Tutorial 1. Molecule processing, handling, and standard tasks

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

Date: 01-2022

In this tutorial we will review basic but important capabilities and syntax for BCL standalone use. The motivation for this section is that it is a useful prerequisite to performing small molecule drug design. It is challenging to do virtual screening or drug design if we are unable to perform quality control on our input structures, orient them in space the way that we need, or compute chemical properties. Moreover, if your goal is to use the BCL in the context of the BCL-Rosetta integration, it is helpful to know a bit about the functionality of the BCL by itself because it will help in understanding why the BCL-Rosetta integration is designed the way that it is.

This tutorial will be organized into several sections. First, we will quickly review the overarching syntax used to call BCL applications. Second, we will perform some standard small molecule processing and quality control tasks so that you can get a feeling for how to manipulate lists of molecules in the BCL. Third, we will introduce you to fragment creation. Finally, we will review basic usage for small molecule conformer generation, comparisons, and molecular alignment.

Some of these topics are independently capable of filling an entire tutorial. Here, we will do just enough that you can continue to explore on your own. For details and background, we recommend checking out a few manuscripts[1–5]. At the time of writing, we also have a pedagogical manuscript in revision at *Frontiers in Pharmacology*. The contents of this tutorial align with sections 2 – 5 of that manuscript, which we have provided with the tutorial materials as a preprint[1]. For additional details, please use the aforementioned manuscripts as reference material.

Before we get started, it will make your life easier if you set the path to your BCL directory as an environment variable. For example, the main BCL directory for these tutorials in the Meiler Lab is in `/sb/apps/bcl/bcl`. In bash, we can set this as an environment variable by doing the following:

```
export BCL=/sb/apps/bcl/bcl
```

In tcsh, this is accomplished with a similar command:

```
setenv BCL "/sb/apps/bcl/bcl"
```

We can also add the BCL executable to our PATH environment variable.

```
export PATH=/sb/apps/bcl/bcl/build/linux64_release/bin:$PATH
```

Note that the "bcl.exe" is interchangeable with "bcl-apps-static.exe" in `${BCL}/build/linux64_release/bin`

Let's get started!

Part 1. Introduction to BCL command-line syntax

Much like Rosetta, the BCL is an application-based software package written in C++. Unlike Rosetta, the primary API is packaged into a few core application groups. The application groups each contain multiple applications.

To view the application groups and associated applications, run the BCL help command:

```
bcl.exe --help
```

The BCL has application groups for cheminformatics, protein folding, machine learning, and other tasks.
The syntax to access a specific BCL application is to first specify its application group in all lower-case letters, a ":", and then the application in FirstLetterIsCapitalizedInEachWord syntax.

```
bcl.exe appgroup:Application
```

For example, to view the applications associated with the application group "molecule" run the application group help command:

```
bcl.exe molecule:Help
```

The help menu for any application cans similarly be accessed as

```
bcl.exe appgroup:Application --help
```

These help options list the basic arguments and parameters available for each application. More detailed help options are available for individual application arguments. We will demonstrate examples of this functionality as we go. The important thing to note is that liberal use of the `help` keyword on the command-line will display internal documentation for the BCL.

Part 2. Molecule preparation and processing

Subsection 2A – MDL SD Files

The BCL reads molecules in the MDL SDF format. This is a common format used in computational chemistry and cheminformatics. An SDF contains blocks of molecules organized into rows of atoms and rows of bonds. Let's look at an SDF of a piperazine ring:

```
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 look at those columns in more detail in Tutorial 4 when we discuss reaction-based drug design.

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.

Subsection 2B – molecule:Filter

Molecules obtained from public databases, manual drawing, conversion from 1D strings or 2D topologies, etc., may contain errors. These errors can be in the form of incorrect bond order assignments for a particular protonation state, formal charge (e.g., if a compound is isolated from a salt complex), invalid atom types (e.g., carbon atoms making 5 or 6 bonds, disallowed hybridizations, etc.), severely clashed geometries, or other errors. Consequently, an important component of any cheminformatics project is identifying and filtering molecules that fail certain criteria. In the BCL, this step is typically performed with the molecule:Filter application.

In this section we will perform a series of molecule processing steps on a dataset of molecules that we use for benchmarking small molecule conformation generation algorithms. The dataset is called the Platinum Diverse Dataset, and it is a subset of high-quality ligands in their protein-bound 3D conformations assembled by Friedrich et al. 2017B[6].

Unless otherwise stated, all commands in this tutorial are run from the `BCL_Workshop_2022/Tutorial_1/inputs/` directory.

```
bcl.exe molecule:Filter \
-input_filenames platinum_diverse_dataset_2017_01.sdf.gz \
-output_matched platinum_diverse_dataset_2017_01.matched.sdf.gz \
-output_unmatched platinum_diverse_dataset_2017_01.unmatched.sdf.gz \
-add_h -neutralize -defined_atom_types -simple \
-logger File platinum_diverse_dataset_2017_01.Filter.log
```

This command performs the following actions as the molecules are read:

- loads the SDF platinum_diverse_dataset_2017_01.sdf (`input_filenames`)
- saturates all molecules with hydrogen atoms (`add_h`)
- neutralizes any formal charges (`neutralize`)

And then filters based on the following criteria:

- whether the molecules have valid atom types (e.g., carbon atoms making five covalent bonds are not valid) (`defined_atom_types`)
- whether the molecules have simple/defined connectivity (i.e., that there are not multiple non-bonded components, such as salt complexes, in the input file)(`simple`)

Molecules are output into one of two files:

- Molecules that pass all criteria (have valid atom types and have simple connectivity) are output to "platinum_diverse_dataset_2017_01.matched.sdf.gz"
- Molecules that fail one or more criteria are output to "platinum_diverse_dataset_2017_01.unmatched.sdf.gz"

The terminal output reads:

