While that is running, go into the XML and change the `atom_comparison` flag of the core `RingSwap` mutate to `ElementType`:

```
<BCLFragmentMutateMover name="ringswap_conservative_core"
ligand_chain="X"
object_data_label="RingSwap(
ring_library=%%rings%%/alternative_rings.core.sdf.gz,
ring_initiation_probability=0.0,
restricted=0,conservative=1,
refine_alignment=1,
atom_comparison=ElementType,
bond_comparison=BondOrderOrAromaticWithRingness,
druglikeness_type=IsConstitutionDruglike,
mutable_atoms=8 9 10 11 23)"
/>
```

Now run the protocol again.

```
bash scripts/AffinityDesignD4.sh scripts/AffinityDesignD4.xml
input/receptors/DRD4.LIG.receptor.pdb input/ligands/LIG_0001.fa.pdb
input/ligands/LIG.fa.params output/AffinityDesignD4/RS_ET_
```

And one more time, change the `atom_comparison` flag to `AtomType` and run the protocol again

```
bash scripts/AffinityDesignD4.sh scripts/AffinityDesignD4.xml
input/receptors/DRD4.LIG.receptor.pdb input/ligands/LIG_0001.fa.pdb
input/ligands/LIG.fa.params output/AffinityDesignD4/RS_AT_
```

When these finish, compare the results. What do you notice about the ensembles you obtain from each? Do the results make sense? What happens if you turn off the `conservative` flag? Food for thought.

Part 3: Designing for DRD4 selectivity against DRD2, DRD3, and DRD5 with alchemical mutates

In this section we will focus less on different mutate options and more on a new way of scoring the mutates. Specifically, this section will demonstrate how selectivity modeling can be performed with RosettaScripts during small molecule drug design.

Before we begin, there are a few observations and assumptions that require mentioning. First, the members of the dopamine receptor family are homologous to varying degrees. Crystallographic evidence has suggested that DRD2, DRD3, and DRD4 adopt similar folds.

Second, we assume based on the homology of the receptors that the primary differences in orthosteric antagonist affinity will arise from amino acid sequence differences at the protein-ligand interface. This assumption is not strictly true because differences in dynamics, energetic barriers to induced-fit deformation of the pocket, long-range electrostatic, and other more complex features may contribute to the differences. Nevertheless, this is a useful assumption because it simplifies the modeling problem to something tractable for larger throughput design.

Third, we assume based on the homology that the DRD4 backbone conformation is compatible with DRD2, DRD3, and DRD5, such that we can alternate between the different proteins using fixed-backbone design. This assumption exists primarily for demonstrative purposes and may be more or less applicable between different proteins. A more rigorous way to go about this approach would be to do the fixed-backbone design on the backbones for each receptor and report the final interaction energy for each design on a single sequence as the average across all the backbones. Alternatively, we could simply generate models of each receptor ahead of time, superimpose their orthosteric binding pockets, and flip between them using the Transform mover, or score them each separately with the ligand in PyRosetta.

Okay, let's begin by preparing several score filters. We will evaluate our designs based on the protein-ligand interaction energy, the difference in interaction energies between off-target receptors and DRD4, and the Boltzmann-weighted ratio between the DRD4 interaction energy and the sum of all Boltzmann-weighted interaction energies

For DRD4, the first two metrics look like this:

```
<LigInterfaceEnergy name="d4_ifscore" scorefxn="t14" include_cstE="0"
energy_cutoff="0.0"/>
<IfThenFilter name="d4_ifscore_bounded" threshold="0">
<IF testfilter="d4_ifscore" valuefilter="d4_ifscore"/>
<ELSE value="0.0"/>
</IfThenFilter>
<ReadPoseExtraScoreFilter name="read_d4_ifx" term_name="d4_ifx"
threshold="0.0"/>
```

This can then be replicated for each of the other receptors. The differences are written with a `CalculatorFilter` filter

```
<CalculatorFilter name="d4-d2_ifx" threshold="10.0" equation="E1-E2" >
<Var name="E1" filter="read_d4_ifx" />
<Var name="E2" filter="read_d2_ifx" />
</CalculatorFilter>
```

and replicated for all differences with respect to DRD4 interaction energy.

