moffragmentor

Release 0.0.6

Kevin Maik Jablonka

Dec 24, 2022

CONTENTS

1 Contents **3**
- 1.1 Getting started with moffragmentor . . . 3
- 1.2 Background information . . . 6
- 1.3 API documentation . . . 7

2 Indices and tables **19**

Bibliography **21**

Python Module Index **23**

Index **25**

MOFfragmentor

The moffragmentor package gives utilities to interact with reticular building blocks in an object-oriented way.

CHAPTER

ONE

CONTENTS

1.1 Getting started with moffragmentor

1.1.1 Installation

We recommend installing moffragmentor in a clean virtual environment (e.g., a conda environment)

You can install the latest stable release from PyPi using

```
pip install moffragmentor
```

or the latest development version using

```
pip install git+https://github.com/kjappelbaum/moffragmentor.git
```

Note: If you install via pip you will need to manually install openbabel (e.g. *conda install openbabel -c conda-forge*).

If you want to determine RCSR codes, you will also need to install a Java runtime environment (JRE) of version 1.5.0 or later on your machine as we use the Systre (Symmetry, Structure (Recognition) and Refinement) code to perform analysis of the labeled quotient graphs we construct in moffragmentor.

Extensions

In case you want to use the `show_molecule()`, or *`show_structure()`* function in Jupyter lab you have to

```
jupyter-labextension install @jupyter-widgets/jupyterlab-manager
jupyter-labextension install nglview-js-widgets
jupyter-nbextension enable nglview --py --sys-prefix
```

You also might find the debugging help in the nglview documentation useful.

1.1.2 Fragmenting a MOF

To fragment a MOF you need to create an instance of *MOF* and then call *fragment()*.

```
from moffragmentor import MOF

mof = MOF.from_cif(<my_cif.cif>)
fragmentation_result = mof.fragment()
```

The result is a `FragmentationResult namedtuple` with the fields `nodes`, `linkers`, both subclasses of a `moffragmentor.sbu.SBUCollection` and `bound_solvent`, `unbound_solvent`, both `moffragmentor.molecule.NonSbuMoleculeCollection`, and a `moffragmentor.net.Net`.

Warning: If you use the `MOF.from_cif` method, we will run `pymatgen.analysis.spacegroup.SpacegroupAnalyzer` on the input structure. This might take some time, and we also have encountered cases where it can be really slow. If you do not want this, you can either "manually" call the constructor or tune the tolerance parameters.

Warning: Note that moffragmentor currently does not automatically delete bound solvent. This is due to two observations:

1. We have very little understanding of what solvent we can remove without affecting the structural integrity.
2. We (currently) do not have a way to estimate if a solvent is charged. We explore different implementation strategies, but we do not have a robust one at this moment.

You might want a quick overview of the composition of the different components. You can access this via the `composition` properties

```
solvent_collection.composition
```

which will return a dictionary of the counts of the compositions, for example `{'C3 H7 N1 O1': 3, 'H2 O1': 4}`.

Clearly, we do not consider floating solvent for the computation of the net.

Known issues

For some structures in the CSD MOF subset, there will be problems with the fragmentation. One example is `CAYSIE`, which is a metalloporphyrinate. Here, the code struggles to distinguish nodes and linkers as a core routine of the moffragmentor is to check if a metal atom is inside another, potential linker, molecule.

