OpenFF Fragmenter Documentation

Chaya D. Stern

Apr 08, 2024

CONTENTS

The main purpose of `openff-fragmenter` is to fragment molecules for quantum chemical (QC) torsion drives.

Warning: `openff-fragmenter` is still pre-alpha. It is not fully tested and the API is still in flux.

Currently two fragmentation schemes are supported:

- a Wiberg Bond Order (WBO) sensitive fragmentation scheme [1] *(recommended)*
- the fragmentation schema detailed by Rai *et al* [2] reffered to in this package as the 'Pfizer' scheme.

CHAPTER

ONE

WBO SENSITIVE FRAGMENTATION

The assumption when fragmenting molecules is that the chemistry is localized and that removing or changing remote substituents (defined as substituents more than 2 bonds away from the central bond that is being driven in the torsion drive) will not change the torsion potential around the bond of interest. However, that is not always the case. `openff-fragmenter` uses the Wiberg Bond Order (WBO) as a surrogate signal to determine if the chemistry around the bond of interest was destroyed during fragmentation relative to the bond in the parent molecule.

The WBO is a measure of electronic population overlap between two atoms in a bond. It can be quickly calculated from an empirical QC calculation and is given by:

$$W_{AB} = \sum_{\mu \in A} \sum_{\nu \in B} |D_{\mu\nu}|^2$$

Where A and B are atoms A and B in a bond, D is the density matrix and μ and ν are occupied orbitals on atoms A and B respectively.

`openff-fragmenter` calculates the WBO of the parent molecules, then fragments according to a set of rules and then recalculates the WBO of the fragments. If the WBO for the fragment of the bond of interest changes more than a user's specified threshold, `openff-fragmenter` will add more substituents until the WBO of the bond of interest is within the user specified threshold.

CHAPTER

TWO

CONTRIBUTORS

- Chaya D. Stern (MSKCC / Weill Cornell)
- John D. Chodera (MSKCC)

CHAPTER

THREE

ACKNOWLEDGMENT

CDS is funded by a fellowship from The Molecular Sciences Software Institute

CHAPTER

FOUR

REFERENCES

4.1 Installation

4.1.1 Installing using conda

The recommended way to install `openff-fragmenter` is via the `conda` package manger:

```
conda install -c conda-forge openff-fragmenter
```

If you do not have Conda installed, see the OpenFF installation guide.

If you have access to the OpenEye toolkits (namely `oechem`, `oequacpac` and `oeomega`) we recommend installing these also as these can speed up fragmentation times significantly:

```
conda install -c openeye openeye-toolkits
```

4.1.2 Installing from source

To install `openff-fragmenter` from source begin by cloning the repository from github:

```
git clone https://github.com/openforcefield/fragmenter.git
cd fragmenter
```

Create a custom conda environment which contains the required dependencies and activate it:

```
conda env create --name fragmenter --file devtools/conda-envs/meta.yaml
conda activate fragmenter
```

Finally, install `openff-fragmenter` itself:

```
python setup.py develop
```

4.2 Fragmenting Molecules

This notebook aims to show how a drug-like molecule can be fragmented using this framework, and how those fragments can be easily visualised using its built-in helper utilities.

To begin with we load in the molecule to be fragmented. Here we load Cobimetinib directly using its SMILES representation using the Open Force Field toolkit:

[1]:
```
from openff.toolkit.topology import Molecule

parent_molecule = Molecule.from_smiles(
    "OC1(CN(C1)C(=O)C1=C(NC2=C(F)C=C(I)C=C2)C(F)=C(F)C=C1)[C@@H]1CCCCN1"
)
```

Next we create the fragmentation engine which will perform the actual fragmentation. Here we will use the recommended `WBOFragmenter` with default options:

[2]:
```
from openff.fragmenter.fragment import WBOFragmenter

frag_engine = WBOFragmenter()
# Export the engine's settings directly to JSON
frag_engine.json()
```