The Boltzmann-weighted ratios can be written as:

```
<CalculatorFilter name="boltz_selectivity" threshold="10.0"
equation=
"t1=exp(-E1/kT);
t2=exp(-E2/kT);
t3=exp(-E3/kT);
t4=exp(-E4/kT);
t1/( t1 + t2 + t3 + t4)">
<Var name="E1" filter="read_d4_ifx" />
<Var name="E2" filter="read_d2_ifx" />
<Var name="E3" filter="read_d3_ifx" />
<Var name="E4" filter="read_d5_ifx" />
     <Var name="kT" value="0.593" />
</CalculatorFilter>
```

where kT is our molar gas constant multiplied by temperature in units of kcal/mol. Note that Talaris2014 does not scale to kcal/mol, and this is not necessarily an optimal value.

Because we are always alternating between four fixed sequences, we can perform the fixed-backbone sequence design using predefined sequence strings with the `SimpleThreadingMover`.

```
<SimpleThreadingMover name="design_d4" pack_neighbors="true"
          neighbor_dis="4.0" start_position="1"
thread_sequence="%%d4_seq%%"
          scorefxn="t14" skip_unknown_mutant="false" pack_rounds="5"
/>
<SimpleThreadingMover name="design_d2" pack_neighbors="true"
          neighbor_dis="4.0" start_position="1"
thread_sequence="%%d2_seq%%"
          scorefxn="t14" skip_unknown_mutant="false" pack_rounds="5"
/>
<SimpleThreadingMover name="design_d3" pack_neighbors="true"
          neighbor_dis="4.0" start_position="1"
thread_sequence="%%d3_seq%%"
          scorefxn="t14" skip_unknown_mutant="false" pack_rounds="5"
/>
<SimpleThreadingMover name="design_d5" pack_neighbors="true"
          neighbor_dis="4.0" start_position="1"
thread_sequence="%%d5_seq%%"
          scorefxn="t14" skip_unknown_mutant="false" pack_rounds="5"
/>
```

We can perform a constrained relax on the new sequence with the inhibitor, and afterward we score the interaction energy. Importantly, however, we need to be able to store the interaction energy for later. Fortunately, we can do this by using the `FilterReportAsPoseExtraScoresMover` mover.

```
<FilterReportAsPoseExtraScoresMover name="save_d4_ifx"
report_as="d4_ifx" filter_name="d4_ifscore_bounded"/>
```

The filter that it applies and saves is one of our previously defined filters for the interface energy. We will be able to access the return value of the filter saved by this mover using `ReadPoseExtraScoreFilter`. Let's also save the pose so that we can write it out at the end of the protocol.

```
<FilterReportAsPoseExtraScoresMover name="save_d4_ifx"
report_as="d4_ifx" filter_name="d4_ifscore_bounded"/>
<SavePoseMover name="save_d4_wt" restore_pose="0"
reference_name="d4_wt" />
```

We can define these two movers for all of our off-target receptors as well, and then we can just cycle between them.

```
<ParsedProtocol name="score_off_targets" mode="sequence">
<Add mover_name="design_d2"/>
<Add mover_name="relax_cycle"/>
<Add mover_name="save_d2_ifx"/>
<Add mover_name="save_d4_d2like"/>
<Add mover_name="design_d3"/>
     <Add mover_name="relax_cycle"/>
     <Add mover_name="save_d3_ifx"/>
     <Add mover_name="save_d4_d3like"/>
    <Add mover_name="design_d5"/>
     <Add mover_name="relax_cycle"/>
     <Add mover_name="save_d5_ifx"/>
     <Add mover_name="save_d4_d5like"/>
</ParsedProtocol>
```

We'll define a protocol to recover the saved poses and write them to disk using the `PDBTrajectoryRecorder` mover

```
<ParsedProtocol name="retrieve_final_poses" mode="sequence">
<Add mover_name="load_d4_wt"/>
<Add mover_name="write_pdb_d4"/>
<Add mover_name="load_d4_d2like"/>
<Add mover_name="write_pdb_d2"/>
<Add mover_name="load_d4_d3like"/>
<Add mover_name="write_pdb_d3"/>
<Add mover_name="load_d4_d5like"/>
<Add mover_name="write_pdb_d5"/>
</ParsedProtocol>
```

where `write_pdb_d4` is

```
<PDBTrajectoryRecorder name="write_pdb_d4" stride="1"
filename="%%prefix%%D4.pdb" cumulate_replicas="0" cumulate_jobs="0"/>
```

