mosdef_cassandra

Ryan S. DeFever

Jul 11, 2022

GETTING STARTED

CHAPTER ONE

OVERVIEW

MoSDeF Cassandra is a Python interface for the **Cassandra** Monte Carlo software. It offers complete integration with the **MoSDeF** tools and a user-friendly interface for Cassandra.

Warning: MoSDeF Cassandra is still in early development (0.x releases). The API may change unexpectedly.

CHAPTER TWO

RESOURCES

- *Installation guide*: Instructions for installing MoSDeF Cassandra
- *Key Concepts*: How we think about MoSDeF Cassandra
- GitHub repository: View the source code, contribute, and raise issues
- Cassandra: Learn more about the Cassandra Monte Carlo software
- Cassandra respository: View the source of the Cassandra Monte Carlo software
- MoSDeF tools: A collection of tools for constructing systems and applying forcefield parameters for particle-based simulations

CHAPTER THREE

CITATION

Please cite **MoSDeF Cassandra**, **Cassandra**, and the **MoSDeF** suite of tools if you use this tool in your research. See *here* for details.

CHAPTER

FOUR

INSTALLATION

Installation instructions are *here*. A conda installation is available:

```
conda create --name mc mosdef_cassandra -c conda-forge
```

CHAPTER

FIVE

EXAMPLE

MoSDeF Cassandra provides a Python interface to Cassandra. The workflow consists of first constructing a system and move set. These two objects are passed to a function that calls Cassandra to perform the Monte Carlo simulation. The example below demonstrates an NVT Monte Carlo simulation of OPLS methane. No additional files are required. Everything required to run the Monte Carlo simulation is contained in the script below.

```
import mbuild
import foyer
import mosdef_cassandra as mc

# Create a methane molecule from a SMILES string
methane = mbuild.load("C", smiles=True)

# Load the forcefield via foyer
ff = foyer.forcefields.load_OPLSAA()

# Apply the forcefield parameters to methane with foyer
methane_ff = ff.apply(methane)

# Define an empty simulation box
box = mbuild.Box([3.0, 3.0, 3.0])

# Define the boxes, species in the system, molecules in the box
ensemble = 'nvt'
box_list = [box]
species_list = [methane_ff]
molecules_to_add = [[100]]

# Create the System
system = mc.System(box_list, species_list, mols_to_add=molecules_to_add)

# Create the MoveSet
moveset = mc.MoveSet(ensemble, species_list)

# Run a Monte Carlo simulation!
mc.run(
    system=system,
    moveset=moveset,
    run_type="equilibration",
    run_length=1000,
    temperature=300.0 * u.K
)
```

Several additional examples can be found *here*.

CHAPTER SIX

CREDITS

Development of MoSDeF Cassandra was supported by the National Science Foundation under grant NSF Grant Number 1835874. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

See *here* for complete credits.

CHAPTER

SEVEN

TABLE OF CONTENTS

7.1 Introduction

MoSDeF Cassandra is a Python interface for the **Cassandra** Monte Carlo software. It offers complete integration with the **MoSDeF** tools and a user-friendly interface for Cassandra.

Warning: **MoSDeF Cassandra** is still in early development (0.x releases). The API may change unexpectedly.

7.2 Installation

We recommend the conda installation for most users. The conda installation will install MoSDeF Cassandra, Cassandra, and all other required dependencies. If you wish to contribute to MoSDeF Cassandra, you may install from source.

7.2.1 Installing with conda

If you already have conda installed, you can create a new conda environment and install MoSDeF Cassandra with a single command:

```
conda create --name mc mosdef_cassandra -c conda-forge
```

The command creates a new conda environment (`mc`) and installs `mosdef_cassandra`. The `-c` flag specifies the conda channels that are searched. To use the environment, run `conda activate mc`.

You can test your installation by opening up a Python interpreter and typing:

```
import mosdef_cassandra as mc
(py, fraglib_setup, cassandra) = mc.utils.detect_cassandra_binaries()
```

If the module is imported without error and is able to find the required binaries (`python`, `cassandra.exe`, and `library_setup.py`, you have successfully installed the package. Example output from the second line is:

```
Using the following executables for Cassandra:
Python: /Users/username/anaconda3/envs/mc-prod/bin/python
library_setup: /Users/username/anaconda3/envs/mc-prod/bin/library_setup.py
Cassandra: /Users/ryandefever/anaconda3/envs/mc-prod/bin/cassandra.exe
```

7.2.2 Installing from source

MoSDeF Cassandra may alternatively be installed from source. First, clone MoSDeF Cassandra from GitHub to a location of your choosing:

```
git clone git@github.com:maginngroup/mosdef_cassandra.git
```

Next, install the required dependencies. You can use the dependencies listed in `requirements.txt` or `requirements-dev.txt`. However, if you are installing from source we recommend the latter:

```
conda install -c conda-forge --file mosdef_cassandra/requirements-dev.txt
```

Finally, run the following commands to complete the installation of MoSDeF Cassandra:

```
cd mosdef_cassandra/
pip install .
```

7.2.3 Installing Cassandra from source

Note: Installing Cassandra from source is unnecessary unless you wish to modify the source code of Cassandra or use a hardware specific (e.g., intel) compiler.

Once you have downloaded the tarball (available here):

```
tar -xzvf Cassandra-1.2.5.gz
cd Cassandra-1.2.5/Src
make -f Makefile.gfortran
cd ../
mkdir bin/
mv Src/cassandra_gfortran.exe ./bin/.
cp Scripts/Frag_Library_Setup/library_setup.py ./bin/.
```

Note: You may also wish to use the openMP version. In that case use the `Makefile.gfortran.openMP` and move the relevant executable to `bin/`. Depending on system size, Cassandra the openMP version may offer speedups for up to ~8 cores. The number of OMP threads can be controlled by setting the `OMP_NUM_THREADS` environment variable, e.g., `export OMP_NUM_THREADS=8`.

Add `Cassandra-1.2.5/bin` to your `PATH`:

```
export PATH=path_to_install/Cassandra-1.2.5/bin:${PATH}
```

Unless you add the preceding line to your `.bashrc` you will need to run it every time you open a new terminal window.

7.3 Quickstart Guide

The following assumes you have MoSDeF Cassandra installed. If not, please refer to our *installation guide*. More details of the core MoSDeF Cassandra functionality can be found under the Guides section of the documentation.

Let's start by setting up an NVT Monte Carlo simulation of OPLS-AA methane. We will use the `mbuild` and `foyer` packages to create the methane molecule, and `mosdef_cassandra` to run the Monte Carlo simulation. We begin with the required imports:

```python
import mbuild
import foyer
import unyt as u
import mosdef_cassandra as mc
```

Next, we create an all-atom methane molecule from a SMILES string:

```python
methane = mbuild.load("C", smiles=True)
```

`methane` is a single all-atom methane molecule. It is an `mbuild.Compound`. `methane` contains particles for each element (C, H, H, H) in the molecule, coordinates associated with each particle, and the bonds that describe the particle connectivity. However, there are no forcefield parameters associated with `methane`.

To add forcefield parameters to `methane`, we first load the OPLS-AA forcefield from foyer. The OPLS-AA forcefield is distributed with foyer. Be aware that not all atomtypes are currently defined.

```python
oplsaa = foyer.forcefields.load_OPLSAA()
```

We then apply the forcefield using foyer:

```python
methane_ff = oplsaa.apply(methane)
```

`methane_ff` is a `parmed.Structure` that contains all the forcefield parameters for our methane molecule.