```
=std=bcl::app=> Loaded 2859 molecules in 00:00:01
=std=bcl::app=> 2859 molecules passed filter with defined gasteiger atom
types   in 00:00:00, 0 additional molecules filtered out
=std=bcl::app=> 2859 molecules passed filter with simple connectivity ->
not a molecular complex containing at least one fragment from   in
00:00:00, 0 additional molecules filtered out
=std=bcl::app=> Wrote 2859 molecules that matched all criteria in 00:00:00
```

In this case, all molecules pass the filter. This allows the user to review the molecules that failed the filter and choose to either fix them or continue without them.

The `molecule:Filter` application can also separate molecules by property and/or substructure. Filter out molecules that contain 10 or more rotatable bonds and a topological polar surface area (TPSA) less than 140 Å^2:

```
bcl.exe molecule:Filter \
-input_filenames platinum_diverse_dataset_2017_01.sdf.gz \
-output_matched platinum_diverse_dataset_2017_01.veber_pass.sdf.gz \
-output_unmatched platinum_diverse_dataset_2017_01.veber_fail.sdf.gz \
-add_h -neutralize \
-compare_property_values TopologicalPolarSurfaceArea less 140 \
NRotBond less_equal 10 \
-logger File platinum_diverse_dataset_2017_01.veber.log
```

Of 2859 molecules, 395 were first filtered out for have a TPSA ≥ 140 Å^2, and then an additional 84 molecules that had greater than 10 rotatable bonds were filtered out. Notice that the filters are applied sequentially, and molecules must pass both filters to be output to the matched file. Alternatively, the any flag can be specified such that if a molecule meets any one of the filter criteria, then it is output to the matched file:

```
bcl.exe molecule:Filter \
-input_filenames platinum_diverse_dataset_2017_01.sdf.gz \
-output_matched platinum_diverse_dataset_2017_01.any_pass.sdf.gz \
-output_unmatched platinum_diverse_dataset_2017_01.any_fail.sdf.gz \
-add_h -neutralize \
-compare_property_values TopologicalPolarSurfaceArea less 140 \
NRotBond less_equal 10 \
-any -logger File platinum_diverse_dataset_2017_01.any.log
```

In this example, 2801 molecules passed at least one of the filters and only 58 were filtered out.

One may also filter based on substructure similarity. This is particularly useful if there are specific substructures that are desired or that need to be avoided. For example, we can filter out molecules that contain six-membered aromatic carbon rings (i.e., benzene or benzene-components of larger aromatic ring systems):

```
bcl.exe molecule:Filter \
-input_filenames platinum_diverse_dataset_2017_01.sdf.gz \
-output_matched drugbank_nonexperimental.simple.benzene.sdf.gz \
-output_unmatched drugbank_nonexperimental.simple.no_benzene.sdf.gz \
-contains  bcl/rotamer_library/ring_libraries/individual_rings/000.sdf.gz
```

The rotamer library file contains a benzene ring. In addition to the standard use cases presented here, `molecule:Filter` can identify molecules with clashes in 3D space, conformers outside of some tolerance value from a reference conformer, exact substructure matches, specific chemical properties, and more. To see these options, you can pass the following command:

```
bcl.exe molecule:Filter -help
```

We regularly use `molecule:Filter` prior to drug screening and design.

Subsection 2C – molecule:Unique

We also frequently filter datasets to remove redundant molecules. This is especially important when preparing datasets for QSAR model training and testing. If molecules appear more than once in a dataset, then it is possible that they could appear simultaneously in the training and test sets, leading to an artificial inflation in test set performance. As there are several special considerations to make here, this is a distinct application from `molecule:Filter`.

For this task we use the `molecule:Unique` application. It has four levels at which it can compare and differentiate molecules:
1. Constitutions – compares atom identities and connectivity disregarding stereochemistry
2. Configurations – compares atom identities, connectivity, and stereochemistry
3. Conformations – compares configurations as well as 3D conformations
4. Exact – checks to see whether atom identities and order are equal with the same connectivities, bond orders, stereochemistry, and 3D coordinates.

The first time the BCL encounters a molecule in an SDF it will store in memory. Any additional encounters with the same molecule (at any of the four levels described above) will be marked as duplicate encounters. The default behavior is to output only the first encounter of each molecule. There are cases in which a molecule appears multiple times but has different MDL properties and/or property values. It may not be desirable to lose the stored properties on duplicate compounds. In such cases, the user can choose to merge the properties or overwrite the duplicate descriptors instead.

For example, let's say we are building a virtual screening model for dopamine receptor D3 (DRD3) and D5 (DRD5) antagonists. We have collected molecules that have confirmatory screening data for each of those targets. We may want to know how many molecules overlap between these two datasets. This is something that we can check with `molecule:Unique`.