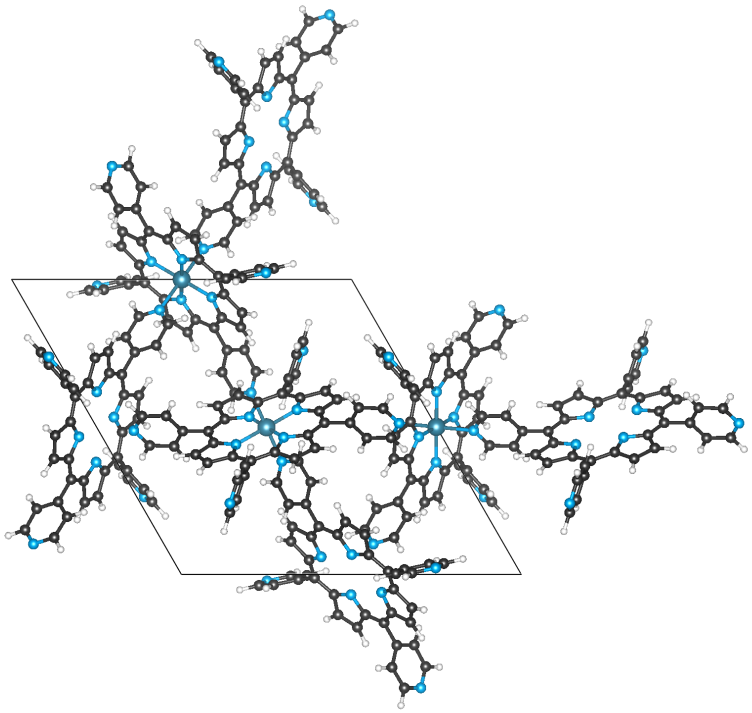

Fig. 1.1: Example of a metalloporphyrinate for which the fragmentor fails.

Also note that there are problems with analyzing the combinatorial topology of 1D rod MOFs. There only recently has been an algorithm proposed that is implemented in ToposPro.

Customizing the log level

moffragmentor uses the loguru for logging. By default, logging from moffragmentor is disabled to not interfere with your logs.

However, you can easily customize the logging:

```
import sys
from loguru import logger

# enable moffragmentor logging
logger.enable("moffragmentor")

# define the logging level
LEVEL = "INFO || DEBUG || WARNING || etc."

# set the handler
# for logging to stdout
logger.add(sys.stdout, level=LEVEL)
# or for logging to a file
logger.add("my_log_file.log", level=LEVEL, enqueue=True)
```

In many cases, however, you might find it convenient to simply call *enable_logging()*

```
from moffragmentor.utils import enable_logging

enable_logging()
```

which will enable logging with sane defaults (i.e. logging to `stderr` for `INFO` and `WARNING` levels).

1.2 Background information

1.2.1 Fragmentation

For the fragmentation of a MOF structure we rely on a structure graph. In moffragmentor we use heuristics in pymatgen to construct the structure graph (which is a networkx multigraph with annotation about the periodic neighbors). If the structure graph does not contain a pair of bonds, moffragmentor cannot consider it in the fragmentation.

Warning: The current implementation of the fragementation is in parts relatively inefficient as we perform multiple traversals of the structure graph.

For the fragmentation, there are a few definitions we have to make:

Bridge

In a graph, a bridge is an edge, if removed, would increase the number of connected components.

Bound solvent

A bound solvent molecule is bound via a bridge edge to one metal. According to this definition, M-OCH_3, M-OH_2, etc. are all bound solvents, whereas a bridging formate is not.

Floating solvent

Floating solvent is an isolated connected component in the structure graph that does not lie across periodic boundaries in a supercell.

Branching site

- has at minimum coordination number 3
- at least one path with maximum 2 edges that leads to metal and does not contain a bride
- **has at minimum 2 non-metal connections that are not bridges**
  (note that in case the connection to a metal goes via 2 edges, then the first node of this path to the metal contributes to this count)

If there are multiple neighboring sites selecting according this definition, we pick the one closest to the metal (the fewest number of edges).

Capping site

A capping site is part of a cycle with one of more metals that does not contain a branching site.

For the fragmentation branching sites define the places at which we make the split between node and linker. The fragmentation algorithm goes through the following steps:

1. Extracting floating solvent.
2. From metal sites perform depth-first search on the structure graph up to a branching site.
3. **"Complete graph" by traversing the graph from all non-branching sites traversed in step 1** up to a leaf node.
4. Extracting nodes as separate connected components.
5. Deleting nodes from the structure graph and extracting linkers as connected components.

1.2.2 SBU dimensionality

For many applications, the dimensionality of the SBUs can be of interest [Rosi2005]. For example, one can hypothesize that 1D nodes can have favorable charge conductance properties. Also, such rod SBUs may prevent interpenetration [Rosi2005].

To compute the dimensionality of the building blocks we use the algorithm proposed by Larsen et al. [Larsen2019].