Now that we have a molecule with forcefield parameters, the next step is to define our simulation box. Since Cassandra can add molecules to a simulation box before the start of a simulation, we can begin with an empty simulation box. We will define an `mbuild.Box` with the box lengths specified in nanometers:

```python
box = mbuild.Box([3.0, 3.0, 3.0])
```

Warning: Even though most quantities in MoSDeF Cassandra must be *specified with the unyt package*, the `mbuild.Box` object is specified in nanometers without using `unyt`. This is because `mbuild` does not currently support `unyt`.

Next, we create the `System` object. It has two required arguments and two optional arguments, depending on your system. The `box_list` and `species_list` are always specified. The `box_list` is simply a list of the simulation boxes in the system. In this case, since we are performing an NVT simulation there is only our single `box`. The `species_list` is a list of the unique chemical species in our system. Here we only have methane.

The two system-dependent arguments are `mols_in_boxes` and `mols_to_add`. Here we have an empty initial box, so we don't need to specify `mols_in_boxes`. Finally, `mols_to_add` specifies the number of molecules that we wish to add to each box prior to beginning the simulation in Cassandra. We will add 50 methane molecules for this example.

```
box_list = [box]
species_list = [methane_ff]
mols_to_add = [[50]]
```

Note: mols_in_boxes and mols_to_add are lists with one entry for each box. Each entry is itself a list, with one entry for each species in the species_list.

We now combine the four components created above into a System:

```
system = mc.System(box_list, species_list, mols_to_add=mols_to_add)
```

Note: mols_in_boxes and mols_to_add are optional arguments when creating the System object. If not provided, the values are taken as zero for all species in all boxes.

Note: Each item in the species_list must be a parmed.Structure object with the associated forcefield parameters. For example, species_list = [methane] would not work because unlike methane_ff, methane is a mbuild.Compound and does not contain forcefield parameters.

Now we create our MoveSet. The MoveSet contains all selections related to the MC moves that will be performed during the simulation. In addition to the probability of performing different types of MC moves, the MoveSet contains the maximum move sizes (e.g., maximum translation distance), whether each species is insertable, and more. To create the MoveSet, we specify the ensemble in which we wish to perform the MC simulation and provide the species_list.

```
ensemble = 'nvt'
moveset = mc.MoveSet(ensemble, species_list)
```

Some attributes of the MoveSet can be edited after it is created. This allows complete control over all the move-related selections in Cassandra. To view the current selections, use moveset.print().

The final step is to run the simulation. The run function requires five arguments: the System, MoveSet object, a selection of "equilibration" or "production" (run_type), the simulation length (run_length), and the desired temperature. Note that since the temperature is a physical quantity it must be specified with *units attached*.

```
mc.run(
    system=system,
    moveset=moveset,
    run_type="equilibration",
    run_length=10000,
    temperature=300.0 * u.K
)
```

A large number of additional keyword arguments can be provided inline or as part of a keyword dictionary. See mc.print_valid_kwargs() for a complete list of the available keyword arguments.

7.4 Examples

Below we provide a few simple examples of short Monte Carlo simulations with MoSDeF Cassandra.

7.4.1 NVT simulation of methane

```
import mbuild
import foyer
import mosdef_cassandra as mc
import unyt as u

# Use mBuild to create a methane molecule
methane = mbuild.load("C", smiles=True)

# Create an empty mbuild.Box
box = mbuild.Box(lengths=[3.0, 3.0, 3.0])

# Load force field
oplsaa = foyer.forcefields.load_OPLSAA()

# Use foyer to apply force field to methane
methane_ff = oplsaa.apply(methane)

# Create box and species list
box_list = [box]
species_list = [methane_ff]

# Use Cassandra to insert some initial number of methane molecules
mols_to_add = [[50]]

# Define the System
system = mc.System(box_list, species_list, mols_to_add=mols_to_add)
# Define the MoveSet
moveset = mc.MoveSet("nvt", species_list)

# Run a simulation at 300 K for 10000 MC moves
mc.run(
    system=system,
    moveset=moveset,
    run_type="equilibration",
    run_length=10000,
    temperature=300.0 * u.K,
)
```

7.4.2 NPT simulation of methane

```python
import mbuild
import foyer
import mosdef_cassandra as mc
import unyt as u

# Use mbuild to create molecules
methane = mbuild.load("C", smiles=True)

# Create an empty mbuild.Box
box = mbuild.Box(lengths=[3.0, 3.0, 3.0])

# Load force field
oplsaa = foyer.forcefields.load_OPLSAA()

# Use foyer to apply force field
methane_ff = oplsaa.apply(methane)

# Create box and species list
box_list = [box]
species_list = [methane_ff]

# Use Cassandra to insert some initial number of species
mols_to_add = [[5]]

# Define the System
system = mc.System(box_list, species_list, mols_to_add=mols_to_add)
# Define the MoveSet
moveset = mc.MoveSet("npt", species_list)

# Here we must specify the pressure since we are performing a
# NpT simulation. It can be provided in the custom_args dictionary
# or as a keyword argument to the "run" function.
custom_args = {
    "pressure": 1.0 * u.bar,
}

# Run a simulation with at 300 K with 10000 MC moves
mc.run(
    system=system,
    moveset=moveset,
    run_type="equilibration",
    run_length=10000,
    temperature=300.0 * u.K,
    **custom_args,
)
```

7.4.3 NVT simulation of methane and propane mixture

```python
import mbuild
import foyer
import mosdef_cassandra as mc
import unyt as u

# Use mbuild to create methane and propane molecules
methane = mbuild.load("C", smiles=True)
propane = mbuild.load("CCC", smiles=True)

# Create an empty mbuild.Box
box = mbuild.Box(lengths=[3.0, 3.0, 3.0])

# Load force field
oplsaa = foyer.forcefields.load_OPLSAA()

# Use foyer to apply the force field
typed_methane = oplsaa.apply(methane)
typed_propane = oplsaa.apply(propane)

# Create box and species list
box_list = [box]
species_list = [typed_methane, typed_propane]

# Use Cassandra to insert some initial number of species
mols_to_add = [[100, 50]]

system = mc.System(box_list, species_list, mols_to_add=mols_to_add)
moveset = mc.MoveSet("nvt", species_list)

mc.run(
    system=system,
    moveset=moveset,
    run_type="equilibration",
    run_length=10000,
    temperature=200.0 * u.K,
)
```

7.4.4 GEMC simulation of methane (united atom)

```python
import mbuild
import foyer
import mosdef_cassandra as mc
import unyt as u

# Use mbuild to create a coarse-grained CH4 bead
methane = mbuild.Compound(name="_CH4")

# Create two empty mbuild.Box
# (vapor = larger, liquid = smaller)
```

(continues on next page)

