Tutorial 4: Ligand-based multi-component reaction design simulation of selective dopamine receptor D4 antagonists.

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

Date: 01-2022

Background

Dopamine receptors are G-protein coupled receptors (GPCRs) that are heavily associated with many neurological functions. The endogenous small molecule ligand is dopamine. Dopamine receptors can be functionally classified based on their downstream signaling pathways. Broadly, D1-like receptors, which include dopamine receptor D1 (DRD1) and DRD5, activate adenylyl cyclase to stimulate an increase in intracellular cyclic adenosine monophosphate (cAMP). The D2-like receptors, which include DRD2, DRD3, and DRD4, inhibit adenylyl cyclase to reduce the intracellular concentration of cAMP. Dysfunction in dopaminergic signaling has been associated with a number of neurological and psychiatric disorders, such as Parkinson's disease, schizophrenia, ADHD, and addiction. These disorders have few pharmacological interventions to reduce diseases severity. Therefore, dopamine receptors represent an important therapeutic target for an unmet clinical need.

In this tutorial, we will use the BCL to design new candidate small molecule orthosteric antagonists for DRD4 using a reaction-based design framework. The tutorial will be split into two sections. The first section will cover the creation of MDL RXN files for reaction-based design. The second section will demonstrate how to execute a reaction with user-specified reagents. At the end there are some questions that integrate what you have learned in Tutorials 1 – 3 with the new material in this tutorial.

Part 1: Preparing an MDL RXN file

The reaction, or RXN, file format is similar in structure to the SDF. An SDF contains blocks of molecules organized into rows of atoms and rows of bonds. For our tutorial on reaction-based design, we will be using a piperazine ring as a constant reagent. So, let's start just by looking at an SDF of piperazine:

```
piperazine
  PyMOL2.3          3D                             0

 16 16  0  0  0  0  0  0  0  0999 V2000
    1.0862    0.1858   -0.9903 C   0  0  0  0  0  0  0  0  0  0  0  0
   -1.2829    0.6961   -0.2533 C   0  0  0  0  0  0  0  0  0  0  0  0
   -1.0862   -0.1857    0.9903 C   0  0  0  0  0  0  0  0  0  0  0  0
    1.2830   -0.6960    0.2534 C   0  0  0  0  0  0  0  0  0  0  0  0
   -0.6440   -0.6818   -1.5984 H   0  0  0  0  0  0  0  0  0  0  0  0
    0.4645   -0.8770    2.1009 H   0  0  0  0  0  0  0  0  0  0  0  0
   -0.3575    0.2458   -1.3197 N   0  0  0  0  0  0  0  0  0  0  0  0
    0.3575   -0.2458    1.3198 N   0  0  0  0  0  0  0  0  0  0  0  0
    1.4337    1.2236   -0.7688 H   0  0  0  0  0  0  0  0  0  0  0  0
    1.7144   -0.1956   -1.8319 H   0  0  0  0  0  0  0  0  0  0  0  0
   -1.0317    1.7547   -0.0019 H   0  0  0  0  0  0  0  0  0  0  0  0
   -2.3555    0.6812   -0.5660 H   0  0  0  0  0  0  0  0  0  0  0  0
   -1.7144    0.1956    1.8319 H   0  0  0  0  0  0  0  0  0  0  0  0
   -1.4337   -1.2235    0.7687 H   0  0  0  0  0  0  0  0  0  0  0  0
    1.0318   -1.7546    0.0019 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.3556   -0.6811    0.5660 H   0  0  0  0  0  0  0  0  0  0  0  0
  1  4  1  0  0  0  0
  1  7  1  0  0  0  0
  1  9  1  0  0  0  0
  1 10  1  0  0  0  0
  2  3  1  0  0  0  0
  2 11  1  0  0  0  0
  2 12  1  0  0  0  0
  3  8  1  0  0  0  0
  3 13  1  0  0  0  0
  3 14  1  0  0  0  0
  4 15  1  0  0  0  0
  4 16  1  0  0  0  0
  2  7  1  0  0  0  0
  5  7  1  0  0  0  0
  4  8  1  0  0  0  0
  6  8  1  0  0  0  0
M  END
$$$$
```

The first three lines are name lines. The fourth line begins with the numbers of atoms and bonds, respectively, which in this case reads 16 and 16. The V2000 SDF format has a limit of 999 atoms per entry. The next block of lines corresponds to each atom in the molecule. If an SDF is loaded into the BCL, the atom vector indices will correspond to these rows (except they will be 0-indexed). The first three columns of the on the atom rows are the X, Y, and Z coordinates, respectively. The element type follows in the next column. The columns following the element type are used to indicate isotope mass deviations for a particular element and atomic charge. There are generally five or more columns that are unused and contain only zeroes. We will get back to those later.

