Cocktail Shaker Documentation

Release 1.0.0

Suliman Sharif

Aug 17, 2020

CONTENTS:

CHAPTER

ONE

COCKTAIL SHAKER

Cocktail Shaker is a **high-performance drug enumeration and expansion library**. Cocktail Shaker leverages the computational power of **RDKit** to create and enumerate large volumes of drug compounds.

```
>>> from cocktail_shaker import Cocktail, FileWriter
>>> from cocktail_shaker import PeptideBuilder
>>> peptide_backbone = PeptideBuilder(2)
>>> cocktail = Cocktail(peptide_backbone,ligand_library = ['Br', 'I'])
>>> combinations = cocktail.shake()
>>> FileWriter('example', combinations, 'mol2')
```

Cocktail Shaker makes your drug enumeration and expansion life easy. It also generates your files for you in as many formats needed for any cheminformatics software.

1.1 Features

- File parsing of TXT, SDF, and Chemical Smiles.
- File writing in a variety of formats some of which include: cif, sdf, pdb, mol, mol2 and many others
- Ability to recognize and expand libraries of compounds some of which include halogens, acyl halides, aldehydes.
- Ability to enumerate in 1D, and 2D structures and produce those compounds.
- Supports Python versions 3.3+.
- Released under the MPL license.

1.2 User guide

A step-by-step guide to getting started with Cocktail Shaker.

1.2.1 Installation

cocktail-shaker supports Python versions 3.3+. Pyhton's required dependencies, most notably RDKit, must be installed.

Dependencies can be located in either *setup.py*, *requirements.txt*, or a pdf generated graph of the cocktail shaker dependency.

Option 1 (Recommended): Use Pip

The easiest way to install cocktail-shaker is using pip:

```
pip install cocktail_shaker
```

This will download the latest version of cocktail-shaker and place it in your *site-packages* folder so it is automatically available to all your Python scripts.

If you do not have pip installed yet, you can install it using get-pip.py:

```
curl -O https://raw.github.com/pypa/pip/master/contrib/get-pip.py
python get-pip.py
```

Option 2: Download the Latest Release

Alternatively, you can get cocktail-shaker by manually download the latest release and installing it yourself:

```
tar -xzvf cocktail_shaker-1.1.6tar.gz
cd cocktail_shaker-1.0.0
python setup.py install
```

The setup.py command will install cocktail-shaker in your *site-packages* folder so it is automatically available to all your Python scripts.

Option 3: Clone the Repository

Lastly, the latest development version of cocktail-shaker is always available on GitHub. The version on GitHub is not guaranteed to be stable, but may include new features that have not yet been released.

Simply clone the repository and install as usual:

```
git clone https://github.com/Sulstice/Cocktail-Shaker.git
cd Cocktail-Shaker
python setup.py install
```

1.2.2 Getting Started

This page instructs you on how to get started with cocktail-shaker. To begin, make sure you have performed *installing cocktail_shaker*.

Basic Usage

The simplest way to use cocktail-shaker is to create a Peptide Builder object to generate your "base" peptide string, then to create cocktail object passing the peptide backbone and with the `shake` function you can generate your combinations:

```
>>> from cocktail_shaker import Cocktail
>>> from cocktail_shaker import PeptideBuilder
>>> peptide_backbone = PeptideBuilder(2)
>>> cocktail = Cocktail(
>>>                      peptide_backbone,
```

(continues on next page)

(continued from previous page)

```
>>>                         ligand_library = ['Br', 'I']
>>>                         )
>>> combinations = cocktail.shake()
>>> print (combinations)
>>> ['NC(Br)C(=O)NC(I)C(=O)NCC(=O)O', 'NC(I)C(=O)NC(Br)C(=O)NCC(=O)O']
```

Write the new compounds into an SDF file:

```
>>> FileWriter('new_compounds', new_compounds, 'sdf')
```

In this example, we have taken one SMILES string and expanded the compounds into a variety of variations into one SDF file.

Cocktail Shaker uses the PeptideBuilder class to generate base peptide backbone strings that can be then passed as into the cocktail shaker object. Alternatively, a user can generate their own peptide string as long as it conforms with the cocktail shaker requirements. See the peptide builder documentation for more information.