(continued from previous page)

```python
liquid_box = mbuild.Box(lengths=[3.0, 3.0, 3.0])
vapor_box = mbuild.Box(lengths=[4.0, 4.0, 4.0])

# Load force field
trappe = foyer.forcefields.load_TRAPPE_UA()

# Use foyer to apply force field
typed_methane = trappe.apply(methane)

# Create box and species list
box_list = [liquid_box, vapor_box]
species_list = [typed_methane]

mols_to_add = [[350], [100]]

system = mc.System(box_list, species_list, mols_to_add=mols_to_add)
moveset = mc.MoveSet("gemc", species_list)

moveset.prob_volume = 0.010
moveset.prob_swap = 0.11

thermo_props = [
    "energy_total",
    "energy_intervdw",
    "pressure",
    "volume",
    "nmols",
    "mass_density",
]

custom_args = {
    "run_name": "equil",
    "charge_style": "none",
    "rcut_min": 2.0 * u.angstrom,
    "vdw_cutoff": 14.0 * u.angstrom,
    "units": "sweeps",
    "steps_per_sweep": 450,
    "coord_freq": 50,
    "prop_freq": 10,
    "properties": thermo_props,
}

mc.run(
    system=system,
    moveset=moveset,
    run_type="equilibration",
    run_length=250,
    temperature=151.0 * u.K,
    **custom_args,
)

# Update run_name and restart_name
```

(continues on next page)

(continued from previous page)

```
custom_args["run_name"] = "prod"
custom_args["restart_name"] = "equil"

mc.restart(
    system=system,
    moveset=moveset,
    run_type="production",
    run_length=750,
    temperature=151.0 * u.K,
    **custom_args,
)
```

7.4.5 GCMC simulation of methane

```
import mbuild
import foyer
import mosdef_cassandra as mc
import unyt as u

# Use mbuild to create a methane
methane = mbuild.load("C", smiles=True)

# Create an empty mbuild.Box
box = mbuild.Box(lengths=[10.0, 10.0, 10.0])

# Load force field
oplsaa = foyer.forcefields.load_OPLSAA()

# Use foyer to apply the force field
methane_ff = oplsaa.apply(methane)

# Create box and species list
box_list = [box]
species_list = [methane_ff]

mols_to_add = [[100]]

system = mc.System(box_list, species_list, mols_to_add=mols_to_add)
moveset = mc.MoveSet("gcmc", species_list)

custom_args = {
    "chemical_potentials": [-35.0 * (u.kJ / u.mol)],
    "prop_freq": 100,
}

mc.run(
    system=system,
    moveset=moveset,
    run_type="equilibration",
    run_length=1000,
```

(continues on next page)

(continued from previous page)

```
    temperature=300.0 * u.K,
    **custom_args,
)
```

7.4.6 GCMC simulation of methane adsorption in a solid framework

```
import mbuild
import foyer
import mosdef_cassandra as mc
import unyt as u

from mosdef_cassandra.examples.structures import carbon_lattice


# Load a structure created with mbuild
lattice = carbon_lattice()
# Use mbuild to create a methane
methane = mbuild.load("C", smiles=True)

# Load force field
trappe = foyer.forcefields.load_TRAPPE_UA()
oplsaa = foyer.forcefields.load_OPLSAA()

# Use foyer to apply the force fields
typed_lattice = trappe.apply(lattice)
methane_ff = oplsaa.apply(methane)

# Create box and species list
box_list = [lattice]
species_list = [typed_lattice, methane_ff]

# Since we have an occupied box we need to specify
# the number of each species present in the initial config
mols_in_boxes = [[1, 0]]

system = mc.System(box_list, species_list, mols_in_boxes=mols_in_boxes)
moveset = mc.MoveSet("gcmc", species_list)

custom_args = {
    "chemical_potentials": ["none", -30.0 * (u.kJ / u.mol)],
    "rcut_min": 0.5 * u.angstrom,
    "vdw_cutoff": 14.0 * u.angstrom,
    "charge_cutoff": 14.0 * u.angstrom,
    "coord_freq": 100,
    "prop_freq": 10,
}

mc.run(
    system=system,
    moveset=moveset,
```

(continues on next page)

(continued from previous page)

```
    run_type="equilibration",
    run_length=10000,
    temperature=300.0 * u.K,
    **custom_args,
)
```

7.4.7 NVT simulation of SPC/E water

```
import mbuild
import foyer
import mosdef_cassandra as mc
import unyt as u
from mosdef_cassandra.utils.get_files import get_example_ff_path, get_example_mol2_path

# Load water with SPC/E geometry from mol2 file
molecule = mbuild.load(get_example_mol2_path("spce"))

# Create an empty mbuild.Box
box = mbuild.Box(lengths=[3.0, 3.0, 3.0])

# Load force field
spce = foyer.Forcefield(get_example_ff_path("spce"))

# Use foyer to apply force field
molecule_ff = spce.apply(molecule)

# Create box and species list
box_list = [box]
species_list = [molecule_ff]

# Use Cassandra to insert some initial number of species
mols_to_add = [[50]]

# Define the System
system = mc.System(box_list, species_list, mols_to_add=mols_to_add)
# Define the MoveSet
moveset = mc.MoveSet("nvt", species_list)

# Note here we need to use the angle_style="fixed" keyword argument
# SPC/E geometry is rigid; default angle style is "harmonic"
custom_args = {"angle_style": ["fixed"]}

# Run a simulation with at 300 K with 10000 MC moveset
mc.run(
    system=system,
    moveset=moveset,
    run_type="equilibration",
    run_length=10000,
    temperature=300.0 * u.K,
    **custom_args,
)
```

7.5 Philosophy

Performing a Monte Carlo simulation should be simple and intuitive. The simulation setup procedure should not be prone to error. The process should be easily reproducible and extensible. And none of the prior goals can sacrifice the complete flexibility required by the expert simulator.

7.6 Key Concepts

MoSDeF Cassandra integrates the Cassandra Monte Carlo (MC) code with the Molecular Simulation Design Framework (MoSDeF). This integration enables users to build systems, apply force field parameters, and setup and run MC simulations from within a single Python script. In principle, MoSDeF Cassandra should make it easier to create TRUE simulations – simulations that are transparent, reproducible, usable by others, and extensible. In addition to improving simulation workflow reproducibility, MoSDeF Cassandra also provides a user-friendly interface to Cassandra to improve the Cassandra user experience for beginning and expert simulators alike.

The Cassandra documentation contains a fairly comprehensive detail on the theory of MC simulations, with a particular focus on the algorithms used in Cassandra. Canonical textbooks by Frenkel and Smit, Allen and Tidesley, and Tuckerman also provide useful reference materials for background on MC simulations. Here however, we focus on the MoSDeF Cassandra package, proceeding with the assumption that the reader has a basic understanding of MC methods as applied in molecular simulations.

7.6.1 Organization and Implementation

The MoSDeF Cassandra interface is motivated by a simple question: **What is the simplest logical organization of the components of a Monte Carlo simulation?**

Our answer to this question divides the simulation setup into two components: the *system* and the *move set*. The *system* specifies what you are simulating; the simulation box(es), initial configuration(s), and the forcefield parameters. The *move set* specifies what happens during the simulation; the types of MC moves that are attempted, the probabilities of each, and any other parameters required to define the attempted moves.

MoSDeF Cassandra implements this organization by mapping the process of setting up and running an MC simulation into three discrete steps:

1. Create the *System*
2. Create the *MoveSet*
3. Pass the `System` and `MoveSet` to the *run function*

Dive into our *Quickstart Guide* or *Examples* to see this workflow in action!

