enu Manual

Contents

1 Introduction

The open-source isomer enumerator *enu* is part of the CombiFF combinatorial force-field parameterization scheme to automatically parameterize force fields for condensed-phase molecular dynamics (MD) simulations.[1,2] The source code of *enu* is written in C++ and can be compiled with *cmake*. The program enumerates the constitutional and spatial isomers of a given molecular formula. The theoretical background of *enu* will be documented in Ref. 3, while the present document provides an introduction to the practical usage of the program.

2 Installation

For installation instructions, please refer to `https://github.com/csms-ethz/CombiFF/blob/main/INSTALL.md`.

3 Basic Usage

This section describes the "basic" use of *enu*, *i.e.*, enumerating the constitutional and (if desired) spatial isomers of a given molecular formula and reporting the detected (stereo)isomers as canonical SMILES strings in an XML format. Usage examples are provided in Sec. 3.2. The use of *enu* in the context of CombiFF to enumerate molecular "families" is described in Sec. 4.

3.1 Command Line Arguments

3.1.1 Input/Output

argument	functionality
`-input`	specify an input file containing one or several arguments recognized by *enu* • these arguments can also be used directly in the command line • lines starting with `#` in the file are comments and are ignored
`-output_directory`	specify the directory where the output files should be created
`-output`	specify the file name of the output file (*i.e.*, the XML file containing the canonical SMILES strings)

3.1.2 Molecular Formula

argument	functionality
`-formula`	specify the molecular formula • this arguments may be used several times to specify multiple molecular formulas

The basic format to specify a molecular formula is a sequence of alternating element types and corresponding counts. The following syntax may be used for an element type `X` and numbers `n`, `m`, `l`:

formula building block	functionality
`Xn`	`n` counts of element type `X`
`X[n-m]`	range from `n` to `m` counts of element type `X`
`X[n,m,l,...]`	`n`, `m`, `l`, ... counts of element type `X`
{`XHn`}	atom type `X` with `n` implicit hydrogens. To specify the count of {`XHn`}, the same syntax can be used as for a simple atom type (*i.e.*, simple count, range, or list)

Note that the notation for ranges and the notation for lists of counts may be combined, *e.g.*, `X[n-m,k,l,i-j]`

3.1.3 Behavior

The following arguments can be used to modify what and how *enu* reports its results:

formula building block	functionality
`-stereo`	enumerate spatial isomers additionally to constitutional ones
`-count_only`	report the number of detected isomers but don't generate XML output
`-quiet`	don't report the current number of isomers during the enumeration process. **This is especially useful when re-directing the standard output to a file**, as the `'\r'` carriage return only works in terminals, and the output will be too verbose if written to a file.

3.1.4 Restrictions

The user can specify restrictions on the molecules that should be counted/reported. The following syntax can be used for numbers `n` and `m`:

argument	functionality
`-max_bond_degree n`	maximum bond degree between two atoms is set to `n` (default: 4)
`-unsaturations n`	`n` unsaturations
`-unsaturations [n-m]`	range of `n` to `m` unsaturations
`-unsaturations *`	no restriction on number of unsaturations (default)
`-total_bonds n`	in total `n` bonds (irrespective of bond multiplicity)
`-total_bonds [n-m]`	range of `n` to `m` bonds
`-total_bonds *`	no restriction on total number of bonds (default)
`-single_bonds n`	`n` single bonds
`-single_bonds [n-m]`	range of `n` to `m` single bonds
`-single_bonds *`	no restriction on total number of single bonds (default)
`-double_bonds n`	`n` double bonds
`-double_bonds [n-m]`	range of `n` to `m` double bonds
`-double_bonds *`	no restriction on total number of double bonds (default)
`-triple_bonds n`	`n` triple bonds
`-triple_bonds [n-m]`	range of `n` to `m` triple bonds
`-triple_bonds *`	no restriction on total number of triple bonds (default)
`-cycles n`	`n` cycles
`-cycles [n-m]`	range of `n` to `m` cycles
`-cycles *`	no restriction on total number of cycles (default)

Note that the number of total/single/double/triple bonds is determined from the entries of the molecule's canonical adjacency matrix, the number of unsaturations d_{unsat} is calculated as

$$d_{\text{unsat}} = 1 + \frac{1}{2}\left(\sum_{k=0}^{K} \lambda_k(\delta_k - 2)\right) , \tag{1}$$

where δ_k is the valence of atom type α_k and λ_k is the count of atom type α_k, and the number of cycles n_{cycles} is calculated as

$$n_{\text{cycles}} = d_{\text{unsat}} - n_{\text{db}} - 2 \cdot n_{\text{tb}} , \tag{2}$$

where n_{db} is the number of double bonds, and n_{tb} is the number of triple bonds.