The bond block columns are organized in triplets indicating the two bonded atom partners beginning with the lower index atom followed by the bond type connecting the two atoms. Bond orders are specified as expected – single bonds with ‘1’, double bonds with ‘2’, and triple bonds with ‘3’. Aromatic structures are frequently notated as alternating single and double bonds (“Kekule form”); however, aromatic bonds can also be explicitly indicated with ‘4’ in the bond order column. The fourth column may specify stereoscopic information.

The end of a single molecule is indicated with “M END”. Afterward, “$$$$” is used as a separator for distinct molecule entries. The “$$$$” is the primary difference between a MOL file and a V2000 SD file, as the latter can contain multiple molecules.

For more detailed references on SD file format, check out http://c4.cabrillo.edu/404/ctfile.pdf and http://www.nonlinear.com/progenesis/sdf-studio/v0.9/faq/sdf-file-format-guidance.aspx.
RXN files differ from SD files in a couple primary ways. Let’s look at a RXN for a 4-component split-ugi reaction.

```
$RXN

      Mrv2113  112920210930

  4  1
$MOL

  Mrv2113 11292109302D

  6  6  0  0  0  0            999 V2000
  -11.0859    0.8250    0.0000 N   0  0  0  0  0  0  0  0  0  1  0  0
  -10.3714    0.4125    0.0000 C   0  0  0  0  0  0  0  0  0  2  0  0
  -10.3714   -0.4125    0.0000 C   0  0  0  0  0  0  0  0  0  3  0  0
  -11.0859   -0.8250    0.0000 N   0  0  0  0  0  0  0  0  0  4  0  0
  -11.8004   -0.4125    0.0000 C   0  0  0  0  0  0  0  0  0  5  0  0
  -11.8004    0.4125    0.0000 C   0  0  0  0  0  0  0  0  0  6  0  0
  1  2  1  0  0  0  0
  1  6  1  0  0  0  0
  2  3  1  0  0  0  0
  3  4  1  0  0  0  0
  4  5  1  0  0  0  0
  5  6  1  0  0  0  0
M  END
$MOL

  Mrv2113 11292109302D

  4  3  0  0  0  0            999 V2000
   -7.3661   -0.7145    0.0000 C   0  0  0  0  0  0  0  0  0 10  0  0
   -7.7786   -0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   -7.3661    0.7145    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
   -8.6036   -0.0000    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
  1  2  1  0  0  0  0
  2  3  1  0  0  0  0
  2  4  2  0  0  0  0
M  END

$MOL

  Mrv2113 11292109302D

  4  3  0  0  0  0            999 V2000
   -4.3607    0.7145    0.0000 C   0  0  0  0  0  0  0  0  0  8  0  0
   -4.7732   -0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  9  0  0
   -4.3607   -0.7145    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
   -5.5982   -0.0000    0.0000 O   0  0  0  0  0  0  0  0  0  7  0  0
  1  2  1  0  0  0  0
  2  3  1  0  0  0  0
  2  4  2  0  0  0  0
M  END
$MOL
```

```
  Mrv2113 11292109302D

  2  1  0  0  0  0            999 V2000
   -1.6500   -0.0000    0.0000 N   0  0  0  0  0  0  0  0  0 11  0  0
   -2.4750   -0.0000    0.0000 C   0  0  0  0  0  0  0  0  0 12  0  0
  1  2  3  0  0  0  0
M  END
$MOL

  Mrv2113 11292109302D

 15 15  0  0  0  0            999 V2000
    3.8966    1.4438    0.0000 N   0  0  0  0  0  0  0  0  0  1  0  0
    4.6111    1.0312    0.0000 C   0  0  0  0  0  0  0  0  0  2  0  0
    4.6111    0.2062    0.0000 C   0  0  0  0  0  0  0  0  0  3  0  0
    3.8966   -0.2062    0.0000 N   0  0  0  0  0  0  0  0  0  4  0  0
    3.1821    0.2063    0.0000 C   0  0  0  0  0  0  0  0  0  5  0  0
    3.1821    1.0313    0.0000 C   0  0  0  0  0  0  0  0  0  6  0  0
    3.8966    2.2688    0.0000 C   0  0  0  0  0  0  0  0  0  9  0  0
    3.1821    2.6813    0.0000 O   0  0  0  0  0  0  0  0  0  7  0  0
    4.6111    2.6813    0.0000 C   0  0  0  0  0  0  0  0  0  8  0  0
    3.8966   -1.0312    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    4.6111   -1.4438    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.1821   -1.4437    0.0000 C   0  0  0  0  0  0  0  0  0 10  0  0
    4.6111   -2.2688    0.0000 N   0  0  0  0  0  0  0  0  0 11  0  0
    5.3256   -1.0313    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
    5.3256   -2.6813    0.0000 C   0  0  0  0  0  0  0  0  0 12  0  0
  1  2  1  0  0  0  0
  1  6  1  0  0  0  0
  2  3  1  0  0  0  0
  3  4  1  0  0  0  0
  4  5  1  0  0  0  0
  5  6  1  0  0  0  0
  1  7  1  0  0  0  0
  7  8  2  0  0  0  0
  7  9  1  0  0  0  0
  4 10  1  0  0  0  0
 10 11  1  0  0  0  0
 10 12  1  0  0  0  0
 11 13  1  0  0  0  0
 11 14  2  0  0  0  0
 13 15  1  0  0  0  0
M  END
```