```
bcl.exe molecule:Unique \
-compare Configurations \
-remove_h \
-neutralize \
-input_filenames DRD3.training_molecules.sdf.gz
DRD5.training_molecules.sdf.gz \
-output ../outputs/DRD3-5.training_molecules.unique_configs.sdf.gz
```

Consider the following questions:

1. How many duplicate molecules do you have at the level of Configurations?
2. How about at the level of Conformations?
3. Does it make sense why one has more duplicates than another?

Importantly, when you discard duplicates, you also discard all of their MDL property information. In our example here, we passed the DRD3 molecules first, which means that duplicates identified in the DRD5 set will be discarded. There are some property labels that are unique between these two files – specifically, "DRD3_Activity_uM" and "DRD5_Activity_uM". If we want to discard duplicates and simultaneously maintain any unique properties on the discarded molecules, we can pass the `merge_descriptors` flag at runtime.

```
bcl.exe molecule:Unique \
-compare Configurations \
-remove_h \
-neutralize \
-input_filenames DRD3.training_molecules.sdf.gz \
DRD5.training_molecules.sdf.gz \
-output ../outputs/DRD3-5.training_molecules.unique_configs.sdf.gz
-merge_descriptors
```

After you run this command, can you think of a way to use `molecule:Filter` to preserve only the molecules that have both the DRD3_Activity_uM and DRD5_Activity_uM result labels? **Hint: use the has_properties flag**. Give it a whirl – how many compounds have both property labels? It should be the same number as the number of duplicates.

Subsection 2D – molecule:Reorder

You may have considered in the previous section that you do not always want to discard the first occurrence of a molecule in a file. This is especially the case if you have redundant molecules within a single dataset. Fortunately, we can sort molecules from a single or multiple input files according to properties of interest. Sort molecules in ascending order by DRD3_Activity_uM and output the top 10 highest affinity molecules:

```
bcl.exe molecule:Reorder \
-input_filenames ../outputs/DRD3-
5.training_molecules.unique_configs.labeled.sdf.gz \
-output ../DRD3-5.training_molecules.unique_configs.sorted.top_10.sdf.gz \
-sort DRD3_Activity_uM \
-output_max 10
```

Repeat the process, but this time sort by the lowest affinity molecules for DRD3.

```
bcl.exe molecule:Reorder \
-input_filenames ../outputs/DRD3-
5.training_molecules.unique_configs.labeled.sdf.gz \
-output \
../DRD3-5.training_molecules.unique_configs.sorted.bottom_10.sdf.gz \
-sort DRD3_Activity_uM \
-reverse \
-output_max 10
```

In this example, the reverse flag indicates that the scores will be sorted from largest to smallest (default behavior is smallest to largest). The output_max flag limits the number of molecules that are output (proceeding in sorted order).

Are there any compounds in the top 10 molecules for DRD3 that have very low affinity for DRD5 (i.e., are selective)? How about any compounds in the bottom 10 molecules for DRD3 that are high affinity for DRD5? Visualize any selective compounds with PyMOL.

Do any of your compounds contain aromatic rings? Aromaticity is automatically detected when reading input files; however, output structures are kekulized (represented as alternating single-double bonds) by default. To output an SDF that contains explicit aromatic bonds, pass the `explicit_aromaticity` flag on the command line.

```
bcl.exe molecule:Reorder \
-input_filenames ../outputs/DRD3-
5.training_molecules.unique_configs.labeled.sdf.gz \
-output ../DRD3-\
5.training_molecules.unique_configs.sorted.bottom_10.explicit_aro.sdf.gz \
-sort DRD3_Activity_uM \
-reverse \
-output_max 10 \
-explicit_aromaticity
```

Compare the output files from our `molecule:Reorder` examples by visualizing the output in PyMOL. Do you notice anything about the two versions of bottom 10 molecules? You should see that PyMOL interprets the aromatic rings differently. This is because in the last file we generated the `explicit_aromaticity` flag set aromatic bond orders to be “4” in the SDF. Check out the SDF using a text editor and you will see.

Subsection 2E – molecule:Coordinates

The last molecule processing application that we will cover is `molecule:Coordinates`. This is an application that performs several convenience tasks related to retrieving coordinate information. First, `molecule:Coordinates` can recenter all molecules in the input file(s) to the origin. Second, it can compute molecular centroids. Neither of these are particularly complex tasks, but they are useful.

Third, and perhaps most importantly, `molecule:Coordinates` can compute statistics on molecular conformers. For example, passing the statistics flag compute statistics on bond lengths, bond angles, and dihedral angles. Passing the `dihedral_scores` flag will compute a per-dihedral breakdown of the BCL 3D conformer score. The BCL 3D conformer score, or ConfScore, computes an amide non-planarity penalty in addition to a normalized dihedral score. Passing the `amide_deviations` and `amide_penalties` will output the amide deviations and penalties on a per-amide basis.

For example, compute all of the described metrics on osimertinib, a third-generation covalent tyrosine kinase inhibitor (TKI).