1.2.3 Net Embedding

A key concept in reticular chemistry is the one of the net. Computing the topology of the net embedding is not entirely trivial as there is no specific rule of clusters of atoms should be condensed to a vertex [Bureekaew2015] (for example, one might place vertices on subfragments of large linkers. In moffragmentor, we use the centers of node and linker clusters as vertices. Using the Systre code [DelagoFriedrichs2003], we can then determine the RCSR code of this net.

1.2.4 References

1.3 API documentation

1.3.1 Core API

Most users will only need to deal with the `MOF` class.

Defining the main representation of a MOF.

class moffragmentor.mof.**MOF**(*structure*, *structure_graph*)

Main representation for a MOF structure.

This container holds a structure and its associated graph. It also provides some convenience methods for getting neighbors or results of the fragmentation.

Internally, this code typically uses IStructure objects to avoid bugs due to the mutability of Structure objects (e.g. the fragmentation code performs operations on the structure and we want to be sure that there is no impact on the input).

Examples

```
>>> from moffragmentor import MOF
>>> mof = MOF(structure, structure_graph)
>>> # equivalent is to read from a cif file
>>> mof = MOF.from_cif(cif_file)
>>> # visualize the structure
>>> mof.show_structure()
>>> # get the neighbors of a site
>>> mof.get_neighbor_indices(0)
>>> # perform fragmentation
>>> fragments mof.fragment()
```

property bridges: Dict[int, int]

Get a dictionary of bridges.

Bridges are edges in a graph that, if deleted, increase the number of connected components.

Returns
dictionary of bridges

Return type
Dict[int, int]

dump(*path*)

Dump this object as pickle file

Return type
None

property frac_coords: ndarray

Return fractional coordinates of the structure.

We cache this call as pymatgen seems to re-compute this.

Returns

fractional coordinates of the structure
in array of shape (n_sites, 3)

Return type
np.ndarray

fragment(*check_dimensionality=True*, *create_single_metal_bus=False*, *break_organic_nodes_at_metal=True*)

Split the MOF into building blocks.

The building blocks are linkers, nodes, bound, unbound solvent, net embedding of those building blocks.

Parameters

- **check_dimensionality** (*bool*) – Check if the node is 0D. If not, split into isolated metals. Defaults to True.
- **create_single_metal_bus** (*bool*) – Create a single metal BUs. Defaults to False.
- **break_organic_nodes_at_metal** (*bool*) – Break nodes into single metal BU if they appear "too organic".

Returns
FragmentationResult object.

Return type
FragmentationResult

classmethod from_cif(*cif*, *symprec=None*, *angle_tolerance=None*, *get_primitive=True*)

Initialize a MOF object from a cif file.

Note that this method, by default, symmetrizes the structure.

Parameters

- **cif** (*str*) – path to the cif file
- **symprec** (*float, optional*) – Symmetry precision
- **angle_tolerance** (*float, optional*) – Angle tolerance
- **get_primitive** (*bool*) – Whether to get the primitive cell

Returns
MOF object

Return type
MOF

get_neighbor_indices(*site*)

Get list of indices of neighboring sites.

Return type
List[int]

get_symbol_of_site(*site*)

Get elemental symbol of site indexed site.

Return type
str

property nx_graph

Structure graph as networkx graph object

show_adjacency_matrix(*highlight_metals=False*)

Plot structure graph as adjaceny matrix

show_structure()

Visualize structure using nglview.

property terminal_indices: List[int]

Return the indices of the terminal sites.

A terminal site is a site that has only one neighbor. And is connected via a bridge to the rest of the structure. That means, splitting the bond between the terminal site and the rest of the structure will increase the number of connected components.

Typical examples of terminal sites are hydrogren atoms, or halogen functional groups.

Returns
indices of the terminal sites

Return type
List[int]

write_cif(*filename*)

Write the structure to a CIF file.

Return type
None

1.3.2 Command line interface

Command line interfaces

1.3.3 SBU subpackage

Defines datastructures for the building blocks as well as collections of building blocks

Representation for a secondary building block.

class moffragmentor.sbu.sbu.**SBU**(*molecule*, *molecule_graph*, *graph_branching_indices*, *binding_indices*, *molecule_original_indices_mapping=None*, *dummy_molecule=None*, *dummy_molecule_graph=None*, *dummy_molecule_indices_mapping=None*, *dummy_branching_indices=None*)

Representation for a secondary building block.