Notice that there is an initial block beginning with $RXN. In addition to providing some identifying information, this block tells us that there are four reagents and one product. What follows are five blocks designated by $MOL at the beginning and “M END” at the end. **Products must follow reactants.** Each of these five blocks should look familiar – they are formatted very similarly to our SDF of piperazine. There is one crucial difference – notice that there are non-zero values in the third column from the right in the atoms section of each $MOL block. **These values indicate the mapping of reactant atoms to product**

atoms and form the basis of reaction-based design using RXN files. Atoms that contain a non-zero value in this column are referred to as “reactive atoms”.
For example, the first atom in the second $MOL, which is the second reactant in the reaction, maps to the twelfth atom in the fifth $MOL, which is the first (and only) product in the reaction.

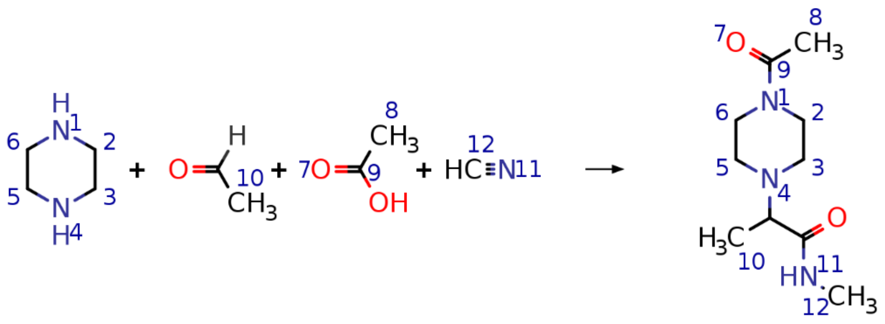

Figure 1. Schematic illustration of MDL RXN prepared for the split-Ugi-4C-diamine reaction. Numbers indicate atom mappings between reactants and product.

So how do you make RXN files? You certainly do not want to manually write them in Vim (though if you did, that would be impressive). There are a number of molecular drawing tools available freely or commercially – Marvin Sketch, ChemDraw, BIOVIA Draw, etc. Personally, I use Marvin Sketch, which is licensed by ChemAxon, but you are free to use whatever suits your needs.