```
 bcl.exe molecule:Coordinates \
-input_filenames osimertinib.clean.sdf.gz \
-statistics \
-dihedral_scores \
-amide_deviations \
-amide_penalties \
-clash_scores \
-explicit_aromaticity \
-output ../outputs/osimertinib.clean
```

In both the dihedral and amide score output files, the torsions are identified by the atom indices contributing to the central bond of the torsion.

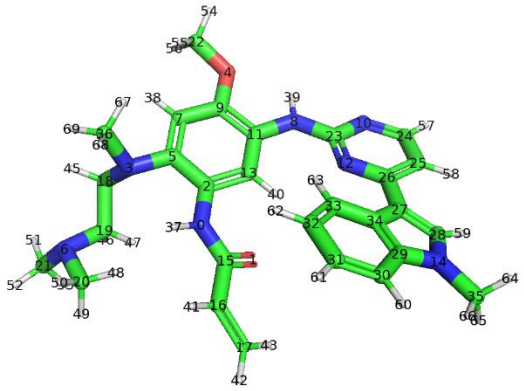

Figure 1. 3D conformation of the third-generation EGFR tyrosine kinase inhibitor osimertinib. Atoms labels are given as 0-indexed atom numbers. Dihedral and amide scores output from `molecule:Coordinates` reference the central bond of the torsion. For example, the dihedral angle of the amide bond in the acrylamide warhead is defined by the torsion of atom indices 2-0-15-16, but in the `molecule:Coordinates` output it would be indicated by atoms 0 and 15.

We can compare these scores to the scores of a conformer with a horrible amide bond (which I perturbed manually):

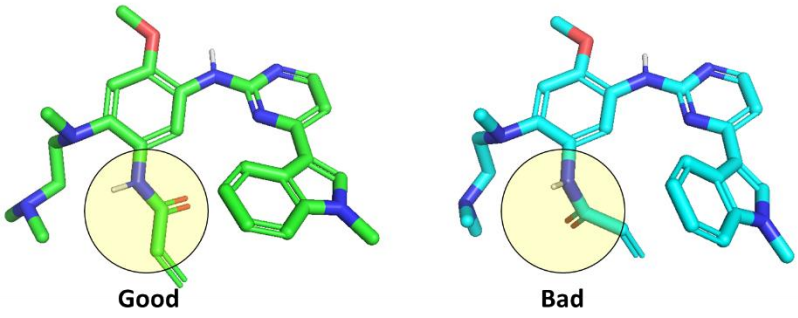

Figure 2. Comparison of acrylamide warhead geometries. (A) Osimertinib with a good acrylamide warhead geometry. The amide deviation from planarity is 3.95 degrees, resulting in no ConfScore penalty. (B) Osimertinib with a bad acrylamide warhead geometry. The amide deviation from planarity is 71.87 degrees, resulting in a ConfScore penalty of 53.17.

Note that the clean osimertinib structure has an amide deviation of approximately 4 degrees and no penalty. The functional form for the amide penalty is <math>. The amide deviation of the bad osimertinib structure is

approximately 71 degrees, and the penalty is very large (53, which is gigantic in ConfScore units; see Mendenhall et al. 2020[4] for details).

As you can see, `molecule:Coordinates` is useful for identifying regions of strain in 3D conformers. It is also useful when comparing conformation sampling algorithms to one another, to different crystal structures, and/or to molecular dynamics (MD) trajectory ensembles.

This concludes Part 2 on molecule processing. Let's keep going!

Part 3. Creating chemical fragments

The BCL application `molecule:Split` gives researchers a tool to derive fragments from starting small molecules to aid in pharmacophore modeling, fragment-based drug discovery, and de novo drug design. There are many different types of fragments `molecule:Split` is able generate from whole molecule(s). For a complete list, check out the help menu options:

```
bcl.exe molecule:Split --help
```

The splits displayed in BCL v4.2 via the help menu are the same as those shown in section 2.5 of Brown et al. 2022[1].

In the previous section, we computed statistics for the third generation TKI osimertinib. Here, let's derive the Murcko scaffold from osimertinib:

```
bcl.exe molecule:Split \
-input_filenames osimertinib.clean.sdf.gz \
-output ../outputs/osi.murcko.sdf.gz -implementation Scaffolds
```

Alternatively, we could remove the Murcko scaffold and return the other components:

```
bcl.exe molecule:Split \
-input_filenames osimertinib.clean.sdf.gz \
-output ../outputs/osi.inverse_scaffold.sdf.gz -implementation
InverseScaffold
```

I frequently use the `molecule:Split` application to remove substructures from molecules. For example, start by splitting osimertinib into its rigid components.

```
bcl.exe molecule:Split \
-implementation Rings \
-input_filenames osimertinib.clean.sdf.gz \
-output ../outputs/osi.Rings.sdf.gz \
-remove_h
```

Let's focus on the indole ring, which is the third fragment in the file. Grab just the indole from the previous output file.

```
bcl.exe molecule:Filter \
-input_filenames ../outputs/osi.Rings.sdf.gz \
-input_start 2 -input_max 1 \
-output_matched ../outputs/osi.indole.sdf.gz
```

Then split the original molecule by largest common substructure using the indole ring as a reference. Set the splitter to return the complement, or inverse, of the largest common substructure.

```
bcl.exe molecule:Split -implementation "LargestCommonSubstructure(\
file=../outputs/osi.indole.sdf.gz,\
atom_comparison=ElementType,\
bond_comparison=BondOrderOrAromaticWithRingness,complement=1)" \
-input_filenames osimertinib.clean.sdf.gz \
-output ../outputs/osi.sans_indole.sdf.gz
```

On visualization, you should see that you now have an osimertinib structure that is missing the indole ring. If you were redesigning osimertinib, it may be useful to remove the indole ring and sample alternative ring structures in that position without perturbing the coordinates of the remainder of the molecule. This is one way to generate the initial scaffold.

A note on substructure comparisons – the BCL encodes molecules as graphs where the edges are bonds, and the atoms are nodes. Edges and nodes are labeled with comparison types. Comparison types are the level of detail with which to describe edges and nodes. For any substructure-based comparison between two or more molecules, some combination of atom and bond comparison types is required. The default combination differs between tasks. In this example, our atom comparison occurs at the level of `ElementType` and our bond comparison at the level of `BondOrderOrAromaticWithRingness`. To see a list of options and the default for this application, you can run the following:

```
bcl.exe molecule:Split -implementation "LargestCommonSubstructure(help)"
```

For example, if atom type resolution occurs at `AtomType`, then an SP3 carbon would match another SP3 carbon but not an SP2. If the resolution is lowered to `ElementType`, then all carbon atoms can match one another independent of their orbital configuration. Similarly, bond type resolutions of `BondOrder` and `BondOrderOrAromatic` will yield different substructure matches – the first case will be sensitive to differences in Kekulization, while the latter case will recognize aromatic systems and compare them accordingly.

Part 4. Generating small molecule conformations

The BCL small molecule conformer generator, also known as BCL::Conf, has undergone several rounds of improvements[4,7]. Its functionality is extensively described in Mendenhall et al. 2020[4] and again reviewed in Brown et al. 2022[1]. Therefore, here we will forego discussing the details of the algorithm and focus instead on frequent use-cases.

Small molecule 3D conformer generation is an important aspect of both ligand- and structure-based computer-aided drug discovery. We rarely know the functionally relevant conformation of a molecule at the outset of a modeling project. Generating discrete conformers is useful for tasks such as protein-ligand docking, small molecule flexible alignment, pharmacophore modeling, non-canonical amino acid rapid rotamer generation, etc.

Briefly, BCL::Conf utilizes a fragment-based rotamer library derived from the crystallography open database (COD) to combine rotamers consisting of one or more dihedral angles according to a statistically derived energy. Clashes are dynamically resolved by iteratively identifying clashed atom pairs and rotating the central-most bonds between them without changing dihedral bins. In this way, conformational ensembles are stochastically generated according to likely rotamer combinations from the COD.

Perhaps more important for this collection of tutorials, BCL::Conf is a critical component of the BCL and BCL-Rosetta drug discovery pipeline. We need a way to rapidly sample 3D conformers when we perturb the chemical structures of molecules. **Thus, while BCL::Conf is a very useful tool by itself, in combination with the graph-based substructure comparison methods, it forms the backbone of the structure-based drug discovery tools in the BCL-Rosetta integration that we will be using in Tutorials 5 – 7.**

BCL::Conf generates 3D conformers of small molecules either from scratch or starting from input coordinates. It can be used to sample a conformational ensemble of the entire protein, or it can be restricted to sample only allowed dihedral angles. Let's give it a whirl.

Subsection 4A – Global conformational ensembles

Start by generating conformers of osimertinib with simple settings.

```
bcl.exe molecule:ConformerGenerator \
-ensemble_filenames -osimertinib.clean.sdf.gz \
-conformers_single_file ../outputs/osimertinib.quick_confs.sdf.gz \
-max_iterations 500 \
-top_models 50
```

There is a one-time setup cost for 2-3 seconds to load the rotamer library into memory, so if you are performing conformer generation on multiple molecules you can avoid repeating this cost by including them all in the some run. Feel free to visualize your conformers in PyMOL.

See the help options for command-line flag details:

```
bcl.exe molecule:ConformerGenerator --help
```

Briefly, it is worth mentioning that `ensemble_filenames` is equivalent in functionality to `input_filenames` in most other applications. The `conformers_single_file` argument outputs all conformers to a single file. The other option is `conformers_separate_files`, which if multiple

molecules are input to `ensemble_filenames` will output a unique SDF for the conformational ensembles of each of the input molecules. The returned conformers are sorted by score, with the best conformers (lower ConfScore) first. The number of iterations and maximum number of final conformers can be specified with the `max_iterations` and `top_models` flags, respectively.

During the conformer generation process, it is often beneficial to take the best scoring conformer of a particular cluster to assemble the final ensemble. This is performed via leader-follower clustering automatically in BCL::Conf unless it is disabled with the `skip_cluster` flag. We typically recommend comparing conformations using the SymmetryRMSD metric at a tolerance of 0.25 Å (all RMSD-based metrics are in Å, all dihedral-based metrics are in degrees). By default, the tolerance is adjusted automatically to yield the desired number of clusters to best represent conformational space; however, a user-provided tolerance is treated as a minimal acceptable difference between clusters.

See Mendenhall et al. 2020 for a detail performance evaluation of BCL::Conf with different settings[4]. More iterations yield better performance, at a cost of an almost-linear increase in time per conformation when clustering is used . If `conformation_comparer` is set to any metric with a tolerance of 0, then no filtering or clustering is applied. This will cause BCL::Conf to perform `max_iterations` conformer generation iterations, randomly select `top_models` conformers, sort them from best to worst by score, and return them. This option is the fastest, and the ensemble returned is sampled by ConfScore without bias from clusteirng.

Generate conformers using two of the described protocols. First, run

```
bcl.exe molecule:ConformerGenerator \
-add_h -neutralize -ensemble_filenames osimertinib.clean.sdf.gz \
-conformers_single_file \
../outputs/osimertinib.symrmsd_cluster.confs.sdf.gz \
-max_iterations 2000 -top_models 25 \
-conformation_comparer SymmetryRMSD 0.25
```

Then,

