

Assemble!
- User Manual -



Matteo Degiacomi, University of Oxford; Valentina Erastova,
Durham University

Contents

1	Getting started	2
2	Introduction	2
3	Files	4
3.1	Input files	4
3.1.1	Monomer coordinates file	4
3.1.2	Monomer topology	6
3.1.3	Force field file	8
3.1.4	Database file	10
3.1.5	Parameter file	10
3.2	Output files	14
4	Graphical User Interface	14

1 Getting started

Assemble! is a tool aimed at generating atomistic polymeric mixtures ready for simulation in Gromacs [1]. It is developed by Dr Matteo Degiacomi, University of Oxford, and Dr Valentina Erastova, Durham University.

Assemble! source code (available under GPL2 license), Windows and Mac OSX binaries can be downloaded from at the following address <http://degiacon.github.io/assemble/>. Executing *Assemble!* from source requires Python 2.6.X or $\geq 2.7.X$, as well as numpy and wx packages to be installed. Files generated by *Assemble!* can be directly submitted to Gromacs 4.X.

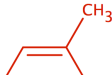
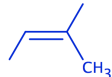
Assemble! logo has been designed by Jos Tasche, Durham University.

2 Introduction

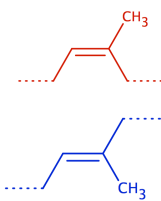
Assemble! represents a polymer as a chain of molecular subunits, having their coordinates and topology associated to a one-letter code within a user-defined database (see section 3.1.4). In order to create a polymer chain, *Assemble!* first determines the position of virtual hooking points (hereon "hooks") next to the terminal atoms of every molecular subunit. Terminal atoms and hooks are used to properly arrange two consecutive subunits. Hooks location can be determined in two ways. If topology information and a force field are available, *Assemble!* creates a polymer chain by extracting connections' bond, angle and dihedral information from them. If no topology is available, user defined hooks coordinates can be provided within the subunits PDB file. When adding a new molecule to a polymer chain, *Assemble!* will make check for atomic clashes. If any is detected, hooks positions will be perturbed until the clash is solved. Polymers sequence can be user defined, or randomly generated. Every generated polymer will be aligned along its inertia tensor in order to minimise its bounding box (first principal axis along x, second along y and third along z). The generated molecules can be finally packed into a simulation box, ready to be submitted to Gromacs. The current version of *Assemble!* has been tested with Gromacs 4.6. Figure 1 shows the whole *Assemble!* pipeline.

Assemble! can be controlled by a Graphical User interface (see section 4) or via terminal, by means of a parameter file. In the latter case, the following

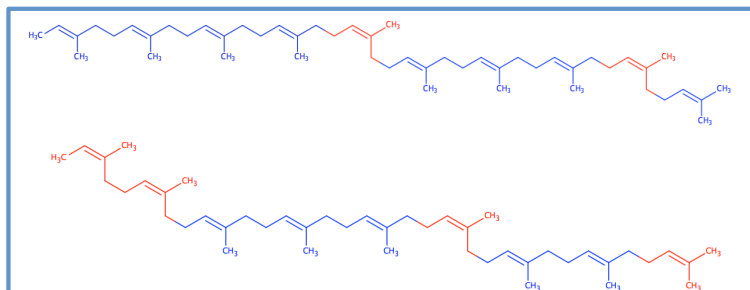
INPUT:

Database	Chain
C 	1) length: 10 mu C -20%, T -80%
T 	2) length: 9 mu C -50%, T -50%

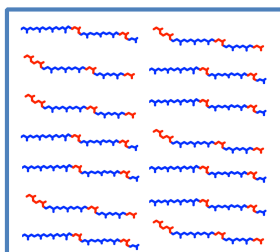
PROCESSING:

Hooks & Terminals	Sequence
	1) TTTTCTTTCT 2) CCTTTCTTC

GENERATION:



PACKING:



OUTPUT:

```
System.gro
System.top
Polymer1.gro
Polymer1.itp
Polymer2.gro
Polymer2.itp
```

Figure 1: Assemble! *pipeline*

call will launch assemble:

```
.\assemble.py input_file.dat
```

Where `input_file.dat` is a parameter file (see Section 3.1.5)

3 Files

3.1 Input files

3.1.1 Monomer coordinates file

A polymer consists of a chain of monomeric subunits. Atomic coordinates for each monomer unit type have to be provided by the user in a .pdb format. Monomers should contain at least two atoms. The folder database, included in the package, contains united atom models of cis-polyisoprene, trans-polyisoprene, cis-polybutodiene, trans-polybutodiene, vinyl-polybutodiene and hydrogenated styrene monomers. Figure 2 shows an example of a cis-PI-monomer.pdb file for a united atom representation of cis-isoprene monomer unit:

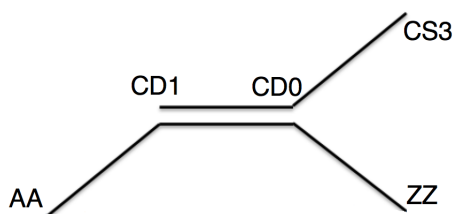


Figure 2: *Monomer of cis-isoprene.*

ATOM	1	CD0	CIS	X	1	19.944	16.788	58.971	1.00	0.00	C
ATOM	2	CD1	CIS	X	1	18.606	16.816	58.918	1.00	0.00	C
ATOM	3	AA	CIS	X	1	17.821	15.516	58.820	1.00	0.00	C
ATOM	4	CS3	CIS	X	1	20.754	18.073	59.069	1.00	0.00	C
ATOM	5	ZZ	CIS	X	1	20.705	15.471	58.934	1.00	0.00	C
END											

In the case of using program in only `pdb` mode, without need of an input topology, the provided monomer unit should contain dummy atoms, that

will be used as a reference for connection. The folder `database-hooks`, included in the package, contains united atom models of cis-polyisoprene, trans-polyisoprene, cis-polybutodiene, trans-polybutodiene, vinyl-polybutodiene and hydrogenated styrene monomers with dummy atoms at the polymerisation sites. Figure 3 shows an example of cis-polyisoprene monomer with dummy atoms on the ends.