It also acts as container for site indices:

- **graph_branching_indices: are the branching indices according**
  to the graph-based definition. They might not be part of the molecule.
- **binding_indices: are the indices of the sites between**
  the branching index and metal
- **original_indices: complete original set of indices that has been selected**
  for this building blocks

Note: The coordinates in the molecule object are not the ones directly extracted from the MOF. They are the coordinates of sites unwrapped to ensure that there are no "broken molecules" .

To obtain the "original" coordinates, use the *_coordinates* attribute.

Note: Dummy molecules

In dummy molecules the binding and branching sites are replaces by dummy atoms (noble gas). They also have special properties that indicate the original species.

Examples

```
>>> # visualize the molecule
>>> sbu_object.show_molecule()
>>> # search pubchem for the molecule
>>> sbu_object.search_pubchem()
```

get_neighbor_indices(*site*)

Get list of indices of neighboring sites

Return type
List[int]

property hash: str

Return hash.

The hash is a combination of Weisfeiler-Lehman graph hash and center.

Returns
Hash.

Return type
str

search_pubchem(*listkey_counts=10*, ***kwargs*)

Search for a molecule in pubchem # noqa: DAR401

Second element of return tuple is true if there was an identity match

Parameters

- **listkey_counts** (*int*) – Number of list keys to return (relevant for substructure search). Defaults to 10.
- **kwargs** – Additional arguments to pass to PubChem.search

Returns
List of pubchem ids and whether there was an identity match

Return type
Tuple[List[str], bool]

property smiles: str

Return canonical SMILES.

Use openbabel to compute the SMILES, but then get the canonical version with RDKit as we observed sometimes the same molecule ends up as different canonical SMILES for openbabel. If RDKit cannot make a canonical SMILES (can happen with organometallics) we simply use the openbabel version.

Returns
Canonical SMILES

Return type
str

Collection for MOF building blocks

class moffragmentor.sbu.sbucollection.SBUCollection(*sbus*)

Container for a collection of SBUs

property smiles

Return a list of the SMILES strings of the SBUs.

Returns
A list of smiles strings.

Return type
List[str]

Create Python containers for node building blocks.

Here we understand metal clusters as nodes.

class moffragmentor.sbu.node.**Node**(*molecule*, *molecule_graph*, *graph_branching_indices*, *binding_indices*, *molecule_original_indices_mapping=None*, *dummy_molecule=None*, *dummy_molecule_graph=None*, *dummy_molecule_indices_mapping=None*, *dummy_branching_indices=None*)

Container for metal cluster building blocks.

Will typically automatically be constructured by the fragmentor.

classmethod from_mof_and_indices(*mof*, *node_indices*, *branching_indices*, *binding_indices*)

Build a node object from a MOF and some intermediate outputs of the fragmentation.

Parameters

- **mof** (MOF) – The MOF to build the node from.
- **node_indices** (*Set[int]*) – The indices of the nodes in the MOF.
- **branching_indices** (*Set[int]*) – The indices of the branching points in the MOF that belong to this node.
- **binding_indices** (*Set[int]*) – The indices of the binding points in the MOF that belong to this node.

Returns
A node object.

Describing a collection of nodes

class moffragmentor.sbu.nodecollection.**NodeCollection**(*sbus*)

Collection of node building blocks

Describing the organic building blocks, i.e., linkers.

class moffragmentor.sbu.linker.**Linker**(*molecule*, *molecule_graph*, *graph_branching_indices*, *binding_indices*, *molecule_original_indices_mapping=None*, *dummy_molecule=None*, *dummy_molecule_graph=None*, *dummy_molecule_indices_mapping=None*, *dummy_branching_indices=None*)

Describe a linker in a MOF

Describing an collection of linkers

class moffragmentor.sbu.linkercollection.**LinkerCollection**(*sbus*)

Collection of linker building blocks

property building_block_composition: List[str]

Return a list of strings of building blocks.

Strings are of the form of L{i} where i is an integer.