```
bcl.exe molecule:ConformerGenerator \
-add_h -neutralize -ensemble_filenames osimertinib.clean.sdf.gz \
-conformers_single_file ../outputs/osimertinib.raw.confs.sdf.gz \
-max_iterations 2000 -top_models 250 -skip_clustering \
-conformation_comparer RMSD 0.0
```

Visualize the two ensembles together in PyMOL. You should see that the 25 conformers generated with clustering enabled and the SymmetryRMSD comparer occupy the densest parts of conformational space sampled by the unbiased method with 10x as many conformers.

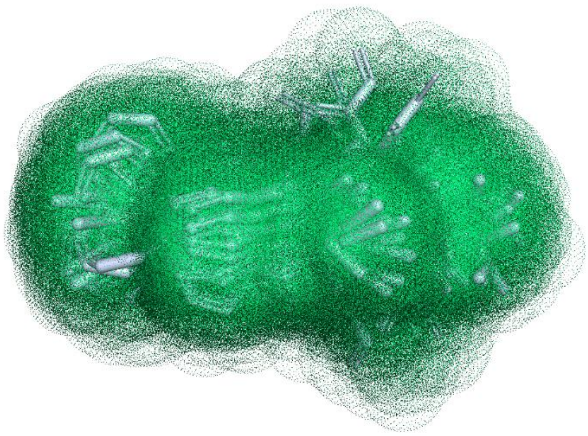

Figure 3. BCL::Conf run with clustering samples densest regions of unbiased conformer sampling simulation. The 250 conformers generated without clustering are depicted as the green point-density cloud. The 25 conformers generated with clustering are depicted as white sticks. Despite generating 10x fewer conformers than the unbiased sampling approach, BCL::Conf with leader-follower clustering fills most of the sampled space well.

Subsection 4B – Local conformational ensembles

Local sampling was implemented in the recent algorithmic improvements to BCL::Conf . The goal was to provide a mechanism for pose refinement in the setting where pre-generated discrete rotamers are used for flexibility.

Local sampling in the BCL is accomplished by restricting the rotamer search in any of three ways:
1. Dihedral bins – restricts search to the input conformer dihedral bins
2. Bond lengths and angles – restricts search to the input conformer bond lengths and angles
3. Rings – restricts search to the input conformer ring conformations

These options are not mutually exclusive. Depending on how they are combined, different levels of sampling can be achieved. Moreover, they can be used in combination with any of the other options (e.g., conformation comparison type, clustering) described above. The flags that control each of these behaviors are `skip_rotamer_dihedral_sampling`, `skip_bond_angle_sampling`, `skip_ring_sampling`, and `skip_clustering`.

I will leave it to the reader as an exercise to try generating conformers with the different combinations described above.

Subsection 4C – Conformational sampling of substructures

One may wish to only sample conformations of part of a molecule. In docking it may be the case that the core scaffold pose may be known with a high degree of confidence from co-crystallized structures of similar compounds. In such cases, you may want to allow conformational sampling of unique dihedrals. Another scenario is one in which you wish to refine a crystal structure of a protein-ligand complex because of low or missing density in part of the bound ligand.

The way that this is accomplished in the BCL is by assigning the MDL property label “SampleByParts” to the molecule(s) of interest. SampleByParts is a list of 0-indexed atom indices for the atoms in dihedrals that are allowed to be sampled by molecule:ConformerGenerator.

Let's follow the example in Brown et al. 2020[1] and sample conformations of just the ethyldimethylamine substituent of osimertinib. We will sample alternative conformations of the ethyldimethylamine substituent than that which is proposed in the reported soaked crystal structure, PDB ID 4ZAU, because the original structure lacked density in this region[8]. First, add the corresponding atom indices to the osimertinib input file.

```
bcl.exe molecule:Properties \
-input_filenames osimertinib.clean.sdf.gz \
-output ../outputs/osimertinib.sample_by_parts.sdf \
-add "Define(SampleByParts=Constant(3,36,18,19,6,20,21))" SampleByParts
```

We have not yet used `molecule:Properties`. This application will be discussed in more detail in the next tutorial. Also note that if you have many molecules for which you want to assign SampleByParts atom indices and you do not want to have to manually identify the relevant indices, you can also use the `molecule:SetSampleByPartsAtoms` application. This application sets SampleByParts indices based on comparison to user-supplied substructures.

Next, generate global conformers as previously described:

```
bcl.exe molecule:ConformerGenerator \
-ensemble_filenames ../outputs/osimertinib.sample_by_parts.sdf.gz \
-conformers_single_file \
.../outputs/osimertinib.sample_by_parts.confs.sdf.gz \
-max_iterations 1000 -top_models 100
```

Visualize the output ensemble. Are the results what you expected? The only dihedrals that are sampled are those that contain the SampleByParts atoms. In this case, only dihedrals containing strictly the ethyldimethylamine atoms are sampled. SampleByParts can also be used in conjunction with the local sampling methods described above.

Part 5: Small molecule comparisons and alignment

This next section deals with molecular comparison and alignment. Very often we are looking to compare the substructure- or property-based similarity of two or more molecules. It is not uncommon that we want to take that comparison a step further and attempt to superimpose two or more molecules such that the similar parts of the molecules overlap in space. This is the basis of 3D pharmacophore modeling and finds much utility elsewhere as well.

In Part 4 Subsection 4A, you may have noticed that the global conformational ensemble is output with all conformers superimposed on a single rigid substructure component, but in the figure with the point cloud the conformers are aligned along a common axis instead. I skipped a step that I used to generate that figure: alignment.

Small molecule alignment is an important component of multiple aspects of drug discovery; practically any time the real space coordinates of a small molecule or its relative coordinate position to another molecule are of importance, some form of alignment can be used to help you with your task.

Alignment can generally be decomposed into two components: (A) sampling, and (B) scoring. Scoring (B) can be conceptualized in two parts: (1) defining what specifically will be compared between the molecules, and (2) defining the metric with which similarity will be measured. In the BCL, this is accomplished primarily through use of the `molecule:Compare` application.