Finally, we'll tie it all together.

```
        <PROTOCOLS>
                <Add mover_name="min_cycle_soft_initial"/>
                <Add mover_name="bcl_design"/>
                Add mover_name="score_only"/>
                <Add filter_name="d4-d2_ifx"/>
                <Add filter_name="d4-d3_ifx"/>
                <Add filter_name="d4-d5_ifx"/>
                <Add filter_name="boltz_selectivity"/>
                <Add mover_name="retrieve_final_poses"/>
     </PROTOCOLS>
```

And that is the bulk of it. Then we just define a `PROTOCOL` block at the very end to minimize our starting structure, perform design on our small molecule, score that molecule design against each of our proteins, compute our final metrics, and dump our poses:

```
<PROTOCOLS>
<Add mover_name="min_cycle_soft_initial"/>
<Add mover_name="bcl_design"/>
      <Add filter_name="d4-d2_ifx"/>
<Add filter_name="d4-d3_ifx"/>
      <Add filter_name="d4-d5_ifx"/>
<Add filter_name="boltz_selectivity"/>
<Add mover_name="retrieve_final_poses"/>
</PROTOCOLS>
```

So! Now feel free to let it fly. Let's build 20 models (you can also view pre-generated 20 models in the output).

```
bash scripts/SelectivityDesignD4.sh scripts/SelectivityDesignD4.xml
input/receptors/DRD4.LIG.receptor.pdb input/ligands/LIG_0001.fa.pdb
input/ligands/LIG.fa.params
input/sequences_for_threading/d4_native.seq
input/sequences_for_threading/d4_d2like.seq
input/sequences_for_threading/d4_d3like.seq
input/sequences_for_threading/d4_d5like.seq
output/SelectivityDesign/SELECT_
```

Once the models have finished, take a look at them. Are there any reoccurring features? Did you produce anything predicted to be more selective than your starting scaffold? To find out, disable the bcl_design component of the protocol and enable score_only:

Part 3 Extension (Optional): Checking a QSAR hit with structure-based modeling

Earlier I mentioned that you often want to have consensus between ligand- and structure-based methods before ordering molecules. This is a recent example from a virtual screen that we did. Our ligand-based QSAR model predicted that molecule "469.sdf" to be selective for DRD4 over DRD2.

We were interested in it because it was structurally very close to a previously identified orthosteric antagonist that was selected for DRD4 over DRD2 (called "002.sdf" in the provided inputs)[5].

They differ only in that the leftmost ring is either a dimethylbenzene or a napthalene. The latter is slightly larger. Unfortunately, our experimental characterization of the compound indicated that it is *not* selective. Could we have avoided this issue if we had performed structural modeling? Let's find out.

To preserve as much coordinate information as possible between the two antagonists and minimize the noise in the relative binding energy estimate, we will use the BCL FragmentMutateInterface to create the napthalene derivative from the docked dimethylbenzene compound. I cheated a bit to do this by looking at the intermediate structures to lookup the atom indices as we added new atoms, but the goal here was to explicitly build a single molecule, so it's fine.

```
cd Tutorial_7/input/ligands
```

```
bcl.exe molecule:Mutate -implementation \
"Alchemy(mutable_atoms=6,allowed_elements=C,restrict_to_bonded_h=1)" \
"Alchemy(mutable_atoms=5,allowed_elements=C,restrict_to_bonded_h=1)" \
"AddBond(mutable_atoms=26,paired_atoms=27,bond_type=AromaticDoubleBond
)" \
"AddBond(mutable_atoms=26,paired_atoms=6,bond_type=AromaticDoubleBond)
" \
"AddBond(mutable_atoms=27,paired_atoms=5,bond_type=AromaticDoubleBond)
" \
"AddBond(mutable_atoms=26,paired_atoms=27,bond_type=AromaticSingleBond
)" \
-input_filenames 469.sdf -output 469.to_002.sdf
```

This is the first time you have seen the "`AddBond`" mutate in action. It is a bit of a misnomer – it should probably be called "`AddOrModifyButDoNotRemoveBond`". `AddBond` allows users to either connect atoms with a new bond or modify the bond type of existing bonds. There is also a "`RemoveBond`" mutate.