Returns
List of strings of building blocks.

Return type
List[str]

1.3.4 molecule subpackage

Defines datastructures for the non-building-block molecules (e.g. solvent) as well as collections of such molecules

Dealing with molecules that not part of a secondary building unit.

class moffragmentor.molecule.nonsbumolecule.**NonSbuMolecule**(*molecule*, *molecule_graph*, *indices*, *connecting_index=None*)

Class to handle solvent or other non-SBU molecules.

classmethod from_structure_graph_and_indices(*structure_graph*, *indices*)

Create a a new NonSbuMolecule from a part of a structure graph.

Parameters

- **structure_graph** (*StructureGraph*) – Structure graph with structure attribute
- **indices** (*List[int]*) – Indices that label nodes in the structure graph, indexing the molecule of interest

Returns
Instance of NonSbuMolecule

Return type
NonSbuMolecule

show_molecule()

Use nglview to show the molecule.

Collections of molecules, e.g. bound solvents and non-bound solvents.

class moffragmentor.molecule.nonsbumoleculecollection.**NonSbuMoleculeCollection**(*non_sbu_molecules*)

Class to handle collections of molecules.

For example, bound solvents and non-bound solvents.

property composition: str

Get a string describing the composition.

Return type
str

1.3.5 Fragmentor subpackage

This subpackage is not optimized for end-users. It is intended for developers who wish to customize the behavior of the fragmentor.

Routines for finding branching points in a structure graph of a MOF.

Note that those routines do not work for other reticular materials as they assume the presence of a metal.

moffragmentor.fragmentor.branching_points.**get_branch_points**(*mof*)

Get all branching points in the MOF.

Parameters
mof (MOF) – MOF object.

Returns
List of indices of branching points.

Return type
List[int]

Some pure functions that are used to perform the node identification.

Node classification techniques described in https://pubs.acs.org/doi/pdf/10.1021/acs.cgd.8b00126.

class moffragmentor.fragmentor.nodelocator.**NodelocationResult**(*nodes*, *branching_indices*, *connecting_paths*, *binding_indices*, *to_terminal_from_branching*)

binding_indices

Alias for field number 3

branching_indices

Alias for field number 1

connecting_paths

Alias for field number 2

nodes

Alias for field number 0

to_terminal_from_branching

Alias for field number 4

moffragmentor.fragmentor.nodelocator.**find_node_clusters**(*mof*, *unbound_solvent_indices=None*, *forbidden_indices=None*)

Locate the branching indices, and node clusters in MOFs.

Starting from the metal indices it performs depth first search on the structure graph up to branching points.

Parameters

- **mof** (MOF) – moffragmentor MOF instance
- **unbound_solvent_indices** (*List[int], optionl*) – indices of unbound solvent atoms. Defaults to None.
- **forbidden_indices** (*List[int], optional*) – indices not considered as metals, for instance, because they are part of a linker. Defaults to None.

Returns

nametuple with the slots "nodes", "branching_indices" and "connecting_paths"

Return type
NodelocationResult

Based on the node location, locate the linkers

moffragmentor.fragmentor.linkerlocator.**create_linker_collection**(*mof*, *node_location_result*, *node_collection*, *unbound_solvents*, *bound_solvents*)

Based on MOF, node locaion and unbound solvent location locate the linkers

Return type
Tuple[*LinkerCollection*, dict]

Funnctions that can be used to locate bound and unbound sovlent

moffragmentor.fragmentor.solventlocator.**get_all_bound_solvent_molecules**(*mof*, *node_atom_sets*)

Identify all bound solvent molecules.

Bound solvent is defined as being connected via one bridge to one metal center.

Parameters

- **mof** (MOF) – instance of a MOF object
- **node_atom_sets** (*List[Set[int]]*) – List of indices for the MOF nodes

Returns

Collection of NonSbuMolecule objects
containing the bound solvent molecules

Return type
NonSbuMoleculeCollection

moffragmentor.fragmentor.solventlocator.**get_floating_solvent_molecules**(*mof*)

Create a collection of NonSbuMolecules from a MOF.