More Examples Peptide Builder

Generation of a circular peptide

```
>>> from cocktail_shaker import PeptideBuilder
>>> peptide_molecule = PeptideBuilder(3, circular=True)
>>> print (peptide_molecule)
>>> O=C1C([*:1])NC(=O)C([*:2])NC(=O)C([*:3])N1
```

Using the stereoisomer function with Cocktail Shaker

```
>>> from cocktail_shaker import Cocktail
>>> from cocktail_shaker import PeptideBuilder
>>> peptide_backbone = PeptideBuilder(1)
>>> cocktail = Cocktail(
>>>     peptide_backbone,
>>>     ligand_library = ['Br'],
>>>     enable_isomers = True
>>> )
>>> combinations = cocktail.shake()
>>> print (combinations)
>>> ['N[C@H](Br)C(=O)NCC(=O)O', 'N[C@@H](Br)C(=O)NCC(=O)O']
```

More Examples of Cocktail Shaker

Using the include amino acid function

```
>>> from cocktail_shaker import Cocktail
>>> from cocktail_shaker import PeptideBuilder
>>> peptide_backbone = PeptideBuilder(1)
>>> cocktail = Cocktail(
>>>     peptide_backbone,
>>>     include_amino_acids = True
>>> )
>>> combinations = cocktail.shake()
>>> print (combinations)
>>> ['NCCCCC(N)C(=O)NCC(=O)O', 'CC(O)C(N)C(=O)NCC(=O)O',
↪'NC(Cc1ccc(O)cc1)C(=O)NCC(=O)O', 'NC(=O)CCC(N)C(=O)NCC(=O)O',
```

(continues on next page)

(continued from previous page)

```
>>>  'CCC(C)C(N)C(=O)NCC(=O)O', 'NC(CS)C(=O)NCC(=O)O', 'NC(CC(=O)O)C(=O)NCC(=O)O',
↪'N=C(N)NCCCC(N)C(=O)NCC(=O)O',
>>>  'NC(Cc1c[nH]cn1)C(=O)NCC(=O)O', 'NC(Cc1ccccc1)C(=O)NCC(=O)O', 'CC(N)C(=O)NCC(=O)O
↪', 'NC(CCC(=O)O)C(=O)NCC(=O)O',
>>>  '[H]C(N)C(=O)NCC(=O)O', 'CSCCC(N)C(=O)NCC(=O)O', 'NC(CO)C(=O)NCC(=O)O',
↪'CC(C)C(N)C(=O)NCC(=O)O',
>>>  'NC(CCc1c[nH]c2ccccc12)C(=O)NCC(=O)O', 'CC(C)CC(N)C(=O)NCC(=O)O']
```

Using the Cocktail Shaker to generate a library of halogens & single atoms with then converting into a Pandas DataFrame Note to use this example you will need to install an extra dependency of ‘tables’ for handling h4 data.

```
>>> from peptide_builder import PeptideBuilder
>>> from functional_group_enumerator import Cocktail
>>> import pandas as pd
>>>
>>>
>>> peptide_molecule = PeptideBuilder(length_of_peptide=7)
>>> cocktail = Cocktail(peptide_backbone=peptide_molecule,
>>>                     ligand_library=["Br", "Cl", "I", "F", "O", "N", "C"],
>>>                     include_amino_acids=False,
>>>                     enable_isomers=False)
>>> molecules = cocktail.shake()
>>>
>>> dataframe = pd.DataFrame(molecules, columns=["Smiles"])
>>> dataframe.to_hdf('data.h5', key='s', mode='w')
```

1.3 API documentation

Comprehensive API documentation with information on every function, class and method.

1.3.1 File Formats

Input can be parsed in a variety of file formats that are autodetected:

```
>>> FileParser('compound.sdf')

['RDKit MolObject']
```

The full list of file formats that cocktail-shaker supports parsing:

```
sdf       # 2D Structure file format
```

Output can be returned in a variety of file formats that are specified your enumerated compounds and your desired file extension:

```
>>> FileWriter('test'', compounds, '.mol2')

Writes all the compounds to 1 mol2 file.
```

The full list of file formats:

```
alc          # Alchemy format
cdxml        # CambridgeSoft ChemDraw XML format
cerius       # MSI Cerius II format
charmm       # Chemistry at Harvard Macromolecular Mechanics file format
cif          # Crystallographic Information File
cml          # Chemical Markup Language
gjf          # Gaussian input data file
gromacs      # GROMACS file format
hyperchem    # HyperChem file format
jme          # Java Molecule Editor format
maestro      # Schroedinger MacroModel structure file format
mol          # Symyx molecule file
mol2         # Tripos Sybyl MOL2 format
pdb          # Protein Data Bank
sdf          # 2D formatted structure data files
sdf3000      # Symyx Structure Data Format 3000
sln          # SYBYL Line Notation
xyz          # xyz file format
```

1.3.2 File API Documentation

This page provides an overview of how cocktail-shaker operates reading and writing files. To see the files that cocktail-shaker supports, refer to *file formats*.