7.7 Unyts

Unyt is a Python library for working with physical units. In MoSDeF Cassandra, all quantities that have physical units associated with them must be specified as a `unyt_quantity`. This approach yields several benefits. Users can specify quantities in any (dimensionally valid) units they desire, and thus do not need to dig through the reference manual to determine the correct units for each quantity. Possible errors in unit conversions are mitigated, and we remove any possible ambiguity with regards to the units of physical quantities in MoSDeF Cassandra scripts.

7.7.1 Basic usage

Adding units to quantities is as easy as:

```
import unyt as u
temperature = 300 * u.K
```

Compound units can be specified as:

```
import unyt as u
energy = 100 * u.Unit('kJ/mol')
```

If a quantity or array is specified as a `unyt_quantity` or `unyt_array`, then performing a unit conversion is as simple as:

```
import unyt as u
energy = 100 * u.Unit('kJ/mol')
energy.in_units('kcal/mol')
```

The value (without units) can be extracted as:

```
energy.in_units('kcal/mol').value
```

7.7.2 Unyts in MoSDeF Cassandra

The base data structure of `unyt` is the `unyt_array` or `unyt_quantity` (a subclass of numpy ndarray) which carries both a value and a unit. One of the main functionalities of **unyt** is the ability to convert units. In **MoSDeF Cassandra**, a user can pass in a `unyt_quantity` of any valid unit type which will get then get converted into the standard unit specified by **Cassandra**. Unyt arrays are expected for values with units, such as cutoffs, angles, volumes, pressures, and temperatures. Unyt arrays are **not** expected for dimensionless values such as probabilities. A list of arguments and their required type can be viewed by running `mosdef_cassandra.print_valid_kwargs`.

7.7.3 Important Cavaets

`mBuild` does not use the `unyt` package. The distance units in `mBuild` are **nanometers**.

7.8 System

The `System` contains all the details of the system (i.e., *what* is being simulated). This includes the simulation box(es), any initial structure(s) in the simulation box(es), and the force field parameters describing the interactions between particles in the system.

The `System` is one of two objects that must be created prior to running a MC simulation. Creating the `System` requires specification of the following:

- A list of the simulation boxes (`box_list`)
- A list of the unique chemical species (`species_list`)

and perhaps,

- The number of molecules already present in each box in the `box_list` (`mols_in_boxes`)
- The number of molecules for Cassandra to add to each box prior to beginning the MC simulation (`mols_to_add`)

Instantiating the System normally appears as follows:

```
import mosdef_cassandra as mc
system = mc.System(
    box_list,
    species_list,
    mols_in_boxes=mols_in_boxes,
    mols_to_add=mols_to_add
)
```

These items comprise a complete description of the system to be simulated in Cassandra. The box information, force-field information, number of species, and coordinates of any initial structure are contained within this object.

7.8.1 box_list

The `box_list` is a Python `list` of the simulation boxes in the system. It should contain a single item in the case of simulations performed in the NVT, NPT, or GCMC ensembles, and two items for simulations in the GEMC or GEMC-NPT ensembles.

Each simulation box can be empty or contain an initial structure. If a simulation box is empty, then the list element should be an `mbuild.Box`. An `mbuild.Box` can be created as follows:

```
box = mbuild.Box(lengths=[3.0, 3.0, 3.0], angles=[90., 90., 90.])
```

where the lengths are specified in units of nanometers and the box angles are specified in degrees. If the angles are not specified they are taken as 90 degrees.

If a simulation box contains an initial structure, then the list element should be an `mbuild.Compound` object. mBuild supports reading many common simulation file formats via `mbuild.load`. See the mBuild documentation for more details.

Warning: If an initial structure (i.e., an `mbuild.Compound`) is provided, the order of atoms is *very* important. Each complete molecule must appear one after another. Within each molecule, the order of atoms in *must match* the order of atoms in the relevant species provided in the `species_list`. If there are multiple different species, then all molecules of species1 must be provided before any molecules of species2, and so on. We hope to relax these restrictions in future releases.

For a single-box simulation with an initially empty simulation box:

```
box = mbuild.Box([3.,3.,3.])
box_list = [box]
```

For a two-box simulation where one box is initially empty and initial structure for the other simulation box is loaded from a PDB file with `mbuild`:

```
zeolite_box = mbuild.load("zeolite.pdb")
vapor_box = mbuild.Box([3.,3.,3.])

box_list = [zeolite_box, vapor_box]
```

In this case, the initial structure and box dimensions for the box containing the zeolite are taken from the PDB file. Note that the box dimensions can be manually edited by changing the `mbuild.Compound.periodicity` attribute.

Note: The box lengths are taken from the `periodicity` attribute of the `mbuild.Compound` object. The box angles are taken from `mbuild.Compound.boundingbox.angles`. This is temporary solution due to the fact the `mbuild.Compound.boundingbox.lengths` attribute is calculated on-the-fly from the extent of the particles in the `Compound` rather than storing periodic box information, while the `Compound.perodocitiy` attribute contains no information regarding the box angles.

7.8.2 species_list

The `species_list` is a Python `list` of the unique chemical species in the system. For example, a simulation of pure methane contains one unique chemical species (methane), regardless of the number of methane molecules in the simulation. A simulation containing a mixture of methane and ethane has two unique chemical species. Therefore, in the first example, the `species_list` contains a single item and in the second example the `species_list` contains two items. Each item in the `species_list` is a `parmed.Structure`. All the forcefield required force field parameters for each species must be in their respective `parmed.Structure`.

Note: The `parmed.Structure` will be replaced with a `gmso.Topology` as the GMSO package matures.

For example, to simulate a mixture of methane and ethane with the OPLS-AA force field, we could use the following sequence of steps to generate the species list. Note that mbuild and foyer allow us to generate a molecule with force field parameters from a SMILES string and a few lines of Python code.

```
import mbuild
import foyer

methane = mbuild.load("C", smiles=True)
ethane = mbuild.load("CC", smiles=True)

ff = foyer.forcefields.load_OPLSAA()

methane_ff = ff.apply(methane)
ethane_ff = ff.apply(ethane)

species_list = [methane_ff, ethane_ff]
```

Note: The order of items in species list determines the labeling of the species. The first is considered species1, the second species2, and so forth.

7.8.3 mols_in_boxes

The `mols_in_boxes` is a `list` containing the number of molecules of each species currently in each box specified in `box_list`. If all simulation box(es) are empty, `mols_in_boxes` does not need to be specified. When specified, it is a nested list with `shape=(n_boxes, n_species)`. This is perhaps easier to explain with a few examples.

Consider a system with one simulation box and one species. If the initial structure provided in `box_list` contains 100 molecules of that species, then:

```
mols_in_boxes = [[100]]
```

For a system with one simulation box and two species, where there are 25 molecules of the first species and 75 molecules of the second species:

```
mols_in_boxes = [[25, 75]]
```

For a system with two simulation boxes and one species, where the first box contains 100 molecules and the second box is empty:

```
mols_in_boxes = [[100], [0]]
```

For a system with two boxes and two species; the first box has 300 molecules of the first species and 50 molecules of the second species, the second box has 30 molecules of the first species and 100 molecules of the second:

```
mols_in_boxes = [[300, 50], [30, 100]]
```

When the `System` object is created, it verifies that the number of atoms provided in each box match the number of atoms specified by `mols_in_boxes`. The number of atoms per molecule are determined from the species provided in the `species_list`.