[2]:
```
'{"functional_groups": {"hydrazine": "[NX3:1][NX3:2]", "hydrazone": "[NX3:1][NX2:2]",
↪"nitric_oxide": "[N:1]-[O:2]", "amide": "[#7:1][#6:2](=[#8:3])", "amide_n": "[#7:1][#6:
↪2](-[O-:3])", "amide_2": "[NX3:1][CX3:2](=[OX1:3])[NX3:4]", "aldehyde": "[CX3H1:1](=[O:
↪2])[#6:3]", "sulfoxide_1": "[#16X3:1]=[OX1:2]", "sulfoxide_2": "[#16X3+:1][OX1-:2]",
↪"sulfonyl": "[#16X4:1](=[OX1:2])=[OX1:3]", "sulfinic_acid": "[#16X3:1](=[OX1:2])[OX2H,
↪OX1H0-:3]", "sulfinamide": "[#16X4:1](=[OX1:2])(=[OX1:3])([NX3R0:4])", "sulfonic_acid":
↪ "[#16X4:1](=[OX1:2])(=[OX1:3])[OX2H,OX1H0-:4]", "phosphine_oxide": "[PX4:1](=[OX1:
↪2])([#6:3])([#6:4])([#6:5])", "phosphonate": "[P:1](=[OX1:2])([OX2H,OX1-:3])([OX2H,OX1-
↪:4])", "phosphate": "[PX4:1](=[OX1:2])([#8:3])([#8:4])([#8:5])", "carboxylic_acid":
↪"[CX3:1](=[O:2])[OX1H0-,OX2H1:3]", "nitro_1": "[NX3+:1](=[O:2])[O-:3]", "nitro_2":
↪"[NX3:1](=[O:2])=[O:3]", "ester": "[CX3:1](=[O:2])[OX2H0:3]", "tri_halide": "[#6:1]([F,
↪Cl,I,Br:2])([F,Cl,I,Br:3])([F,Cl,I,Br:4])"}, "scheme": "WBO", "wbo_options": {"method":
↪ "am1-wiberg-elf10", "max_conformers": 800, "rms_threshold": 1.0}, "threshold": 0.03,
↪"heuristic": "path_length", "keep_non_rotor_ring_substituents": false}'
```

Use the engine to fragment the molecule:

[3]:
```
result = frag_engine.fragment(parent_molecule)
# Export the result directly to JSON
result.json()
```

[3]:
```
'{"parent_smiles": "[H:31][c:1]1[c:3]([c:9]([c:11]([c:8]([c:6]1[C:13](=[O:25])[N:23]2[C:
↪18]([C:21]([C:19]2([H:46])[H:47])([C@@:20]3([C:16]([C:14]([C:15]([C:17]([N:22]3[H:
↪49])([H:42])[H:43])([H:38])[H:39])([H:36])[H:37])([H:40])[H:41])[H:48])[O:26][H:
↪51])([H:44])[H:45])[N:24]([H:50])[c:7]4[c:2]([c:4]([c:12]([c:5]([c:10]4[F:28])[H:
↪35])[I:30])[H:34])[H:32])[F:29])[F:27])[H:33]", "fragments": [{"smiles": "[H:34][c:
↪4]1[c:12]([c:5]([c:10]([c:7]([c:2]1[H:32])[N:24]([H:50])[c:8]2[c:6]([c:1]([c:3]([c:
↪9]([c:11]2[F:29])[F:27])[H:33])[H:31])[C:13](=[O:25])[N:23]3[C:18]([C:21]([C:19]3([H:
↪46])[H:47])([H])[H])([H:44])[H:45])[F:28])[H:35])[H]", "bond_indices": [8, 24]}, {
↪"smiles": "[H:48][C@:20]1([C:16]([C:14]([C:15]([C:17]([N:22]1[H:49])([H:42])[H:43])([H:
↪38])[H:39])([H:36])[H:37])([H:40])[H:41])[C:21]2([C:18]([N:23]([C:19]2([H:46])[H:
↪47])[C:13](=[O:25])[C:6]([H])([H])[H])([H:44])[H:45])[O:26][H:51]", "bond_indices":␣
↪[20, 21]}, {"smiles": "[H:34][c:4]1[c:12]([c:5]([c:10]([c:7]([c:2]1[H:32])[H:24])[H:
↪50])[c:8]2[c:6]([c:1]([c:3]([c:9]([c:11]2[F:29])[H])[H:33])[H:31])[C:13](=[O:25])[N:
↪23]3[C:18]([C:21]([C:19]3([H:46])[H:47])([H])[H])([H:44])[H:45])[F:28])[H:35])[H]",
↪"bond_indices": [7, 24]}, {"smiles": "[H:33][c:3]1[c:9]([c:11]([c:8]([c:6]([c:1]1[H:
↪31])[C:13](=[O:25])[N:23]2[C:18]([C:21]([C:19]2([H:46])[H:47])([H])[H])([H:44])[H:
↪45])[N:24]([H:50])[C:7]([H])([H])[H])[H])[H]", "bond_indices": [13, 23]}, {"smiles":
↪"[H:33][c:3]1[c:1]([c:6]([c:8]([c:11]([c:9]1[H])[F:29])[N:24]([H:50])[C:
```