The FileParser Module

You instantiate a `FileParser` by providing the exact path to a file with the extension included. cocktail-shaker is smart enough to detect the file extension and allocate its specific parsing. If the file being parsed is *not* supported, then `FileNotSupportedError` will be raised instead.

```
>>> from cocktail_shaker import FileParser
>>> molecules = FileParser('compounds.sdf')
>>> print (molecules)
>>> ['c1cc(CCCO)ccc1']
```

FileParser will then return a list of SMILES.

`file_path`
The path to a file

The FileWriter Module

The FileWriter module uses the NIH web cactus resolver under the hood to generate the 2D or 3D data. By issuing requests of the compound in SMILES and returning a XML formatted data struct of the 2D/3D coordinates in whatever chemical format indicated by the user. Unfortunately, the cactus resolver can only process one api call at a time and not bulk requests. This means that sending combinations worth of 1000+ -> 1000+ api requests might not be in your best interest and eventually the server will deny any requests.

This feature is still in development so use at your own discretion.

You instantiate a `FileWriter` by providing the path to the file, compounds to be written, and the extension you would like the files in. If the file being written is *not* supported, then `FileNotSupportedError` will be raised instead.

```
>>> from cocktail_shaker import Cocktail, FileWriter
>>> from cocktail_shaker import PeptideBuilder
>>> peptide_backbone = PeptideBuilder(2)
>>> cocktail = Cocktail(peptide_backbone,ligand_library = ['Br', 'I'])
>>> combinations = cocktail.shake()
>>> FileWriter('new_compounds', combinations, 'sdf')
Generates an SDF File....
```

However, if you would like to generate the files into separate files, then you can pass in the fragmentation parameter.

```
>>> from cocktail_shaker import Cocktail, FileWriter
>>> from cocktail_shaker import PeptideBuilder
>>> peptide_backbone = PeptideBuilder(2)
>>> cocktail = Cocktail(peptide_backbone,ligand_library = ['Br', 'I'])
>>> combinations = cocktail.shake()
>>> FileWriter('new_compounds', combinations, 'sdf', fragmentation=2)
Generates 2 SDF Files....
```

name
The path to a file

molecules
List of RDKit molecules you would like to write.

option
The extension of the file you would like to write

fragmentation (*optional*)
How many files you would like to produce.

1.3.3 Functional Groups

Below is a list of the functional groups that cocktail-shaker currently supports.

Functional Group	Class	SMILES	Smarts
Bromine	Halogens	Br	[Br]
Chlorine	Halogens	Cl	[Cl]
Fluorine	Halogens	F	[F]
Acyl Bromide	Acyl Halides	C(=O)Br	[CX3](=[OX1])[Br]
Acyl Chloride	Acyl Halides	C(=O)Cl	[CX3](=[OX1])[Cl]
Acyl Fluoride	Acyl Halides	C(=O)F	[CX3](=[OX1])[F]
Acyl Iodide	Acyl Halides	C(=O)I	[CX3](=[OX1])[I]
Primary Alcohol	Alcohols	O	[OX2H]
Ketone	Ketones	C(=O)OC	[CX3]=[OX1]
Carboxylic Acid	Acids	C(=O)O	[CX3](=O)[OX1H0-,OX2H1]
Acetic Anhydride	Anhydrides	CC(=O)OC(=O)C	[CX3](=[OX1])[OX2][CX3](=[OX1])
Primary Amine	Amines	N	[NX3;H2;!$(NC=[!#6]);!$(NC#[!#6])][#6]
Secondary Amine	Amines	NC	[NX3;H2,H1;!$(NC=O)]
Enamine	Amines	N	[NX3][CX3]=[CX3]
Amide	Amides	C(=O)N	[NX3][CX3](=[OX1])[#6]
Nitro	Nitros	[N+](=O)[O-]	[$([NX3](=O)=O),$([NX3+](=O)[O-])][!#8]
Sulfoxide	Sulfoxides	S(=O)(=O)	[$([#16X3](=[OX1])([#6])[#6]),$([#16X3+]([OX1-])([#6])[#6])]
Ether	Ethers	COC	[OD2]([#6])[#6]
Azide	Azides	C([N-][N+]#N)	[$(-*[NX2-]-[NX2+]#[NX1]),$(*-[NX2]=[NX2+]=[NX1-])]

If you would like to add to this list, the functional group information is stored in the **datasource** directory under R_Groups.yml.

Each smart pattern recognition is tested thoroughly before moving into the cocktail-shaker package, so please PR your test as well. You can follow the schema outlined in the function **test_primary_finding_r_groups**.