Finally, notice that in the current reaction we do not have any aromatic rings. **It is critical that if we do have aromatic rings in our reaction that we explicitly create the reaction file with aromatic bond order (‘4’) for the corresponding bonds.**

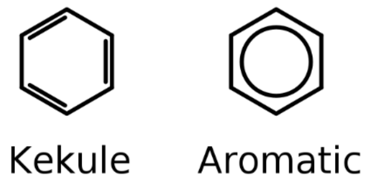

Figure 2. Comparison of Kekule- and aromatic-form benzene ring. Use the aromatic form when preparing MDL RXN files for the BCL.

This is **not** the case for our SD files, so please take care to prepare the files correctly.

For Part 2 we are going to use a variant of this reaction in which the second reactant is a formaldehyde. Can you create a reaction file that encodes the reaction above with a formaldehyde in place of the acetaldehyde? (If you do not have a readily-available molecular drawing software package, no worries – we have prepared one for you).

Part 2: Introduction to reaction-based design applications in the BCL

The primary application that we will use is called molecule:React. The main syntax is as follows:

```
bcl.exe molecule:React \
-starting_fragments <primary reactant> \
-reagents <pool of reagents> \
-reactions <reaction directory> \
-routine <Random/Exhaustive>
-output_filename <file in which to write output>
```

The `starting_fragments` option is mandatory. In principle, we could supply a single file full of reagents and match all the reactant pairs to the RXN file and then perform all of the reactions. Indeed, this latter effect can be achieved either by passing the same file to both `starting_fragments` and `reagents`, or if it is a 2-component reaction by passing one set of reactants as the starting_fragments and the other as the reagents. However, usually we want to do something more targeted (i.e., we have a few scaffolds that are functionalized, and we want to modify them through different reaction/reactant combinations).

There are two benefits to having a separate flag for `starting_fragments` vs. supplying everything in `reagents`. First, we already mentioned it allows increased control over how reactants are combined. Note that neither `starting_fragments` nor `reagents` are required to indicate correspond to any specific position of the reactants in the reaction (i.e., `starting_fragments` could contain fragments that are in either the first and/or second reactant positions). Second, the `starting_fragments` flag allows us to specify which substructure's coordinate information we try to preserve.

The reactions flag specifies which directory contains our RXN files. This flag takes a directory path, not a file. If nothing is specified, it will default to a few predefined reactions in `bcl/rotamer_library/functional_reactions/`. It will signal the application to parse any files ending in ".rxn".

The `routine` flag currently accepts one of two options: `Random` or `Exhaustive`. In both cases, the application will identify all possible valid reactions given the provided `starting_fragments`, `reagents`, and `reactions`. If `routine` is `Random`, then only one random reaction will be executed and the products output to `output_filename`. If the `routine` is `Exhaustive`, then all valid reactions will be executed and the products output to `output_filename`.

Subsection 2A – 2-component reactions

Before we jump into a 4-component reaction, let's start with a few simpler reactions and the different applications that are available for reaction-based design in the BCL.

The first reaction we will look at is a classic 2-component Buchwald-Hartwig coupling:

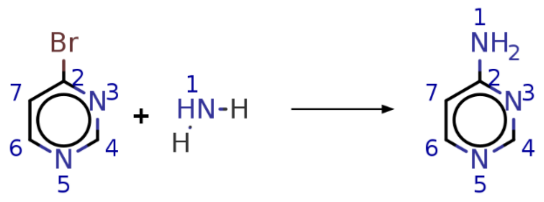

Figure 3. Schematic illustration of MDL RXN prepared for the Buchwald-Hartwig 2C reaction. Numbers indicate atom mappings between reactants and product. Note that the aromaticity of the rings is explicitly shown.

Let's do an example to make these ideas clearer. As our primary reactant, we will extract the amide-linked core rings of the targeted therapeutic imatinib, which is a tyrosine kinase inhibitor that preferentially inhibits Abl, c-Kit, and PDGFR. **Please note that the following does not reflect the actual synthetic route used for the creation of imatinib and is just for illustrative purposes**.