Parameters
mof (MOF) – instance of MOF

Returns
collection of NonSbuMolecules

Return type
NonSbuMoleculeCollection

Extraction of pymatgen Molecules from a structure for which we know the branching points.

This module contains functions that perform filtering on indices or fragments.

Those fragments are typically obtained from the other fragmentation modules.

moffragmentor.fragmentor.filter.**bridges_across_cell**(*mof*, *indices*)

Check if a molecule of indices bridges across the cell

Return type
bool

moffragmentor.fragmentor.filter.**in_hull**(*pointcloud*, *hull*)

Test if points in *p* are in *hull*.

Taken from https://stackoverflow.com/a/16898636

Parameters

- **pointcloud** (*np.array*) – points to test (*NxK* coordinates of *N* points in *K* dimensions)
- **hull** (*np.array*) – Is either a scipy.spatial.Delaunay object or the *MxK* array of the coordinates of *M* points in *K* dimensions for which Delaunay triangulation will be computed

Returns
True if all points are in the hull, False otherwise

Return type
bool

Generate molecules as the subgraphs from graphs

moffragmentor.fragmentor.molfromgraph.**wrap_molecule**(*mol_idxs*, *mof*, *starting_index=None*, *add_additional_site=True*)

Wrap a molecule in the cell of the MOF by walking along the structure graph.

For this we perform BFS from the starting index. That is, we use a queue to keep track of the indices of the atoms that we still need to visit (the neighbors of the current index). We then compute new coordinates by computing the Cartesian coordinates of the neighbor image closest to the new coordinates of the current atom.

To then create a Molecule with the correct ordering of sites, we walk through the hash table in the order of the original indices.

Parameters

- **mol_idxs** (*Iterable[int]*) – The indices of the atoms in the molecule in the MOF.
- **mof** (MOF) – MOF object that contains the mol_idxs.
- **starting_index** (*int, optional*) – Starting index for the walk. Defaults to 0.
- **add_additional_site** (*bool*) – Whether to add an additional site

Returns
wrapped molecule

Return type
Molecule

1.3.6 Utils subpackage

Also the `utils` subpackage is not optimized for end-users.

Helper functions.

class moffragmentor.utils.**IStructure**(*lattice*, *species*, *coords*, *charge=None*, *validate_proximity=False*, *to_unit_cell=False*, *coords_are_cartesian=False*, *site_properties=None*)

pymatgen IStructure with faster equality comparison.

This dramatically speeds up lookups in the LRU cache when an object with the same __hash__ is already in the cache.

moffragmentor.utils.**enable_logging**()

Set up the mofdscribe logging with sane defaults.

Return type
List[int]

moffragmentor.utils.**get_sub_structure**(*mof*, *indices*)

Return a sub-structure of the structure with only the sites with the given indices.

Parameters

- **mof** (MOF) – MOF object
- **indices** (*Collection[int]*) – Collection of integers

Returns
sub-structure of the structure with only the sites with the given indices

Return type
Structure

moffragmentor.utils.**is_tool**(*name*)

Check whether *name* is on PATH and marked as executable.

https://stackoverflow.com/questions/11210104/check-if-a-program-exists-from-a-python-script

Parameters
name (*str*) – The name of the tool to check for.

Returns
True if the tool is on PATH and marked as executable.

Return type
bool

Errors reused across the moffragmentor package

exception moffragmentor.utils.errors.**JavaNotFoundError**

Raised if Java executable could not be found

exception moffragmentor.utils.errors.**NoMetalError**

Raised if structure contains no metal

Methods to rank molecules according to some measure of similarity.

moffragmentor.utils.mol_compare.**mcs_rank**(*smiles_reference*, *smiles*, *additional_attributes=None*)

Rank SMILES based on the maximum common substructure to the reference smiles.

moffragmentor.utils.mol_compare.**tanimoto_rank**(*smiles_reference*, *smiles*, *additional_attributes=None*)

Rank SMILES based on the Tanimoto similarity to the reference smiles.

Methods on structure graphs

Methods for running systre

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[Rosi2005] Rosi, N. L. et al. Rod Packings and MetalOrganic Frameworks Constructed from Rod-Shaped Secondary Building Units. J. Am. Chem. Soc. 127, 1504–1518 (2005).