7.8.4 mols_to_add

Cassandra can insert molecules in a simulation box prior to starting an MC simulation. Therefore, you can provide an empty simulation box and request Cassandra to add some number of molecules before beginning the simulation. This capability is controlled through the `mols_to_add` option. The format of `mols_to_add` is analogous to `mols_in_boxes`. If specified, it is provided as a nested list with `shape=(n_boxes, n_species)`.

For example, consider a system with a single simulation box and two species. If we wish to add 10 molecules of the first species and 0 molecules of the second species, we could use:

```
mols_to_add = [[10,0]]
```

Warning: If `mols_to_add` is too large for the given box/species, the MC simulation may never begin. Cassandra will be stuck attempting (and failing) to insert the requested number of molecules.

7.9 MoveSet

The `MoveSet` contains all the information related to the Monte Carlo moves that will be attempted during the simulation. This includes the types of moves, the probability of selecting each move type, and the other related choices, such as the maximum translation distance, maximum volume move size, configurational biasing options, etc. The desired ensemble and the species in the system are used to assign default values to all of the attributes of the `MoveSet`. Nonetheless, most attributes can be edited once the object has been created.

The `MoveSet` is one of two objects that must be created prior to running a MC simulation in MoSDeF Cassandra. Creating the `MoveSet` requires specification of the following:

- The desired ensemble
- A list of the unique chemical species in the system (species_list)

Instantiating the `MoveSet` normally appears as follows:

```
import mosdef_cassandra as mc
moveset = mc.MoveSet("nvt", species_list)
```

7.9.1 Attributes

The `MoveSet` contains attributes that can be grouped into the following four categories.

Overall attributes, specified as a single number for the entire system:

- `ensemble` - ensemble of the MC simulation (`nvt`, `npt`, `gcmc`, `gemc`, `gemc_npt`)
- `prob_translate` - probability of attempting a translation move
- `prob_rotate` - probability of attempting a rotation move
- `prob_angle` - probability of attempting an angle change move
- `prob_dihedral` - probability of attempting a dihedral change move
- `prob_regrow` - probability of attempting a regrowth move
- `prob_volume` - probability of attempting a volume change move
- `prob_insert` - probability of attempting a molecule insertion move
- `prob_swap` - probability of attempting a molecule swap move
- `max_volume` - maximum volume move size (except for `gemc_npt`, where it is optionally per-box)
- `cbmc_n_insert` - number of locations to attempt a CBMC insertion
- `cbmc_n_dihed` - number of dihedral angles to attempt when regrowing a molecule with CBMC
- `cbmc_rcut` - cutoff to use when calculating energies during CBMC trials (optionally specified per-box)

Attributes specified per-species:

- `insertable` - boolean, is species insertable
- `prob_regrow_species` - probability of attempting a regrowth move with each species
- `prob_swap_species` - probability of attempting a swap move with each species
- `max_dihedral` - maximum dihedral angle change for dihedral change move

Attributes specified per-box:

- `prob_swap_from_box` - probability of selecting each box as donor for a swap move

Attributes specified per-box-per-species:

- `max_translate` - maximum translation distance
- `max_rotate` - maximum rotation angle

7.9.2 Printing the contents of the MoveSet

Imagine we have created a `MoveSet` as follows:

```
moveset = mc.MoveSet('nvt', species_list)
```

We can then print the current contents with:

```
moveset.print()
```

Example output for a single species (OPLS-AA methane):

```
Ensemble:  nvt

Probability of selecting each move type:

    Translate: 0.33
    Rotate:    0.33
    Regrow:    0.34
    Volume:    0.0
    Insert:    0.0
    Delete:    0.0
    Swap:      0.0
    Angle:     0.0
    Dihedral:  0.0

CBMC selections:

    Number of trial positions: 10
    Number of trial dihedral angles: 10
    CBMC cutoff(s):
        Box 1: 6.0


Per species quantities:

                             species1
                             ========
    Max translate (Ang):     2.00          (Box 1)
    Max rotate (deg):        30.00         (Box 1)
    Insertable:              False
    Max dihedral:            0.00
    Prob swap:               0.00
    Prob regrow:             1.00


Max volume (Ang^3):
    Box 1: 0.0
```

7.9.3 Default values for attempting each move type

`prob_translate`, `prob_rotate`, `prob_angle`, `prob_dihedral`, `prob_regrow`, `prob_volume`, `prob_insert`, and `prob_swap` are the probabilities of selecting each of those respective move types. The default move probabilities are as follows for each ensemble. Move probabilities that are not explicitly defined have a default probability of 0.0 for that ensemble.

NVT:

- `prob_translate = 0.33`
- `prob_rotate = 0.33`
- `prob_regrow = 0.34`

NPT:

- `prob_translate = 0.33`
- `prob_rotate = 0.33`
- `prob_regrow = 0.335`
- `prob_volume = 0.005`

GCMC:

- `prob_translate = 0.25`
- `prob_rotate = 0.25`
- `prob_regrow = 0.30`
- `prob_insert = 0.1`

Note: In GCMC the deletion probability is set equal to the insertion probability, making the sum of the move probabilities 1.0

GEMC:

- `prob_translate = 0.30`
- `prob_rotate = 0.30`
- `prob_regrow = 0.295`
- `prob_swap = 0.1`
- `prob_volume = 0.005`

GEMC-NPT:

- `prob_translate = 0.30`
- `prob_rotate = 0.30`
- `prob_regrow = 0.295`
- `prob_swap = 0.1`
- `prob_volume = 0.005`

7.9.4 Default values for other quantities

- max_translate: 2.0 Angstroms
- max_rotate : 30.0 degrees
- max_volume : 500 Angstroms^3 for Box 1, 5000 Angstroms^3 for Box 2
- max_dihedral : 0.0 degrees
- cbmc_n_insert : 10
- cbmc_n_dihed : 10
- cbmc_rcut : 6.0 Angstroms

max_translate and max_rotate are specified per-box-per-species. For example, if the system contained two species and the ensemble was GEMC (a two-box ensemble), then the default max translate would be [[2.0,2.0],[2.0,2.0]]. To set the max translation distance of species 1 in box 2 to 30.0 Angstroms, set max_translate = [[2.0,2.0],[30.0,2.0]].

Note: Exceptions to the above values are implemented based upon the topologies provided in species_list. The maximum rotation of single particle species is set to 0.0 degrees. Species that are multi-particle but contain zero bonds are considered fixed and not insertable; the maximum translation and rotation are set to 0.0 Angstroms and 0.0 degrees, respectively.

7.10 Keyword Arguments

Nearly all options of MoSDeF Cassandra can be controlled through the use of keyword arguments to the run/restart functions. These arguments can be specified individually or provided to the run/restart functions via a dictionary. The dictionary-based approach is preferred if there are a large number of keyword arguments to keep the number of explicit arguments to the run/restart functions manageable.

7.10.1 Usage

Below is an example of providing the vdw_cutoff option to run as an extra keyword argument.

```
mc.run(
  system=system,
  moveset=moveset,
  run_type="equilibration",
  run_length=1000,
  temperature=300.0 * u.K,
  vdw_cutoff=9.0 * u.angstroms
)
```

or as a dictionary, where the ** operator is used to expand the dictionary.

```
custom_args = {
  'vdw_cutoff': 9.0 * u.angstroms,
  'charge_cutoff': 9.0 * u.angstroms,
}
```

(continues on next page)

(continued from previous page)

```
mc.run(
  system=system,
  moveset=moveset,
  run_type="equilibration",
  run_length=1000,
  temperature=300.0 * u.K,
  **custom_args
)
```

7.10.2 Valid arguments

A list of the valid keyword arguments is provided below with a brief explanation. If more detail is required, please consult the Cassandra user manual. Most arguments below have nearly a one-to-one mapping with options of the Cassandra input file.