Once we have the two molecules, do the following:

(1) Generate conformers with molecule:ConformerGenerator
(2) Generate Rosetta params

You can use Tutorials 1 and 5 as examples if you do not remember the steps.

Finally, in your SelectivityDesignD4.xml, remove the "bcl_design" block from the PROTOCOLS group and enable the "score_only" block, as so:

```
<PROTOCOLS>
<Add mover_name="min_cycle"/>
<Add mover_name="score_only"/>
<Add filter_name="d4_ifscore"/>
<Add metrics="lig_ifscore"/>
</PROTOCOLS>
```

And then run the protocol for both inhibitors.

When I ran the calculation, it suggested in both cases that the antagonists bind DRD4 better than DRD2 with dIE_d4-d2 = -1.92 and -2.56 for 469 and 002, respectively. The compounds have IE_d4 of -17.06 and -17.66 for 469 and 002, respectively. The IE_d2 are comparable for both compounds. Although we do not have a calibration curve for this chemical series, the results generally suggest that 002 should be more selective for DRD4 over DRD2 than 469; however, it is because 002 is expected to bind DRD4 greater than 469. What we would have expected based on the experimental data is that the difference is driven by a loss in affinity for DRD2.

Clearly, more work can be done to improve our protein-ligand interaction energy estimates in Rosetta. At the very least, perhaps it is worth considering performing one of the alternative receptor modeling strategies discussed in paragraph four of Part 3. Keep in mind, there is also the possibility that the predicted binding modes for one or both inhibitors is incorrect. This is one of the challenges of structure-based modeling – you must accurately predict both the binding pose and the binding energy. Error in pose prediction propagates into error in binding affinity estimation (on top of energy function inaccuracies, the general chemistry issue related to estimating a binding free energy in the absence of an equilibrium distribution, inaccuracies in explicit water placement, etc.).

Part 4: Designing for DRD4 selectivity against DRD2, DRD3, and DRD5 with one-shot reactions

Okay, so hypothetically here you are, trying to design selective antagonists, and none of your medicinal chemistry friends are available to help you because they are overworked with all of the complex molecules you have already asked them to synthesize for some other project. What can you do?

One option is to reach out to a commercial on-demand synthesis company to identify reactions and reagents that may be useful for your project. Once you have those materials, you need a way to model the product compounds and see if they are potentially good antagonists. One way in which simple reactions are encoded is by artificially adding undefined or non-druglike atoms to connection points (e.g., X, Pb, Nd, etc.) in complementary file pairs.

For example, to mimic an amide coupling, one could make a file of fragments containing nitrogen atoms bonded to a dummy atom and a second file of fragments containing a carboxylic acid group where one of the acidic oxygen atoms is replaced by a dummy atom. Then for any pair of fragments from each of the two files, you know that they can be linked with an amide.

The key then is to make sure that the fragments, or "building blocks" in each file are constructed such that there is minimal risk of competing reactions occurring. One potential caveat to this approach is that it depends upon the availability of preformed building block scaffolds. Unlike the multi-component reactions demoed in Tutorial 3, you will not build complex scaffolds from small pieces using these approaches. However, provided you have access to a rich source of building blocks, this can be a powerful approach to large high-throughput on-demand chemical library screens.

So how do we do perform these reactions in the BCL?

Well, the `AddMedChem` mutate is essentially a generalization of the above-described procedure. Up to this point, we have used `AddMedChem` in a one-directional fashion – we take a library of fragments where each fragment contains a single dummy atom and we attach the fragments to user-specified atoms on our scaffold of interest. However, we can use `AddMedChem` to mimic those simple reactions by carefully specifying our `mutable_element` type.

Try it out!

```
cd Tutorial_7/input/ligands/made_on_demand_style
```

```
bcl.exe molecule:Mutate \
-input_filenames amine.sdf \
-output amide_product.2.sdf \
-implementation \
"AddMedChem(medchem_library=acid.sdf,mutable_elements=X)"
```

The product is the two fragments connected by an amide bond. And, as we saw before, the product retains the coordinate information of the starting fragment.