We will modify the scaffold by brominating the pyrimidine such that it matches the first reactant in the RXN file.
<image>

We will then try reacting it with a series of previously prepared amines.

```
bcl.exe molecule:React \
-starting_fragments input/2c/buchald-hartwig/imatinib_core.br.sdf \
-reagents input/2c/buchald-hartwig/aryl_amine.fluorinated.sdf \
-output_filename output/imatinib_core.buchald-hartwig.2.sdf \
-reactions input/2c/buchald-hartwig/rxns/ \
-routine Exhaustive
```

What does your output look like? Did the reaction work as you expected? Here is what I got:

<image>

There are several more reagent files in the same directory. Try the same reaction again but with a different reagent pool. After you do that, try passing one of the amine reagents files as the `starting_fragments` and pass the "imatinib_core.br.sdf" file as `reagents`. How did this change your output?

As you will also see when we begin alchemical design in Tutorial 5, it is a theme in the BCL molecule mutate classes that we try to preserve pose information as we perturb molecules. Due to implementation differences between the different strategies for mutation, however, the way in which we achieve this effect differs between reaction- and alchemical-based design. At times, the time cost of preserving this information is not worth it.

Let's run the same design again, but this time let's pass the `ligand_based` flag.

```
bcl.exe molecule:React \
-starting_fragments input/2c/buchald-hartwig/imatinib_core.br.sdf \
-reagents input/2c/buchald-hartwig/aryl_amine.fluorinated.sdf \
-output_filename output/imatinib_core.buchald-hartwig.2.lb.sdf \
-reactions input/2c/buchald-hartwig/rxns/ \
-ligand_based \
-routine Exhaustive
```

Compare "output/imatinib_core.buchald-hartwig.2.lb.sdf" and "output/imatinib_core.buchald-hartwig.2.sdf". Note that the latter perfectly preserves the coordinate information of the atoms in "input/2c/buchald-hartwig/imatinib_core.br.sdf" (sans the Br), while the former does not. You may also notice that the ligand_based flag makes the simulation faster. How much faster is it to do the calculation with the `ligand_based` flag enabled?

The `ligand_based` flag more or less takes the raw output from the reaction atom mapping process and runs BCL::Conf to generate a 3D conformer. The options for BCL::Conf can be controlled with the `sample_confs` flag.

There is another 2-component reaction in the "BCL_Workshop_2022/Tutorial_4/input/2c" directory called the Ullmann-type reaction. There is also a different version of the imatinib scaffold and reagent files. Before you run the reaction, look at the RXN file and visualize the reactants with PyMOL. What are all the ways in which this reaction is going to yield different products than the previous reaction? Do you think you will get fewer or more total products from running the `Exhaustive` routine?

After you think about the questions, run the reaction with the different reagent sets. Were you correct? Do you understand the output?

One side note – There is an additional way in which we can perform 2-component reactions. In Tutorial 7 Part 4 we will demonstrate how to use the AddMedChem alchemical mutate in the `molecule:Mutate` application to mimic common 2-component coupling reactions that you may find in on-demand synthesis libraries.

When you're ready, move on to the subsection on 3-component reactions.

Subsection 2B – 3-component reactions

Performing 3-component reactions does not differ substantially from performing 2-component reactions. The primary challenge is related to 3D conformer generation – it can be more difficult to preserve *exact* coordinate information of a `starting_fragment` in complicated products. In cases where this is not possible, we do our best to retain a conformer *near* the `starting_fragment` pose that allows a reasonable geometry to be obtained.

For our test case, let's pretend that we are building a covalent inhibitor using an Ugi Tetrazole reaction like the one demonstrated in Sutanto et al. 2021 (https://pubmed.ncbi.nlm.nih.gov/33536213/). Technically, this is a 4-component reaction, but because TMS-N3 is never substituted to derivatize the product we can take its contribution to our product for granted and not worry about mapping. That leave 3-components in our RXN file:

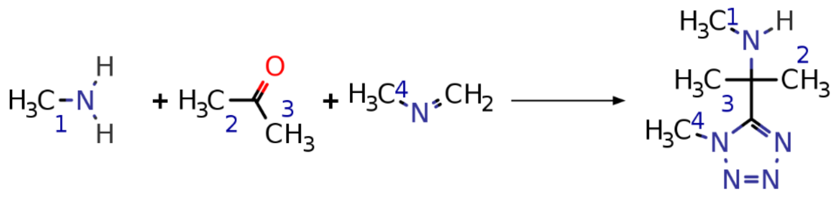

Figure 4. Schematic illustration of MDL RXN prepared for the Ugi-tetrazole 3C reaction. Numbers indicate atom mappings between reactants and product. Note that technically this is a 4-component reaction, but we formulated it as a 3-component reaction because one of the reactants has atoms that do not map to any reactant that we can modify in our reagent pool.