(continued on next page)

(continued from previous page)

Any generated fragments will be returned in a `FragmentationResult` object. We can loop over each of the generated fragments and print both the SMILES representation of the fragment as well as the map indices of the bond that the fragment was built around:

[4]:
```
for fragment in result.fragments:
    print(f"{fragment.bond_indices}: {fragment.smiles}")
```

```
(8, 24): [H:34][c:4]1[c:12]([c:5]([c:10]([c:7]([c:2]1[H:32])[N:24]([H:50])[c:8]2[c:6]([c:
↪1]([c:3]([c:9]([c:11]2[F:29])[F:27])[H:33])[H:31])[C:13](=[O:25])[N:23]3[C:18]([C:
↪21]([C:19]3([H:46])[H:47])([H])[H])([H:44])[H:45])[F:28])[H:35])[H]
(20, 21): [H:48][C@:20]1([C:16]([C:14]([C:15]([C:17]([N:22]1[H:49])([H:42])[H:43])([H:
↪38])[H:39])([H:36])[H:37])([H:40])[H:41])[C:21]2([C:18]([N:23]([C:19]2([H:46])[H:
↪47])[C:13](=[O:25])[C:6]([H])([H])[H])([H:44])[H:45])[O:26][H:51]
(7, 24): [H:34][c:4]1[c:12]([c:5]([c:10]([c:7]([c:2]1[H:32])[N:24]([H:50])[c:8]2[c:6]([c:
↪1]([c:3]([c:9]([c:11]2[F:29])[H])[H:33])[H:31])[C:13](=[O:25])[N:23]3[C:18]([C:21]([C:
↪19]3([H:46])[H:47])([H])[H])([H:44])[H:45])[F:28])[H:35])[H]
(13, 23): [H:33][c:3]1[c:9]([c:11]([c:8]([c:6]([c:1]1[H:31])[C:13](=[O:25])[N:23]2[C:
↪18]([C:21]([C:19]2([H:46])[H:47])([H])[H])([H:44])[H:45])[N:24]([H:50])[C:
↪7]([H])([H])[H])[H])[H]
(6, 13): [H:33][c:3]1[c:1]([c:6]([c:8]([c:11]([c:9]1[H])[F:29])[N:24]([H:50])[C:
↪7]([H])([H])[H])[C:13](=[O:25])[N:23]2[C:18]([C:21]([C:19]2([H:46])[H:47])([H])[H])([H:
↪44])[H:45])[H:31]
```

Finally, we can visualize the produced fragments:

[5]:
```
from openff.fragmenter.depiction import depict_fragmentation_result

depict_fragmentation_result(result=result, output_file="example_fragments.html")

from IPython.core.display import HTML

with open("example_fragments.html") as file:
    display(HTML(file.read()))
```

```
<IPython.core.display.HTML object>
```

4.3 API

Below is an outline of the API for the main functions of `openff-fragmenter` See the examples for details on how to use these objects.

Warning: The `openff-fragmenter` package is still pre-alpha so the API is still in flux.

4.3.1 Fragmentation Engines

WBO

Pfizer

4.3.2 Fragmentation Outputs

BIBLIOGRAPHY

[1] Chaya D Stern, Christopher I Bayly, Daniel G A Smith, Josh Fass, Lee-Ping Wang, David L Mobley, and John D Chodera. Capturing non-local through-bond effects when fragmenting molecules for quantum chemical torsion scans. *bioRxiv*, 2020. URL: https://www.biorxiv.org/content/early/2020/08/28/2020.08.27.270934, arXiv:https://www.biorxiv.org/content/early/2020/08/28/2020.08.27.270934.full.pdf, doi:10.1101/2020.08.27.270934.

[2] Brajesh K. Rai, Vishnu Sresht, Qingyi Yang, Ray Unwalla, Meihua Tu, Alan M. Mathiowetz, and Gregory A. Bakken. Comprehensive assessment of torsional strain in crystal structures of small molecules and protein–ligand complexes using ab initio calculations. *Journal of Chemical Information and Modeling*, 59(10):4195–4208, 2019. PMID: 31573196. URL: https://doi.org/10.1021/acs.jcim.9b00373, arXiv:https://doi.org/10.1021/acs.jcim.9b00373, doi:10.1021/acs.jcim.9b00373.