`run_name`

Type: `str`
Description: run name to prepend to output files
Default: ensemble of simulation (i.e., `"nvt"`, `"npt"`, etc.)

`restart`

Type: `bool`
Description: if `True`, restart from a Cassandra `.chk` file
Default: `False`

`restart_name`

Type: `str`
Description: name of the checkpoint file without the extension, i.e., the `run_name` for the simulation from which you wish to restart
Default:
Notes: only relevant if `restart=True`

`verbose_log`

Type: `bool`
Description: if `True`, print the Cassandra log file with additional verbosity
Default: `False`

vdw_style

Type: str
Description: type of van der Waals interactions. Valid options include lj or none
Default: lj

cutoff_style

Type: str
Description: method of handling the cutoff for the van der Waals interactions. Valid options include cut_tail, cut_switch, cut_shift
Default: cut_tail

vdw_cutoff

Type: unyt_quantity, dimensions=length, except for cutoff_style="cut_switch", which requires a list of, [inner_cutoff, outer_cutoff].
Description: cutoff distance for van der Waals interactions
Default: 12.0 * u.angstroms
Notes: In a system with multiple boxes, per box values can be specified with vdw_cutoff_box1 and vdw_cutoff_box2 keywords. If provided, these will override the vdw_cutoff.

charge_style

Type: str
Description: method of computing electrostatic energy, options include none, ewald, or dsf
Default: ewald

charge_cutoff

Type: unyt_quantity, dimensions=length
Description: cutoff distance for short-range portion of charged interactions
Default: 12.0 * u.angstroms
Notes: In a system with multiple boxes, per box values can be specified with charge_cutoff_box1 and charge_cutoff_box2 keywords. If provided, these will override the charge_cutoff. In GEMC simulations where the vapor box is much larger than the liquid box, it may be necessary to increase the charge cutoff of the vapor box to maintain the desired ewald_accuracy without exceeding the maximum number of k-space vectors.

ewald_accuracy

Type: float
Description: relative accuracy of ewald summation
Default: 1.0e-5
Notes: Only relevant if charge_style="ewald"

dsf_damping

Type: float
Description: damping parameter for dsf charge style
Default: None
Notes: Only relevant if charge_style="dsf"

mixing_rule

Type: str
Description: the type of mixing rule to apply to van der Waals interactions. Options include lb (Lorentz-Berthelot), geometric or custom
Default: lb

custom_mixing_dict

Type: dict
Description: dictionary specifying the custom mixing rules. One key-value pair is specified per pair of atomtypes. The key is a string of the species combination, and the value is a list of the relevant parameters. For example, the two atom types are opls_140 and opls_141 and the mixed epsilon and sigma are 10.0 * u.Unit('kJ/mol') and 3.0 * u.angstrom, then the dict would be:

```
{ 'opls_140 opls_141': [10.0 * u.Unit('kJ/mol'), 3.0 * u.angstrom] }
```

Default: None

seeds

Type: list of two ints
Description: the starting seeds for the random number generator.
Default: selected at random

`rcut_min`

Type: `unyt_quantity, dimensions = length`

Description: minimum distance to calculate interaction energy. If particles are closer than this distance the energy is taken as infinity and the move is automatically rejected. If the value is too large moves that might possibly be accepted will be unecessarily rejected.

Default: `1.0 * u.angstrom`

`pair_energy`

Type: `bool`

Description: store pair interactions energies (requires more memory but may be faster)

Default: `True`

`max_molecules`

Type: `list` of `ints`, `len=n_species`

Description: maximum number of molecules of each species. Cassandra will exit if the number of molecules of a species exceeds this number at any point during a simulation.

Default: Number of molecules in the `System` for `nvt`, `npt`, `gemc`, `gemc_npt`, and non-insertable species in `gcmc`. Number of molecules in the `System` plus 500 for insertable molecules in `gcmc`.

Notes: The default may need to be overridden in GCMC if the initial configuration has many fewer molecules than at equilibrium.

`pressure`

Type: `unyt_quantity`, valid units of pressure

Description: desired pressure (NPT or GEMC-NPT) ensembles

Default: `None`

Notes: in GEMC-NPT, different pressures for `box1` and `box2` can be specified with the `pressure_box1` and `pressure_box2`. If specified, these values will override the value in `pressure`.

`chemical_potentials`

Type: `list` of `unyt_array/unyt_quantity` with units of `energy/mol`, or `"none"` for species that are not insertable

Description: specify the desired chemical potential for each species (`gcmc`)

Default: `None`

thermal_stat_freq

Type: `int`

Description: frequency, in number of thermal moves, of printing statistics and (if `run_type="equilibration"`), updating the maximum translation and rotation sizes

Default: `1000`

Notes: in `equilibration` mode, the maximum translation and rotation move sizes are continuously adjusted to target 50% of moves accepted.

vol_stat_freq

Type: `int`

Description: frequency, in number of volume moves, of printing statistics and (if `run_type="equilibration"`), updating the maximum volume move size

Default: `100`

Notes: in `equilibration` mode, the maximum volume move size is continuously adjusted to target 50% of moves accepted.

units

Type: `str`

Description: units for measuring simulation length, valid options include `minutes`, `steps`, or `sweeps`

Default: `steps`

steps_per_sweep

Type: `int`

Description: the number of MC steps in one MC sweep

Default: `None`

Notes: required if `units="steps"`. A standard choice is one sweep is one attempted move per molecule in the system.

prop_freq

Type: `int`

Description: frequency of writing thermo properties to the `.prp` file

Default: 500

Notes: units determined by the `units` argument

coord_freq

Type: int
Description: frequency of writing system coordinates to the .xyz file
Default: 5000
Notes: units determined by the units argument

block_avg_freq

Type: int
Description: block average size
Default: None
Notes: units determined by the units argument

properties

Type: list of str
Description: list of properties to write to the .prp file. Valid options include: energy_total, energy_intra, energy_bond, energy_angle, energy_diheral, energy_improper, energy_intravdw, energy_intraq, energy_inter, energy_intervdw, energy_lrc, energy_interq, energy_recip, energy_self, enthalpy, pressure, pressure_xx, pressure_yy, pressure_zz, volume, nmols, density, mass_density.
Default: ["energy_total", "energy_intra", "energy_inter", "enthalpy", "pressure", "volume", "nmols", "mass_density"]

7.11 Run Monte Carlo

To run a Monte Carlo simulation, use:

```
mc.run(
  system=system,
  moveset=moveset,
  run_type="equilibration",
  run_length=1000,
  temperature=300.0 * u.K
)
```

The run function has five required arguments: a System, MoveSet, a choice of run_type, the run_length, and the temperature. Other optional arguments can be specified individually or with a dictionary. For example, if we were performing and NPT simulation and needed to specify the pressure, we could do the following:

```
mc.run(
  system=system,
  moveset=moveset,
  run_type="equilibration",
  run_length=1000,
  temperature=300.0 * u.K,
  pressure=1.0 * u.bar
)
```

or, if we wished to use a dictionary:

```
custom_args = {
  'pressure' : 1.0 * u.bar
}

mc.run(
  system=system,
  moveset=moveset,
  run_type="equilibration",
  run_length=1000,
  temperature=300.0 * u.K,
  **custom_args
)
```