Note that the isonitrile in the third reactant position is written in the neutral resonance state rather than the zwitterion resonance state because the BCL does not currently contain an atom type for carbanions (this is also on the list of things to add). This is a complicated product that introduces a stereocenter.

When doing structure-based covalent inhibitor design, we usually want to keep the geometry of the covalent warhead fixed. This is because we want the remainder of the non-covalent portion of the molecule to be designed to both reversibly bind the pocket and stabilize the orientation of the warhead needed to form the covalent adduct. It defeats the purpose of having a covalent warhead if the remainder of the molecule binds in such a way that we cannot form a covalent bond with the receptor.
Let's give it a whirl. We will use an acrylamide as our `starting_fragment`, which is a common covalent warhead. I have selected a random ketone and isonitrile with which to react the acrylamide.

Run the following:

```
bcl.exe molecule:React \
-starting_fragments acrylamide_amine.sdf \
-reagents all_no_acryl.sdf \
-reactions rxn/ \
-output_filename product.sdf \
-routine Random
```

How did it go? If your run was anything like mine, this was unsuccessful. Lame. Why was it unsuccessful? Does this mean the reaction product cannot be made? Let's look at some of the output messages from the run. My terminal looks like this:

```
<skipping a few lines>
=std=bcl::chemistry=> Done associating reactants with reactions
=std=bcl::app=> Setting pose-dependent options
=std=bcl::app=> Ligand-based: false
=std=bcl::app=> Fix bad geometry: false
=std=bcl::app=> Fix bad ring geometry: false
=std=bcl::app=> Extend adjacent atoms: 0
=std=bcl::app=> Performing a random reaction!
=std=bcl::chemistry=> Skipping drug-likeness filter!
=std=bcl::chemistry=> Skipping drug-likeness filter!
=std=bcl::chemistry=> Getting atom indices for conformer sampling...
=std=bcl::chemistry=> Generating single conformer
=std=bcl::chemistry=> Molecule cleaning failed to generate a valid 3D
conformer!
=std=bcl::chemistry=> Defined: true
=std=bcl::chemistry=> Has good geometry: false
=std=bcl::chemistry=> Final molecule size: 24
=std=bcl::chemistry=> Returning null...
=std=bcl::util=> Reactions has run for 0.00212 seconds
```

Hmm… It says, "Molecule cleaning failed to generate a valid 3D conformer!". The important thing to note is that the "Has good geometry" line returns "false", which also causes the final "Returning null…" line. This tells me that the product had a very poor 3D conformer.

There are some more lines that may be helpful. It says up above that "Fix bad geometry" and "Fix bad ring geometry" both are false. These are command-line options that are enabled during pose-sensitive reaction-based design. Typically, the only dihedrals that can be sampled during 3D conformer generation in reaction-based design are those that are *not* in the starting_fragment. We can extend this selection to include atoms with bad geometry. We can further extend this selection to include up to six adjacent atoms from any atom *not* in the starting fragment, which can help with tricky connection points.

Let's try it out and see if it helps.

```
bcl.exe molecule:React \
-starting_fragments acrylamide_amine.sdf \
-reagents all_no_acryl.sdf \
-reactions rxn/ \
-output_filename product.sdf \
-routine Random \
-fix_geometry -fix_ring_geometry -extend_adjacent_atoms 4
```

Any luck? None for me, I am afraid. The combination of reagents we chose and the reaction in question is stretching the limits of our pose retention capabilities at this time. In situations like this, the code in place to try and preserve 3D coordinate information during a tricky design is preventing us from getting a good 3D conformer at all. So, the best thing to do at this point is wholesale it and build the molecule from scratch. How do we do that? We pass the `ligand_based` flag and specify a `sample_confs` object with `generate_3D` enabled.

```
bcl.exe molecule:React \
-starting_fragments acrylamide_amine.sdf \
-reagents all_no_acryl.sdf \
-reactions rxn/ \
-output_filename product.sdf \
-routine Random \
-sample_confs \ "(conformation_comparer=SymmetryRMSD,tolerance=0.25,\
cluster=1,generate_3D=1,\
max_iterations=2000,max_conformations=1,\
change_chirality=0)" \
-ligand_based
```