[Larsen2019] Larsen, P. M., Pandey, M., Strange, M. & Jacobsen, K. W. Definition of a scoring parameter to identify low-dimensional materials components. Phys. Rev. Materials 3, (2019).

[Bureekaew2015] Bureekaew, S., Balwani, V., Amirjalayer, S. & Schmid, R. Isoreticular isomerism in 4,4-connected paddle-wheel metal–organic frameworks: structural prediction by the reverse topological approach. CrystEngComm 17, 344–352 (2015).

[DelagoFriedrichs2003] Delgado-Friedrichs, O. & O'Keeffe, M. Identification of and symmetry computation for crystal nets. Acta Cryst Sect A 59, 351–360 (2003).

PYTHON MODULE INDEX

m

moffragmentor.cli, 10
moffragmentor.fragmentor.branching_points, 13
moffragmentor.fragmentor.filter, 15
moffragmentor.fragmentor.linkerlocator, 14
moffragmentor.fragmentor.molfromgraph, 15
moffragmentor.fragmentor.nodelocator, 14
moffragmentor.fragmentor.solventlocator, 14
moffragmentor.fragmentor.splitter, 15
moffragmentor.mof, 7
moffragmentor.molecule.nonsbumolecule, 13
moffragmentor.molecule.nonsbumoleculecollection, 13
moffragmentor.sbu.linker, 12
moffragmentor.sbu.linkercollection, 12
moffragmentor.sbu.node, 11
moffragmentor.sbu.nodecollection, 12
moffragmentor.sbu.sbu, 10
moffragmentor.sbu.sbucollection, 11
moffragmentor.utils, 16
moffragmentor.utils.errors, 17
moffragmentor.utils.mol_compare, 17
moffragmentor.utils.periodic_graph, 17
moffragmentor.utils.systre, 17

INDEX

B

binding_indices (*moffragmentor.fragmentor.nodelocator.NodelocationResult attribute*), 14
branching_indices (*moffragmentor.fragmentor.nodelocator.NodelocationResult attribute*), 14
bridges (*moffragmentor.mof.MOF property*), 8
bridges_across_cell() (*in module moffragmentor.fragmentor.filter*), 15
building_block_composition (*moffragmentor.sbu.linkercollection.LinkerCollection property*), 12

C

composition (*moffragmentor.molecule.nonsbumoleculecollection.NonSbuMoleculeCollection property*), 13
connecting_paths (*moffragmentor.fragmentor.nodelocator.NodelocationResult attribute*), 14
create_linker_collection() (*in module moffragmentor.fragmentor.linkerlocator*), 14

D

dump() (*moffragmentor.mof.MOF method*), 8

E

enable_logging() (*in module moffragmentor.utils*), 16

F

find_node_clusters() (*in module moffragmentor.fragmentor.nodelocator*), 14
frac_coords (*moffragmentor.mof.MOF property*), 8
fragment() (*moffragmentor.mof.MOF method*), 8
from_cif() (*moffragmentor.mof.MOF class method*), 9
from_mof_and_indices() (*moffragmentor.sbu.node.Node class method*), 12
from_structure_graph_and_indices() (*moffragmentor.molecule.nonsbumolecule.NonSbuMolecule class method*), 13

G

get_all_bound_solvent_molecules() (*in module moffragmentor.fragmentor.solventlocator*), 14
get_branch_points() (*in module moffragmentor.fragmentor.branching_points*), 13
get_floating_solvent_molecules() (*in module moffragmentor.fragmentor.solventlocator*), 15
get_neighbor_indices() (*moffragmentor.mof.MOF method*), 9
get_neighbor_indices() (*moffragmentor.sbu.sbu.SBU method*), 10
get_sub_structure() (*in module moffragmentor.utils*), 16
get_symbol_of_site() (*moffragmentor.mof.MOF method*), 9

H

hash (*moffragmentor.sbu.sbu.SBU property*), 11

I

in_hull() (*in module moffragmentor.fragmentor.filter*), 15
is_tool() (*in module moffragmentor.utils*), 16
IStructure (*class in moffragmentor.utils*), 16