The dictionary-based approach is easier to read when specifying a larger number of custom options. For example:

```
custom_args = {
  'pressure' : 1.0 * u.bar,
  'cutoff_style' : 'cut_shift',
  'vdw_cutoff' : 14.0 * u.angstrom,
  'units' : 'sweeps',
  'prop_freq' : 10,
  'coord_freq' : 100
}

mc.run(
  system=system,
  moveset=moveset,
  run_type="equilibration",
  run_length=1000,
  temperature=300.0 * u.K,
  **custom_args
)
```

7.12 Restart a Simulation

MoSDeF Cassandra also supports restarting from a Cassandra checkpoint file. The checkpoint file contains the coordinates, box information, and state of the random number generator required for an exact restart. In order for the restart to work correctly, the directory must contain: (1) the original Cassandra input (.inp) file, (2) the Cassandra MCF files and fragment libraries, (3) the checkpoint (.chk) file that will be used for the restart.

Note: It is easiest if the restart is performed from within the same directory as the original simulation. If no files have been deleted since the original run, all of the required items should be present.

The restart function accepts four arguments: (1) the total simulation length, (2) the prefix for the files you wish to use for the restart, `restart_from`, (3) the prefix for the files generated by the new simulation, `run_name`, and (4) the `run_type`, “equilibration” or “production”. Depending on the specific use-case, some or all of the arguments may be optional.

There are a few scenarios where it is useful to use the restart capability.

7.12.1 Switch from equilibration to production

One of the most common use-cases for the restart function is switching from an equilibration to production simulation. In equilibration mode, Cassandra actively adjusts the maximum translation, rotation, and volume move sizes to achieve a 50% acceptance ratio. In production mode, the maximum move sizes are fixed. If we use a restart and switch from equilibration to production mode, Cassandra will take the optimized translation, rotation, and volume move sizes from the checkpoint file.

```
mc.run(
    system=system,
    moveset=moveset,
    run_type="equilibration",
    run_length=1000,
    temperature=300.0 * u.K,
    run_name="equil",
)

mc.restart(
    restart_from="equil",
    run_name="prod",
    run_type="production",
    total_run_length=2000
)
```

Note that the `total_run_length` is the sum of the equilibration and production run lengths – so in this example we are running a 1000 MC step production following a 1000 MC step equilibration.

7.12.2 Restart a simulation that has not completed

Sometimes a simulation is terminated prematurely. In this case, the goal is to restart the simulation from the checkpoint file and complete the original simulation. Here, we can simply use:

```
mc.restart()
```

The new `run_name` will be the original with `.rst.001` appended. If there are multiple `.inp` files in the current directory, you will need to specify the `restart_from` option. E.g., if the current directory contains both `equil.inp` and `prod.inp` and we wish to restart the simulation created by `prod.inp`:

```
mc.restart(restart_from="prod")
```

The new `run_name` will be `prod.rst.001`.

7.12.3 Extend a simulation

Sometimes it is necessary to extend a simulation. In this case, we must specify the `total_run_length`. Once again, note this is the *total* number of simulation steps. For example, imagine our initial simulation is 1000 steps:

```
mc.run(
    system=system,
    moveset=moveset,
    run_type="equilibration",
    run_length=1000,
```

(continues on next page)

(continued from previous page)

```
    temperature=300.0 * u.K,
    run_name="example",
)
```

Now we wish to extend the simulation by an additional 1000 steps. We use:

```
mc.restart(total_run_length=2000)
```

If we needed to again extend the simulation by 1000 steps:

```
mc.restart(total_run_length=3000)
```

The prefix for the files from the three simulations would be `example`, `example.rst.001`, and `example.rst.002`.

We could alternatively manually specify the `run_name` for the extended simulations if we wished:

```
mc.restart(
    restart_from="example",
    run_name="my_example_restart",
    total_run_length=2000,
)
```

7.13 API Documentation

`class mosdef_cassandra.`**`System`**(*boxes*, *species_topologies*, *mols_in_boxes=None*, *mols_to_add=None*, *fix_bonds=True*)

`__init__`(*self*, *boxes*, *species_topologies*, *mols_in_boxes=None*, *mols_to_add=None*, *fix_bonds=True*)

A class to contain the system to simulate in Cassandra

A System comprises the initial simulation box(es) (empty or occupied), the topologies of each species to be simulated, and the number of each species to be added to the simulation box(es) prior to the start of the simulation. These three items are represented by `boxes`, `species_topologies`, and `mols_to_add`. If providing a box with existing species, you are required to specify `mols_in_boxes`, the number of each species that already exists.

Each argument is specified as a list, with either one element for each box or one element for each species. Arguments must be provided as a list even in the case of a single species or single box.

Parameters

- **boxes** (*list*) – one element per box. Each element should be a mbuild.Compound or mbuild.Box
- **species_topologies** (*list*) – list of parmed.Structures, with one species per element
- **mols_in_boxes** (*list, optional*) – one element per box. Each element is a list of length n_species, specifying the number of each species that are currently in each box
- **mols_to_add** (*list, optional*) – one element per box. Each element is a list of length n_species, specifying the number of each species that should be added to each box
- **fix_bonds** (*boolean, optional, default=True*) – update the bond lengths in any initial structure (i.e., boxes) to match the values specified in the species_topologies

Return type *mosdef_cassandra.System*

property **boxes**(*self*)

check_natoms(*self*)
Confirm that the number of existing atoms in each box agrees with the number of atoms specified from the combination of the number of atoms in each species and the number of each species in the box.

fix_bonds(*self*)
Apply the bond length constraints to each molecule in the system

property **mols_in_boxes**(*self*)

property **mols_to_add**(*self*)

property **species_topologies**(*self*)

class mosdef_cassandra.**MoveSet**(*ensemble*, *species_topologies*)

__init__(*self*, *ensemble*, *species_topologies*)
A class to contain all the move probabilities and related values required to perform a simulation in `Cassandra`.

A MoveSet contains the move probabilities and other related quantities (e.g., max translation/rotation) that are required to run Cassandra. When the MoveSet is created the specified `ensemble` and `species_topologies` are used to generate initial guesses for all required values. Depending upon the specifics of your system, these guesses may be very reasonable or downright terrible. Use the same `species_topologies` for your call to `mosdef_cassandra.System()` and `mosdef_cassandra.MoveSet()`.