Ah, and this time we have a successful reaction with a valid 3D conformer. But you may say, "Ben, that's swell and all, but I really need to keep that acrylamide really close to where it was at. This is not going to work for me." **I hear you. I feel similarly. We have a few options.**

First, you can do a standard rigid substructure-based alignment back to the acrylamide.

```
bcl.exe molecule:AlignToScaffold \
acrylamide_amine.sdf \
product.sdf \
product.ats.sdf
```

This looks okay-ish…

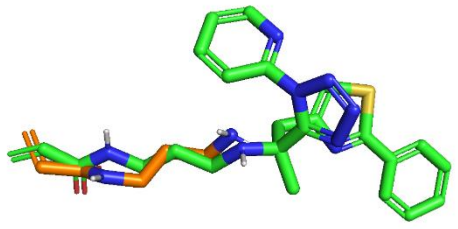

Figure 5. Covalent inhibitor product rigidly aligned via partial substructure matching of the acrylamide warhead. Orange is the starting acrylamide reactant. Green is the product.

But not good enough.

We could alternatively generate an ensemble of conformers, align all of them to the acrylamide, and then choose the one that aligns closest. This would be doable in a few commands using `molecule:ConformerGenerator` followed by `molecule:AlignToScaffold` and then `molecule:Properties`. That sounds like something that should its own application.

Indeed, there is an app in-development right now undergoing some alpha testing that is meant to allow users to assemble combination of different alignment methods, pose scoring tools, etc. under one umbrella app. The app is called `cheminfo:MoleculeFit`, and you may remember it from Tutorial 1. It does not, unfortunately, currently restrict the SampleByParts atoms; however, we can do this in a two-step process.

First, let's add some info the MDL SDF to indicate that we only want to allow sampling of dihedral angles in the acrylamide atoms of our new product molecule.

```
bcl.exe molecule:SetSampleByPartsAtoms \
-input_filenames product.sdf \
-atom_comparison_type ElementType \
-bond_comparison_type BondOrderOrAromatic \
-reference_mol acrylamide_amine.sdf \
-disable_complement_indices \
-output product.labeled.sdf
```

Our output SDF contains the following property block at the end of the file:

```
> <SampleByParts>
11 12 13 14 15 16 17 2 3
```

These values correspond to the acrylamide heavy atoms in our molecule. You can check yourself, too, by visualizing in PyMOL.

Now when we generate conformers for product.labeled.sdf, no matter whether it is via the command line with `molecule:ConformerGenerator` or if it is in some other application, only the dihedrals containing the specified atoms will be sampled.

Second, let's perform the flexible alignment.

```
bcl.exe cheminfo:MoleculeFit \
-input_filenames product.labeled.sdf \
-output_filename product.fit.sdf \
-scaffold_fragments acrylamide_amine.sdf \
-routine 2 \
-sample_confs "(conformation_comparer=SymmetryRMSD,tolerance=0.125,\
generate_3D=0,cluster=true,max_iterations=2000,\
max_conformations=1000,change_chirality=0)" \
-add_h \
-refine_alignment \
-bond_comparison_type BondOrderAmideOrAromaticWithRingness
```

And now we can see that we have a valid 3D conformer that aligns the acrylamide back to its original pose.

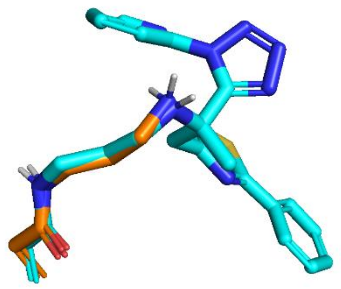

Figure 6. Covalent inhibitor product flexibly aligned via partial substructure matching of the acrylamide warhead. Orange is the starting acrylamide reactant. Blue is the product. Subsection 2C: Designing a library of piperazine core molecules with a split-Ugi 4-component reaction

Subsection 2C: Designing a library of piperazine core molecules with a split-Ugi 4-component reaction

Piperazine is a common molecular scaffold, often considered a “privileged substructure” because of its relatively high frequency in bioactive compounds. It is well-established that piperazine derivatives are effective scaffolds for orthosteric dopamine receptor antagonists[1]. There are, however, millions of ways in which a piperazine scaffold can be derivatized. Here, we will use a split-Ugi diamine 4-component reaction.