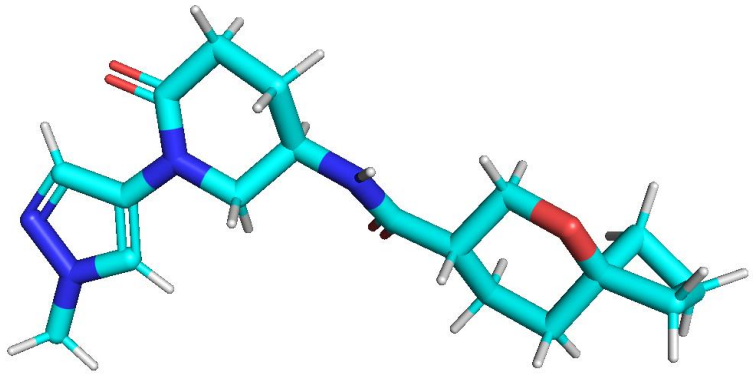

Figure 2. Alchemical amide coupling reaction.

Reverse the positions of the amine and the acid in that reaction and try again.

```
bcl.exe molecule:Mutate \
-input_filenames acid.sdf \
-output amide_product.1.sdf \
-implementation \
"AddMedChem(medchem_library=amine.sdf,mutable_elements=X)"
```

Note that now the coordinate information of the acid is retained instead of the amine.
I have provided a small collection of amide coupling reagents (amine.sample.sdf and acid.sample.sdf). They have already been aligned to the DRD4 orthosteric binding pocket (Tutorial_7/input/receptors/DRD4.LB0.receptor.pdb).

Try designing a protocol in the RosettaScripts framework that will build new molecules using an amide coupling reaction with `AddMedChem`. Are you able to build any selective compounds? How do the predicted binding energies and selectivities compare to the compounds we generated in Part 3?

And that's it! Congratulations. You have finished the tutorial on selectivity design using alchemical mutates and one-shot reactions.

If you have some free time and are interested, think about how you might go about Parts 3 and 4 of Tutorial 6 using one-shot reactions and the `AddMedChem` mover.

References

(1) Jeffries, D. E.; Witt, J. O.; McCollum, A. L.; Temple, K. J.; Hurtado, M. A.; Harp, J. M.; Blobaum, A. L.; Lindsley, C. W.; Hopkins, C. R. Discovery, Characterization and Biological Evaluation of a Novel (R)-4,4-Difluoropiperidine Scaffold as Dopamine Receptor 4 (D4R) Antagonists. *Bioorganic & Medicinal Chemistry Letters* **2016**, *26* (23), 5757–5764. https://doi.org/10.1016/j.bmcl.2016.10.049.

(2) Smith, S. T.; Meiler, J. Assessing Multiple Score Functions in Rosetta for Drug Discovery. *PLoS One* **2020**, *15* (10), e0240450. https://doi.org/10.1371/journal.pone.0240450.

(3) Zhou, Y.; Cao, C.; He, L.; Wang, X.; Zhang, X. C. Crystal Structure of Dopamine Receptor D4 Bound to the Subtype Selective Ligand, L745870. *eLife* **2019**, *8*, e48822. https://doi.org/10.7554/eLife.48822.

(4) Brown, B. P.; Mendenhall, J.; Meiler, J. BCL::MolAlign: Three-Dimensional Small Molecule Alignment for Pharmacophore Mapping. *J. Chem. Inf. Model.* **2019**, *59* (2), 689–701. https://doi.org/10.1021/acs.jcim.9b00020.

(5) Carato, P.; Graulich, A.; Jensen, N.; Roth, B. L.; Liégeois, J.-F. Synthesis and in Vitro Binding Studies of Substituted Piperidine Naphthamides. Part II: Influence of the Substitution on the Benzyl Moiety on the Affinity for D2L, D4.2, and 5-HT2A Receptors. *Bioorg Med Chem Lett* **2007**, *17* (6), 1570–1574. https://doi.org/10.1016/j.bmcl.2006.12.106.