Parameters

- **ensemble** (*str*) – string describing the desired ensembled. Supported values include `'nvt'`, `'npt'`, `'gcmc'`, `'gemc'`, `'gemc_npt'`
- **species_topologies** (*list*) – list of `parmed.Structures`, with one species per element

Return type `mosdef_cassandra.MoveSet`

add_restricted_insertions(*self*, *species_topologies*, *restricted_type*, *restricted_value*)
Add restricted insertions for specific species and boxes

Parameters

- **species_topologies** (*list*) – list of `parmed.Structures` containing one list per box of species
- **restricted_type** (*list*) – list of restricted insertion types containing one list per box of species
- **restricted_value** (*list*) – list of restricted insertion values (unyt arrays) containing one list per box of species

property **cbmc_n_dihed**(*self*)

property **cbmc_n_insert**(*self*)

property **cbmc_rcut**(*self*)

property **ensemble**(*self*)

property **insertable**(*self*)

property **max_dihedral**(*self*)

property **max_rotate**(*self*)

property `max_translate`(*self*)

property `max_volume`(*self*)

`print`(*self*)
Print the current contents of the MoveSet

property `prob_angle`(*self*)

property `prob_dihedral`(*self*)

property `prob_insert`(*self*)

property `prob_regrow`(*self*)

property `prob_regrow_species`(*self*)

property `prob_rotate`(*self*)

property `prob_swap`(*self*)

property `prob_swap_from_box`(*self*)

property `prob_swap_species`(*self*)

property `prob_translate`(*self*)

property `prob_volume`(*self*)

`mosdef_cassandra.run`(*system*, *moveset*, *run_type*, *run_length*, *temperature*, ***kwargs*)
Run the Monte Carlo simulation with Cassandra

The following steps are performed: write the molecular connectivity files for each species to disk, write the starting structures (if any) to disk, generate and write the Cassandra input file to disk, call Cassandra to generate the required fragment libraries, and call Cassandra to run the MC simulation.

Parameters

- **system** (*mosdef_cassandra.System*) – the System to simulate
- **moveset** (*mosdef_cassandra.MoveSet*) – the MoveSet to simulate
- **run_type** (*"equilibration" or "production"*) – the type of run; in "equilibration" mode, Cassandra adaptively changes the maximum translation, rotation, and volume move sizes to achieve an acceptance ratio of 0.5
- **run_length** (*int*) – length of the MC simulation
- **temperature** (*float*) – temperature at which to perform the MC simulation
- ****kwargs** (*keyword arguments*) – any other valid keyword arguments, see `mosdef_cassandra.print_valid_kwargs()` for details

`mosdef_cassandra.restart`(*total_run_length=None*, *restart_from=None*, *run_name=None*, *run_type=None*)
Restart a Monte Carlo simulation from a checkpoint file with Cassandra

The function requires the following in the working directory. These items would have all been generated for the original run:

- Cassandra input (.inp) file named {restart_from}.inp
- Cassandra checkpoint file (.chk) name {restart_from}.out.chk
- MCF files for each species
- Fragment libraries for each species

The maximum translation, rotation, and volume move sizes are read from the checkpoint file. Similarly, the starting structure is taken from the checkpoint file. If the "restart_name" is not provided or if the "run_name" is the same as "restart_name", ".rst.N" will be appended to the "run_name".

If you wish to extend a simulation you will need to specify the _total_ number of simulation steps desired with the total_run_length option. For example, if your original run was 1e6 MC steps, but you wish to extend it by an additional 1e6 steps, use total_run_length=2000000.

Parameters

- **total_run_length** (*int, optional, default=None*) – total length of the MC simulation; if None, use original simulation length
- **restart_from** (*str, optional, default=None*) – name of run to restart from; if None, searches current directory for Cassandra inp files
- **run_name** (*str, optional, default=None*) – name of this run; if None, appends ".rst.NNN." to run_name, where "NNN" is the restart iteration "001", "002", . . . ,
- **run_type** (*str, "equilibration" or "production", default=None*) – the type of run; in "equilibration" mode, Cassandra adaptively changes the maximum translation, rotation, and volume move sizes to achieve an acceptance ratio of 0.5. If None, use the same choice as the previous run.

`mosdef_cassandra.`**`print_valid_kwargs`**`()`

Print the valid keyword arguments with a brief description

`mosdef_cassandra.`**`print_inputfile`**`(`*system*, *moveset*, *run_type*, *run_length*, *temperature*, ***kwargs*`)`

Print an example Cassandra input file to screen

This function allows one to look at the Cassandra input file that will be generated without running the MC simulation. The arguments are identical mosdef_cassandra.run

Parameters

- **system** (*mosdef_cassandra.System*) – the System to simulate
- **moveset** (*mosdef_cassandra.MoveSet*) – the Move set to simulate
- **run_type** (*"equilibration" or "production"*) – the type of run; in "equilibration" mode, Cassandra adaptively changes the maximum translation, rotation, and volume move sizes to achieve an acceptance ratio of 0.5
- **run_length** (*int*) – length of the MC simulation
- **temperature** (*float*) – temperature at which to perform the MC simulation
- ****kwargs** (*keyword arguments*) – any other valid keyword arguments, see `mosdef_cassandra.print_valid_kwargs()` for details

`class mosdef_cassandra.analysis.`**`ThermoProps`**`(`*filename*`)`

Store thermodynamic properties from a Cassandra .prp file

`__init__``(`*self*, *filename*`)`

Create ThermoProps from a .prp file

Parameters **filename** (*string*) – path to the .prp file

Returns object containing the contents of the .prp file

Return type *ThermoProps*

`property` **`filename`**`(`*self*`)`

print_props(*self*)
Print the available properties

prop(*self*, *prp_name*, *start=None*, *end=None*, *units=True*)
Extract the specified property

Parameters

- **prp_name** (*string*) – the property to extract
- **start** (*int*) – the starting step/sweep/etc.
- **end** (*int*) – the ending step/sweep/etc.

Returns the property with units

Return type unyt_array

to_df(*self*)
Convert ThermoProps to a pandas.DataFrame

7.14 Contributing

We welcome contributions to MoSDeF Cassandra. If you wish to contribute you can find us on GitHub.

7.15 Citing MoSDeF Cassandra

If you use MoSDeF Cassandra in your research, please cite the following.

The first is the publication describing the Cassandra Monte Carlo package:

```
@article{cassandra,
  title={Cassandra: An open source Monte Carlo package for molecular simulation},
  author={Shah, Jindal K and
          Marin-Rimoldi, Eliseo and
          Mullen, Ryan Gotchy and
          Keene, Brian P and
          Khan, Sandip and
          Paluch, Andrew S and
          Rai, Neeraj and
          Romanielo, Lucienne L and
          Rosch, Thomas W
          Yoo, Brian and
          Maginn, Edward J},
  journal={Journal of Computational Chemistry},
  volume={38},
  number={19},
  pages={1727--1739},
  year={2017},
  publisher={Wiley Online Library}
}
```

and the second is for the MoSDeF Cassandra package:

```
@misc{OpenSourceSurvey17,
  author = {DeFever, Ryan S and Maginn, Edward J},
  title = {MoSDeF Cassandra},
  year = {2020},
  journal = {GitHub repository},
  howpublished = {\url{https://github.com/MaginnGroup/mosdef_cassandra}}
}
```

Please also consider citing references associated with the MoSDeF tools. If your workflows resemble any of the examples, you should cite the relevant references for mbuild and foyer. Some possible references can be found here.

7.16 License

7.17 Credits

MoSDeF Cassandra developers:

- Ryan DeFever - **Creator and lead developer**
- Ray Matsumoto - **Developer**

MoSDeF Cassandra was developed in close collaboration with the MoSDeF team.

MoSDeF Cassandra uses the Cassandra Monte Carlo package for all Monte Carlo calculations. Cassandra was developed by the Maginn group at the University of Notre Dame. Cassandra can be found here and is distributed under the GNU GPL license.

Development of MoSDeF Cassandra was supported by the National Science Foundation under grant NSF Grant Number 1835874. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search