By this point you are familiar with the `molecule:React` app, so let’s use a different one. There is an application called `molecule:Mutate` that can perform many perturbations to a molecule.

```
bcl.exe molecule:Mutate –help
```

You will see under the implementation flag several choices. These largely represent the mutates that we can perform in the alchemical drug design framework. **We will explore these in detail in Tutorials 5 – 7**. For now, I just want you to notice that one of the available implementations is `React`. This is more or less a mini version of the `molecule:React` app that meets the qualifications to derive from `Frag–mentMutateInterface`. This is important because `FragmentMutateInterface` is the primary drug design interface class between BCL and Rosetta (for right now).

We can pilot `molecule:Mutate` with a simple case where we have one reactant that matches each potential position in the split-Ugi reaction (Figure 1).

Run the following:

```
bcl.exe molecule:Mutate \
-input_filenames piperazine.sdf \
-output product.sdf \
-implementation "React(reactions_directory=rxns/, \
reagents=demo_reagents.sdf, ligand_based=0, \
fix_geometry=1, extend_adjacent_atoms=4)"
```

Did we obtain a product? Hopefully the answer is yes. Visualize the resultant product and the piperazine starting fragment in PyMOL. How did we do? Does the piperazine core of our new molecule retain the coordinate information of the starting_fragment? If not, it should at least be close.

One caveat to `molecule:Mutate` is that it can only return a single molecule from a single product. This is fine for stochastic sampling – we can just run the mutate a bunch of times with different random seeds – but it is not ideal for enumeration. So, for now, let's go back to `molecule:React`.

Let's extract a subset of reagents from the larger set:

```
bcl.exe molecule:Reorder \
-input_filenames reagents_le_20.sdf \
-randomize \
-output_max 50 \
-output reagents_le_20.rand_50.sdf
```

Add in our formaldehyde:

```
cat formaldehyde.sdf reagents_le_20.rand_50.sdf > reagents.demo.sdf
```

And then let's run the reaction. You can run it retaining pose information:

```
bcl.exe molecule:React \
-starting_fragments piperazine.noh.sdf \
-reagents reagents.demo.sdf \
-output_filename product.sdf \
-reactions rxns/ \
-routine Exhaustive \
-fix_geometry -fix_ring_geometry -extend_adjacent_atoms 4
```

Or you can run it in generating a random conformer:

```
bcl.exe molecule:React \
-starting_fragments piperazine.noh.sdf \
-reagents reagents.demo.sdf \
-output_filename product.sdf \
-reactions rxns/ \
-routine Exhaustive \
-ligand_based -sample_confs
"(conformation_comparer=SymmetryRMSD,tolerance=0.25,\
generate_3D=1,cluster=true,max_iterations=1000,\
max_conformations=1,change_chirality=0)"
```

The latter generates about one product molecule every second while the former requires ~4 – 6 seconds per product molecule. This procedure will generate 49 molecules in `ligand_based` mode, and I believe only a couple fail in pose-dependent mode.

Once you have your small ensemble, try to answer the following questions using BCL cheminformatics tools we reviewed in Tutorials 1 – 3:

1. How many of our product molecules violate Lipinski's Rules? How many violate Veber's rules?

2. Generate statistics on the following properties: number of hydrogen bond donors, number of hydrogen bond acceptors, number of rotatable bonds, topological polar surface area, logP, and synthetic accessibility. How many of the molecules have fewer than 5 HBD, fewer than 10 HBA, fewer than 10 rotatable bonds, a TPSA below 140, a clogP between 0 and 3, and a synthetic accessibility score below 4?

3. Using the QSAR model we built earlier for DRD4, estimate localPPV values for the design. Which design has the highest localPPV?

4. Using the equation below and the BCL descriptor framework, write a property to estimate relative selectivity of each design for DRD4 relative to the other dopamine receptors.

Congratulations! You have finished Tutorial 4.