1.3.4 Amino Acids

Below is a list of the amino acids that cocktail-shaker currently supports.

Amino Acid	SMILES
Alanine	C
Arginine	CCCCNC(N)=N
Asparagine	CCC(N)=O
Aspartic Acid	CC(O)=O
Cysteine	CS
Glutamic Acid	CCC(O)=O
Glutamine	CCC(N)=O
Glycine	[H]
Histidine	CC1=CNC=N1
Isoleucine	C(CC)([H])C
Leucine	CC(C)C
Lysine	CCCCN
Methionine	CCSC
PhenylAlanine	CC1=CC=CC=C1
Proline	C2CCCN2
Serine	CO
Threonine	C(C)([H])O
Tryptophan	CCC1=CNC2=C1C=CC=C2
Tyrosine	CC1=CC=C(O)C=C1
Valine | C(C)C	

1.3.5 Cocktail API

This page introduces the functionality of the cocktail object and provides a deeper look into what it can accomplish in the future.

The GlobalChem Class

`amino_acid_side_chains`
The peptide backbone string generated from PeptideBuilder

`ligand_library`
The list of the ligands you would like installed on the peptide. It can be of any order.

`enable_isomers` (*optional*)
Include stereochemistry and stereoisomers in the the results. Defaults to False.

`include_amino_acids` (*optional*)
Include the natural amino acids except for Proline (TBD), list of smiles found in *amino acids*. Defaults to False.

1.3.6 Peptide Builder API

This page introduces the functionality of the peptide molecule object and provides a deeper look into what it can accomplish in the future.

The PeptideBuilder Class

The peptide molecule class is the preliminary peptide builder work before you pass it into Cocktail Shaker. It will build the peptides smiles for you (woo!)

You instantiate a `PeptideBuilder` object by parsing in the length of peptide marked by how many amino acid sides you would like. Peptide Molecule will handle any SMARTS and validation under the hood so you will not have too. A simple example is detailed below:

```
>>> from cocktail_shaker import PeptideBuilder
>>> peptide_molecule = PeptideBuilder(2)
>>> print (peptide_molecule)
>>> NCC(NC([*:1])C(NC([*:2])C(O)=O)=O)=O
```

Here we have two slots ready for our combinations in the cocktail. Alternatively, you can include the `include_proline` function on the n-terminus that will provide a SMILES string like so.

```
>>> from cocktail_shaker import PeptideBuilder
>>> peptide_molecule = PeptideBuilder(2, include_proline=True)
>>> print (peptide_molecule)
>>> N2CCCC2C(NC([*:1])C(NC([*:2])C(O)=O)=O)=O
```

Generation of a circular peptide

```
>>> from cocktail_shaker import PeptideBuilder
>>> peptide_molecule = PeptideBuilder(3, circular=True)
>>> print (peptide_molecule)
>>> O=C1C([*:1])NC(=O)C([*:2])NC(=O)C([*:3])N1
```

length_of_peptide
Length of the peptide the user would like generated

include_proline (*optional*)
Whether the user would like to include proline on the N-Terminus. Defaults to False.

circular (*optional*)
If you would like the peptide to be circular or not.

1.3.7 Contributing

Contributing

Contributions of any kind to cocktail-shaker are greatly appreciated! All contributions are welcome, no matter how big or small.

If you are able to contribute changes yourself, just fork the source code on GitHub, make changes, and file a pull request.

Quick Guide to Contributing

1. Fork the Cocktail-Shaker repository on GitHub, then clone the fork to your local machine:

```
git clone https://github.com/<username>/Cocktail-Shaker.git
```

2. Install the development requirements:

```
cd Cocktail-Shaker
pip install -r requirements/development.txt
```

3. Create a new branch for your changes:

```
git checkout -b <name-for-changes>
```

4. Make your changes or additions. Ideally create some tests and ensure they pass.
5. Commit your changes and push to your fork on GitHub:

```
git add .
git commit -m "<description-of-changes>"
git push origin <name-for-changes>
```

4. Submit a pull request.

Tips

- Follow the PEP8 style guide.
- Include docstrings as described in PEP257.
- Try and include tests that cover your changes.
- Try to write good commit messages.
- Consider squashing your commits with rebase.
- Read the GitHub help page on Using pull requests.

1.4 Cocktail Shaker's license

Cocktail Shaker is released under the Mozilla Public License 2.0. This is a short, permissive software license that allows commercial use, modifications, distribution, sublicensing and private use. Basically, you can do whatever you want with Cocktail Shaker as long as you include the original copyright and license in any copies or derivative projects.

INDEX