3.2 Usage Examples

call	result
`./enu -formula C19H42`	enumerate the constitutional isomers of $C_{19}H_{42}$. By default, the output is written to the file `isomerlist.xml`.
`./enu -formula C19H42 -quiet`	enumerate the constitutional isomers of $C_{19}H_{42}$. Don't report the current number of detected isomers during the enumeration process. By default, the output is written to the file `isomerlist.xml`.
`./enu -formula C19H42 -count_only`	enumerate the constitutional isomers of $C_{19}H_{42}$ without generating an output file containing the corresponding SMILES strings.
`./enu -formula C10H22 -stereo`	enumerate the constitutional and spatial isomers of $C_{10}H_{22}$. By default, the output is written to the file `isomerlist.xml`.
`./enu -formula C10H22 -output_directory out`	enumerate the constitutional isomers of $C_{10}H_{22}$ and create the output file in `out/isomerlist.xml`.
`./enu -formula C10H22 -output_directory out -output c10h22.xml`	enumerate the constitutional isomers of $C_{10}H_{22}$ and write the output to `out/c10h22.xml`.
`./enu -formula C[1-20]H[4-42] -unsaturations 0 -stereo`	enumerate the constitutional and spatial isomers of the straight-chain alkanes from C_1H_4 to $C_{20}H_{42}$.
`./enu -formula C5H[0-12]Br[0-2]Cl[0-2] -cycles 1 -unsaturations 1`	enumerate the constitutional isomers of all molecular formulas with five carbon atoms, zero to twelve hydrogen atoms, zero to two bromine atoms, and zero to two chlorine atoms that contain one cycle and one unsaturation.
`./enu -formula C[3,5]H[0-12]Br[1,3,5,7-10]`	enumerate the constitutional isomers of all molecular formulas with three or five carbon atoms, zero to twelve hydrogen atoms, and one, three, five, seven, eight, nine, or ten bromine atoms.
`./enu -formula {CH1}1{CH3}2O1C2{OH1}1H2`	enumerate the constitutional isomers of $C_5H_{10}O_2$ with one carbon atom connected to exactly one hydrogen, two carbon atoms connected to exactly three hydrogens, one oxygen atom without a restriction on the number of connected hydrogens, two carbon atoms without a restriction on the number of connected hydrogens, one oxygen atom connected to exactly one hydrogen, and two hydrogen atoms (additionally to the implicit hydrogens).

4 Use of *enu* for CombiFF: Molecular Families

For the application of *enu* for the CombiFF force-field parameterization scheme, the definition of so-called molecular families is an essential tool. Family definitions also offer more flexibility to specify the molecules of interest using so-called *element aliases* and *pseudoatoms* and by allowing the filtering of substructures.[3] For the convenience of the user, the CombiFF repository contains additional python scripts, most importantly `scr/run_enu.py` that can be used to automatically run *enu* for the families defined in the CombiFF repository (see also `https://github.com/csms-ethz/CombiFF#scripts`).

In the following sections, first the command line arguments are explained that may be used to pass the arguments related to molecular families to *enu*, then the file formats of the respective input files and output files are outlined, and finally some usage examples are provided.

4.1 Command Line Arguments

formula building block	functionality
`-families`	specify the families that should be enumerated • specified *via* the family code from the family definition XML • may be used several times to specify multiple families • multiple family codes may be listed consecutively
`-family_files`	specify the path to the family definition input file(s) • multiple file paths may be listed consecutively
`-element_alias_files`	specify the path to the element alias input file(s) • multiple file paths may be listed consecutively
`-pseudoatom_files`	specify the path to the pseudoatom input file(s) • multiple file paths may be listed consecutively
`-substructure_files`	specify the path to the substructure input file(s) • multiple file paths may be listed consecutively

4.2 Family Definition Files

The family definition files are located in the `use/input_files/family_definitions` directory and follow an XML structure. The directory contains several examples of family definition files that have been used in the context of CombiFF. The directory also contains the DTD file defining the format of a family definition file. A family definition file should follow the following format:

```
<family_definitions version="[version number]">
  <!--
    [comment describing the family]
  -->
  <family_definition code="[family code]">
    <formula>[formula, e.g. C10H22]</formula>
  </family_definition>
  ...
</family_definitions>
```

Analogous to the `-formula` command line argument of *enu*, the formula entry in a family definition consists of alternating element types and corresponding counts.