```
bcl.exe molecule:Compare \
<mandatory_parameter_one.sdf> <optional_parameter_two.sdf> \
-output <mandatory_output.file>
```

Note that the syntax is slightly different than some other applications. This syntax uses parameters, which are sequence-dependent arguments, instead of flagged arguments. If a single SDF is specified as a parameter, then the molecules are compared in a pairwise manner. If two SDFs are specified as parameters, then the molecule(s) in the second file will be compared against the molecule(s) in the first file.

Finally, something else we have not discussed so far is the use of multiple threads. Most BCL applications support multiple threads at least for loading in molecules from the command-line. it is worth noting that `molecule:Compare` is notable in that the calculations performed with it benefit substantially from multiple threads.

Not all similarity comparisons occur at the structural/substructural level. Several comparison metrics in the BCL occur between properties computed at the whole molecular, substructural, or atomic level. Further, distance-based comparisons between molecules that are constitutionally identical can also be made.

All right! With all of that in mind, let's walk through an example how you might consider analyzing molecular similarity and performing alignments. For this example, we will revisit our DRD3 antagonists.
First we will perform a maximum common substructure similarity comparison of our molecule of interest, which is a DRD3 antagonist from our training set, to our small set of ~200 molecules that appear in both the DRD3 and DRD5 datasets. This is a pure comparison that does not include or require alignment:

```
bcl.exe molecule:Compare \
DRD3.sample_inhibitor.sdf.gz ../outputs/DRD3-
5.training_molecules.unique_configs.sorted.sdf.gz \
-output ../outputs/DRD3.sample_inhibitor.mcs_tanimoto.txt \
-method "LargestCommonSubstructureTanimoto" \
-scheduler PThread 8
```

For a brief overview of Tanimoto similarity in substructure comparisons, see Brown et al. 2020[1].
The following bash line will convert the output text file into a two-column CSV file where the first column is the 0-indexed position of each molecule in the SD file and the second column is the similarity:

```
tail -n+3 ../outputs/DRD3.sample_inhibitor.mcs_tanimoto.txt | \
awk '{print NR-1,$1}' | tr ' ' ',' > \
../outputs/DRD3.sample_inhibitor.mcs_tanimoto.csv
```

Next we will map those similarity scores back to the ensemble of molecules we are comparing to our chosen antagonist. For this task we will use `molecule:Properties`; however, since we have not covered `molecule:Properties` in detail, I have written a small wrapper script to do it for us:

```
bash ../scripts/AddMappedProperties.sh bcl.exe \
../outputs/DRD3-5.training_molecules.unique_configs.sorted.sdf.gz \
../outputs/DRD3-5.training_molecules.unique_configs.sorted.mcs_tani.sdf.gz\
 ../outputs/DRD3.sample_inhibitor.mcs_tanimoto.csv \
MCS_TANI
```

We now have a new output ensemble that contains the MDL property "MCS_TANI", which is how we have saved the similarity score to the molecules. Let's grab any molecules from that subset that have greater than or equal to 10x higher activity on DRD3 than DRD5 and which have an MCS_TANI of at least 0.25.

```
bcl.exe molecule:Filter \
-input_filenames ../outputs/DRD3-
5.training_molecules.unique_configs.sorted.mcs_tani.sdf.gz \
-compare_property_values \
"Divide(lhs=DRD3_Activity_uM,rhs=DRD5_Activity_uM)" greater_equal 10.0 \
MCS_TANI greater_equal 0.25 \
-output_matched ../outputs/DRD3.mcs_tani_match.sdf.gz \
-add_h -neutralize
```

And convenient for ease, only one molecule passes those criteria. Let's perform a substructure-based alignment of that molecule to our originally chosen inhibitor:

```
bcl.exe molecule:AlignToScaffold \
DRD3.sample_inhibitor.sdf.gz \
../outputs/DRD3.mcs_tani_match.sdf.gz \
../outputs/DRD3.mcs_tani_match.ats.sdf.gz
```

The application `molecule:AlignToScaffold` is deterministic – it matches common substructure atoms to minimize the overall RMSD between matched atom pairs.

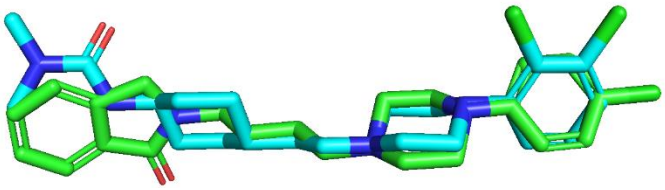

Figure 4. Rigid conformer substructure-based alignment.

This alignment is surprisingly good considering we did not sample any conformers. If we want to incorporate flexibility into the alignment, we can generate conformers with BCL::Conf, align each conformer with `molecule:AlignToScaffold`, and select the best alignment using some type of score. For example, we could use the property-based alignment score we developed for BCL::MolAlign[5] even if we choose to do a substructure-based sampling method.

Nevertheless, this is a few steps to do from the command-line. We have an in-development application (consider it in alpha testing) that will eventually be a one-stop-shop for various alignment procedures. It can perform exactly the procedure I described, so let's give it a whirl:

```
bcl.exe cheminfo:MoleculeFit \
-input_filenames ../outputs/DRD3.mcs_tani_match.ats.sdf.gz \
-output_filename ../outputs/DRD3.mcs_tani_match.fit.sdf \
-scaffold_fragments DRD3.sample_inhibitor.h.sdf.gz \
-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 here is our result:

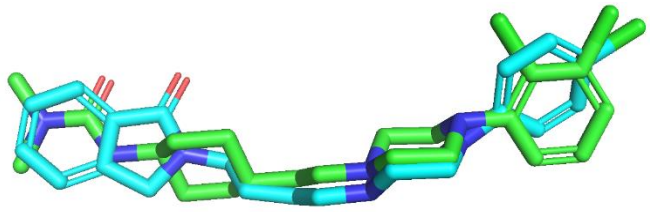

Figure 5. Flexible conformer substructure-based alignment.