J

JavaNotFoundError, 17

L

Linker (*class in moffragmentor.sbu.linker*), 12
LinkerCollection (*class in moffragmentor.sbu.linkercollection*), 12

M

mcs_rank() (*in module moffragmentor.utils.mol_compare*), 17
module
    moffragmentor.cli, 10
    moffragmentor.fragmentor.branching_points, 13
    moffragmentor.fragmentor.filter, 15

moffragmentor.fragmentor.linkerlocator, 14
moffragmentor.fragmentor.molfromgraph, 15
moffragmentor.fragmentor.nodelocator, 14
moffragmentor.fragmentor.solventlocator, 14
moffragmentor.fragmentor.splitter, 15
moffragmentor.mof, 7
moffragmentor.molecule.nonsbumolecule, 13
moffragmentor.molecule.nonsbumoleculecollection, 13
moffragmentor.sbu.linker, 12
moffragmentor.sbu.linkercollection, 12
moffragmentor.sbu.node, 11
moffragmentor.sbu.nodecollection, 12
moffragmentor.sbu.sbu, 10
moffragmentor.sbu.sbucollection, 11
moffragmentor.utils, 16
moffragmentor.utils.errors, 17
moffragmentor.utils.mol_compare, 17
moffragmentor.utils.periodic_graph, 17
moffragmentor.utils.systre, 17
MOF (*class in moffragmentor.mof*), 7
moffragmentor.cli
module, 10
moffragmentor.fragmentor.branching_points
module, 13
moffragmentor.fragmentor.filter
module, 15
moffragmentor.fragmentor.linkerlocator
module, 14
moffragmentor.fragmentor.molfromgraph
module, 15
moffragmentor.fragmentor.nodelocator
module, 14
moffragmentor.fragmentor.solventlocator
module, 14
moffragmentor.fragmentor.splitter
module, 15
moffragmentor.mof
module, 7
moffragmentor.molecule.nonsbumolecule
module, 13
moffragmentor.molecule.nonsbumoleculecollection
module, 13
moffragmentor.sbu.linker
module, 12
moffragmentor.sbu.linkercollection
module, 12
moffragmentor.sbu.node
module, 11
moffragmentor.sbu.nodecollection
module, 12
moffragmentor.sbu.sbu
module, 10
moffragmentor.sbu.sbucollection
module, 11
moffragmentor.utils
module, 16
moffragmentor.utils.errors
module, 17
moffragmentor.utils.mol_compare
module, 17
moffragmentor.utils.periodic_graph
module, 17
moffragmentor.utils.systre
module, 17

N

Node (*class in moffragmentor.sbu.node*), 11
NodeCollection (*class in moffragmentor.sbu.nodecollection*), 12
NodelocationResult (*class in moffragmentor.fragmentor.nodelocator*), 14
nodes (*moffragmentor.fragmentor.nodelocator.NodelocationResult attribute*), 14
NoMetalError, 17
NonSbuMolecule (*class in moffragmentor.molecule.nonsbumolecule*), 13
NonSbuMoleculeCollection (*class in moffragmentor.molecule.nonsbumoleculecollection*), 13
nx_graph (*moffragmentor.mof.MOF property*), 9

S

SBU (*class in moffragmentor.sbu.sbu*), 10
SBUCollection (*class in moffragmentor.sbu.sbucollection*), 11
search_pubchem() (*moffragmentor.sbu.sbu.SBU method*), 11
show_adjacency_matrix() (*moffragmentor.mof.MOF method*), 9
show_molecule() (*moffragmentor.molecule.nonsbumolecule.NonSbuMolecule method*), 13
show_structure() (*moffragmentor.mof.MOF method*), 9
smiles (*moffragmentor.sbu.sbu.SBU property*), 11
smiles (*moffragmentor.sbu.sbucollection.SBUCollection property*), 11

T

tanimoto_rank() (*in module moffragmentor.utils.mol_compare*), 17
terminal_indices (*moffragmentor.mof.MOF property*), 9
to_terminal_from_branching (*moffragmentor.fragmentor.nodelocator.NodelocationResult attribute*), 14

W

wrap_molecule() (*in module moffragmentor.fragmentor.molfromgraph*), 15
write_cif() (*moffragmentor.mof.MOF method*), 9