An element type may be specified *via*

- directly as an element type (*e.g.*, `C`)
- including implicit hydrogens (*e.g.*, {`CH2`})
- as an element alias (*e.g.* `Hal`) → see Sec. 4.3
- as a pseudoatom in single quotes (*e.g.* `'C=O'`) → see Sec. 4.4

An element count may be specified *via*

- a simple count (*e.g.*, `C10`)
- a range in square brackets (*e.g.*, `C[8-10]`)
- a list of numbers in square brackets (*e.g.*, `C[8,10,12]`)
- a mix of lists and ranges in square brackets (*e.g.*, `C[1-2,3,5,7-10,15]`)

For element aliases, the following additional syntax may be used to specify various combinations of the members of an element alias set. For example if `Hal` corresponds to the set of {`Br,Cl,F,I`}:

- AND: the same element type is used
  - `Hal[1-5]` corresponds to one to five elements of a `Hal` type, *e.g.*, `Br[1-5]`
- OR: the same or a different element type is used
  - `Hal2Hal1` corresponds to two `Hal` elements and one `Hal` elements without any restrictions, *e.g.* `Br2Cl1` or `Br3`
- XOR: different element types should be used, specified by `^`
  - `^Hal2^Hal1` corresponds to two `Hal` elements of one type and one `Hal` element of a different type, *e.g.* `Br2Cl1`
  - `^Hal[4-5]^Hal3^Hal1` corresponds to four to five `Hal` elements of one type, three `Hal` elements of a different type, and one `Hal` element of yet another type, *e.g.* `Br5Cl3F1`

After the `formula` entry, restrictions can be defined (same principle as for the command line arguments described in Sec. 3.1.4):

- `max_bond_degree`
- `unsaturations`
- `single_bonds`
- `double_bonds`
- `triple_bonds`
- `cycles`

As for the command line arguments, the restrictions can be a specific number `n` or a range in rectangular brackets `[n-m]`. The XML syntax to specify, for example, one to two unsaturations and zero cycles is:

```
<unsaturations>[1-2]</unsaturations>
<cycles>0</cycles>
```

Finally, substructures may be specified that should or should not occur in the molecule using the substructure code (→ see Sec. 4.5). In a CombiFF substructure, only the connectivity of the

molecular graph nodes and the multiplicity of the graph edges are defined, but not the element types of the nodes. The element types of the nodes are specified in the family definition file using the following format:

```
<substructures>
  <substructure substructure_code="[substructure code]" number="[number, given
   as number or range in rectangular brackets]">
    <atoms>
      <atom type="[atom type 1]"/>
      <atom type="[atom type 2]"/>
      ...
    </atoms>
  </substructure>
  ...
</substructures>
```

Here, an atom type may be

- directly as an element type (*e.g.*, `C`)
- including implicit hydrogens (*e.g.*, {`CH2`})
- as an element alias (*e.g.* `Hal`) → see Sec. 4.3
- a wildcard, specified by a `*`

If element aliases are used for the atom types, the following additional syntax may be used to specify the valid combinations of element types from the element alias set. For example if `Hal` corresponds to the set of {`Br,Cl,F,I`}:

- OR: any combination is allowed, no specified
  - `<atom type="Hal"/><atom type="Hal"/>` allows any combination of halogen element types
- AND: the same element type should be used, specified by `n&`, where the element types with the same number `n` have to be the same
  - `<atom type="&1Hal"/><atom type="&1Hal"><atom type="&2Hal"/><atom type="&2Hal"/>`: the element types with the `&1` specifier have to be the same and the element types with the `&2` specifier have to be the same. However, both the case where all four are the same, and the case where the first two are different from the second two is valid.
- XOR: different element types should be used, specified by `n^`, where the element types with the same number `n` have to be different from one another
  - `<atom type="^1Hal"/><atom type="^1Hal"/>` corresponds to two different halogen element types
  - `<atom type="^1Hal"/><atom type="^1Hal"/><atom type="^2Hal"/><atom type="^2Hal"/>`: the element types with the `^1` specifier should be different from each other and the element types with the `^2` specified should be different from each other. However, one of the element types with the `^1` specifier may be the same as one of the element types with the `^2` specifier (but does not have to be the same).