This is chemically satisfying because this time the core of the molecule is preserved but we have better orientation of the carbonyl oxygen atoms and the chlorine atoms.

The sampling procedure used in the maximum common substructure-based methods does not consider chemical properties of the atoms, just the mapping identified during the graph comparison. As an alternative approach, we can ignore substructure comparisons and perturb our molecules based on chemical properties. In the BCL, we do this by attempting to minimize the property-distance between two molecules using a Monte Carlo – Metropolis search procedure - the algorithm is called BCL::MolAlign and it is under the PsiFlexField method of `molecule:Compare`.

```
bcl.exe molecule:Compare \
./outputs/DRD3.mcs_tani_match.sdf.gz DRD3.sample_inhibitor.h.sdf.gz \
-output ../outputs/DRD3.mcs_tani_match.mol_align.txt \
-method \
"PsiFlexField(\
output_aligned_mol_a=../outputs/DRD3.mcs_tani_match.mol_align,\
number_flexible_trajectories=1,
number_outputs=5,number_rigid_trajectories=1, \
conformer_pairs=500,
sample_conformers=(conformation_comparer=SymmetryRMSD,\
tolerance=0.25,generate_3D=0,cluster=true,max_iterations=8000,\
max_conformations=500,change_chirality=0))" \
-add_h \
-neutralize
```

This will take 4-5 minutes to run on molecules of this size – the MCM sampling procedure is slow to converge, and we will use many conformers. Note that the scoring used in this approach is identical to the scoring in the previous alignment, thus the two algorithms differ only in their sampling approach. Future development of `cheminfo:MoleculeFit` will allow semi-customizable sampling procedures to be specified on-the-fly.

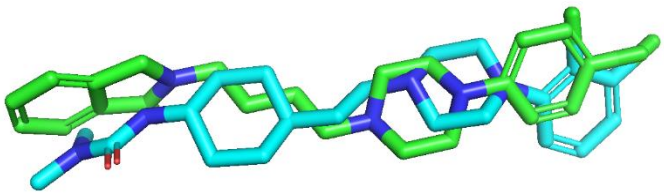

Figure 6. Flexible conformer property-based alignment.

How does your output compare to the previous method? What are similarities in pose? What are differences? Consider aligning a molecule with less selectivity toward DRD3 (i.e., filter such that the activity ratios are near 1). Come up with a hypothesis regarding the determinants of DRD3 selectivity vs. DRD5.

Congratulations! You have finished Tutorial 1. For additional materials related to the topics in this tutorial, check out the references.

References

(1) Brown, B. P.; Vu, O.; Geanes, A. R.; Sandeepkumar, K.; Butkiewicz, M.; Lowe Jr, E. W.; Mueller, R.; Pape, R.; Mendenhall, J.; Meiler, J. Introduction to the BioChemical Library (BCL): An Application-Based Open-Source Toolkit for Integrated Cheminformatics and Machine Learning in Computer-Aided Drug Discovery.

(2) Sliwoski, G.; Kothiwale, S.; Meiler, J.; Lowe, E. W. Computational Methods in Drug Discovery. *Pharmacol. Rev.* **2014**, *66* (1), 334–395. https://doi.org/10.1124/pr.112.007336.

(3) Sliwoski, G.; Mendenhall, J.; Meiler, J. Autocorrelation Descriptor Improvements for QSAR: 2DA_Sign and 3DA_Sign. *J. Comput. Aided Mol. Des.* **2015**. https://doi.org/10.1007/s10822-015-9893-9.

(4) Mendenhall, J.; Brown, B. P.; Kothiwale, S.; Meiler, J. BCL::Conf: Improved Open-Source Knowledge-Based Conformation Sampling Using the Crystallography Open Database. *J. Chem. Inf. Model.* **2020**. https://doi.org/10.1021/acs.jcim.0c01140.

(5) 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.

(6) Friedrich, N.-O.; Meyder, A.; de Bruyn Kops, C.; Sommer, K.; Flachsenberg, F.; Rarey, M.; Kirchmair, J. High-Quality Dataset of Protein-Bound Ligand Conformations and Its Application to Benchmarking Conformer Ensemble Generators. *J. Chem. Inf. Model.* **2017**, *57* (3), 529–539. https://doi.org/10.1021/acs.jcim.6b00613.

(7) Kothiwale, S.; Mendenhall, J. L.; Meiler, J. BCL::Conf: Small Molecule Conformational Sampling Using a Knowledge Based Rotamer Library. *J. Cheminform.* **2015**, *7*, 47. https://doi.org/10.1186/s13321-015-0095-1.

(8) Yosaatmadja, Y.; Silva, S.; Dickson, J. M.; Patterson, A. V.; Smaill, J. B.; Flanagan, J. U.; McKeage, M. J.; Squire, C. J. Binding Mode of the Breakthrough Inhibitor AZD9291 to Epidermal Growth Factor Receptor Revealed. *J. Struct. Biol.* **2015**, *192* (3), 539–544. https://doi.org/10.1016/j.jsb.2015.10.018.