4.3 Element Alias Files

An element alias is a set of element types, *e.g.*, `Hal={Br,Cl,F,I}`.[3] Element alias files are contained in the `use/input_files/aliases` directory of the CombiFF repository. The directory also contains the DTD file defining the format of an element alias file. The XML structure of an element alias file follows the following format

```
<aliases version="[version number]">
  <alias name="[name of alias]">
    <member>[element symbol of member, e.g. H, F. Also united hydrogens such as
    CH3]</member>
    ...
    <aliasmember>[name of other alias (listed **before the current one** whose
    members should be included]</aliasmember>
    ...
  </alias>
  ...
</aliases>
```

4.4 Pseudoatom Files

A pseudoatom is a molecular fragment that only contains one atom that is not fully bonded.[3] During the enumeration, a pseudoatom is treated like a normal atom and can be used to define substructures. A pseudoatom is defined by an identifier (`code`), a list of element types, and an adjacency matrix stack. Pseudoatom files are contained in the `use/input_files/pseudoatoms` directory of the CombiFF repository. The directory also contains the DTD file defining the format of a pseudoatom file. The XML structure of a pseudoatom file follows the following format

```
<pseudoatoms version="[version number]">
  <pseudoatom code="[pseudoatom code]">
    <atoms>
      <atom>[atom type of 1st atom]</atom>
      ...
    </atoms>
    <adjacency_stack>[stack of adjacency matrix, which defines the connections
    of the above atoms]</adjacency_stack>
  </pseudoatom>
  ...
</pseudoatoms>
```

An atom type may be

- an element type, *e.g.* `C`
- an element type including implicit hydrogens, *e.g.* `CH3`

Note that adjacency matrix stacks can be conveniently created from SMILES strings with the CombiFF program `cnv`. For example, the adjacency matrix stack for the molecule `CCC(=O)C` can be generated by running `./cnv -O stack 'CCC(=O)C'`.

4.5 Substructure Files

Substructure files are contained in the `use/input_files/substructures` directory of the CombiFF repository. The directory also contains the DTD file defining the format of a substructures file. The XML structure of a substructures file follows the following format

```
<substructures version="[version number]">
  <substructure code="[unique substructure identifier code]">
    <num_atoms>[# atoms in substructure]</num_atoms>
    <adjacency_stack>[adjacency stack as numbers separated by spaces, e.g. 1 1 0
     0 1 1]</adjacency_stack>
  </substructure>
  ...
</substructures>
```

Note that the atom types are not specified, only the connectivity of the nodes and the multiplicity of the edges. The atom types can then be specified in the family definition (see Sec. 4.2). Note that adjacency matrix stacks can be conveniently created from SMILES strings with the CombiFF program `cnv`. For example, the adjacency matrix stack for the molecule `CCC(=O)C` can be generated by running `./cnv -O stack 'CCC(=O)C'`.

4.6 Usage

4.6.1 `run_enu.py`

The most convenient way to use *enu* to create family isomer enumeration files that can then be processed by *tbl* is *via* the Python script `run_enu.py` located in the `scr/` directory. We recommend that you create a `conda` (`https://www.anaconda.com/`) or `mamba` (`https://anaconda.org/conda-forge/mamba`) environment as follows to run the script:

```
conda env create -f dev/conda_envs/combiff.yml -n combiff
conda activate combiff
```

The location of the executables, as well as the input and output files is defined in `use/global_settings.py`. To control the behavior of the script, you can create a file `use/user_settings.py`. In this file you can overwrite any of the variables defined in `use/global_settings.py`, namely in the context of *enu*:

variable	functionality
`families_to_enumerate`	list of strings, containing the code of the families that should be enumerated by *enu*. If it contains the element `all`, all families will be enumerated. You may use regex such as `'01..'` or `'*1*'`. *e.g.*, `families_to_enumerate = ['01..', '0300']`
`family_exceptions`	list of strings, containing the code of the families that should be ignored by the scripts, even if `all` is specified for `families_to_enumerate`. You may use regex such as `'01..'` or `'*1*'`. *e.g.*, `family_exceptions = ['0102', '0104']`
`force_update`	boolean, specifying whether output files should be regenerated even if input files did not change since the last execution of *enu* default: False
`stereo`	boolean, specifying whether spatial isomers should also be enumerated by *enu* default: False

References

[1] Marina P Oliveira, Maurice Andrey, Salomé R Rieder, Leyla Kern, David F Hahn, Sereina Riniker, Bruno AC Horta, and Philippe H Hünenberger. Systematic optimization of a fragment-based force field against experimental pure-liquid properties considering large compound families: Application to saturated haloalkanes. *J. Chem. Theory*, 16:7525–7555, 2020.

[2] Marina P Oliveira and Philippe H Hünenberger. Systematic optimization of a fragment-based force field against experimental pure-liquid properties considering large compound families: Application to oxygen and nitrogen compounds. *Phys. Chem. Chem. Phys.*, 23:17774–17793, 2021.

[3] Salomé R. Rieder, Marina P. Oliveira, Sereina Riniker, and Philippe H. Hünenberger. Development of an open-source software for isomer enumeration. *J. Cheminform.*, 2022. submitted.