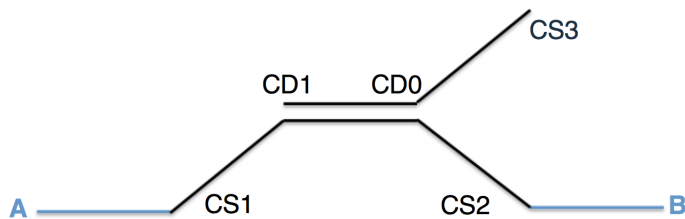


Figure 3: *Monomer of cis-isoprene (black) with dummy atoms (blue) at the end.*

It is important that the .pdb file is written in the correct form. In the example below, the .pdb file of a cis-isoprene monomer unit is shown (see figure 3):

```
LIMIT HEAD 3
LIMIT TAIL 5
LIMIT HEAD_HOOK 6
LIMIT TAIL_HOOK 6
ATOM 1 CD1 CIS X 1 19.944 16.788 58.971 1.00 0.00 C
ATOM 2 CD0 CIS X 1 18.606 16.816 58.918 1.00 0.00 C
ATOM 3 CS1 CIS X 1 17.821 15.516 58.820 1.00 0.00 C
ATOM 4 CS3 CIS X 1 20.754 18.073 59.069 1.00 0.00 C
ATOM 5 CS2 CIS X 1 20.705 15.471 58.934 1.00 0.00 C
ATOM 6 A CIS X 1 22.204 15.186 58.980 1.00 0.00 C
ATOM 7 B CIS X 1 16.327 15.825 58.775 1.00 0.00 C
END
```

Where LIMIT HEAD 3 is the first, CS1, atom of the monomer the previous monomer unit attaches to, LIMIT TAIL 5 is the the last, CS2, atom of the monomer that attaches to the following monomer unit. LIMIT HEAD_HOOK 6 is a dummy atom A the atom that is next to LIMIT HEAD 3 and merges into the LIMIT TAIL 5. Consequently, LIMIT TAIL_HOOK 7 is the dummy atom that is next to LIMIT TAIL 5 and will merge into the LIMIT HEAD 3. As the

atoms are merged a single bond between CS2 and following CS1 is created, as illustrated in the figure below. The terminal atoms are not changed or renamed, it is done during conversion to Gromacs input (.gro and .top) files. These files can be generated with `pdb2gmX` tool. Within the package provide the `database-hooks/TraPPE-pdb2gmX` folder, containing parameters for a small collection of monomers that can be used to assign the forcefield to the polymer.

3.1.2 Monomer topology

The `gromacs` mode assigns the forcefield and topology to the polymer system. Therefore a user would need to provide a monomer specific topology. The topology format is based .rtp file used in Gromacs with program-specific amendments. Please see Gromacs manual, Chapter 5.6 for details. The monomer topology file has to work in conjunction with the force field file, detailed here in section 3.1.3. As example, a set of topology files for aforementioned united atom monomer units are given in the `database` folder.

The following inputs should be provided:

- [mapping]
Input two columns: First column is atom name, according to the .pdb file; Second column is an atom force field type. Please note that atom names must be unique.
- [bonds]
Input three columns: First and second columns are atom names, according to .pdb. Third column gives a name of a force field parameter type available within the force field file.
- [angles]
Input four columns: First, second and third columns are atom names, according to .pdb. Fourth column a name of a force field parameter type available within the force field file.
- [impropers]
Input five columns: First, second, third and fourth columns are atom name, according to .pdb. Fifth column a name of a force field parameter type available within the force field file.
- [dihedrals]
Input five columns: First, second, third and fourth columns are atom

names, according to .pdb. Fifth column a name of a force field parameter type available within the force field file.

- [nterminal] and [cterminal]

These sections provide information about n-terminus (beginning) and c-terminus (ending) part of the residue. In these sections, corrections to apply to the topology of a molecule in order to transform it into a terminal molecule can be listed. These patches will be automatically applied by *Assemble!* on the first and last molecules of every polymer. Modifiers for atomtypes (two columns), bonds (three columns), angles (four columns) and dihedrals (five columns) can be listed here. Contents of these modifying lines matches that of their respective sections listed above.

For the bonded interactions ([bonds], [angles], [impropers] and [dihedrals]) between atoms of two monomer units, adding + before the atom name signifies the atom in the following monomer and - for the atom in the previous one. When joining two building blocks, *Assemble!* will look in both building blocks topologies for bond, angle and dihedral information about the connection. Atoms order is irrelevant (e.g. "ZZ CD0 CS3" and "CS3 CD0 ZZ" describe the same angle).

Below is an example for the monomer topology file for the cis-PI-monomer shown on figure 2.

```
[ mapping ]
; name  FF-type
  CD0    DC0
  CD1    DC1
  AA     CH2
  CS3    CH3
  ZZ     CH2

[ bonds ]
;1atom 2atom FF-parameter
  ZZ    +AA    s-bn
  AA     CD1    s-bn
  ZZ     CD0    s-bn
  CS3    CD0    s-bn
  CD1    CD0    d-bn

[ angles ]
```



```

AA      -ZZ      -CD0      s-an
ZZ      +AA      +CD1      s-an
ZZ      CD0      CS3       d-an
ZZ      CD0      CD1       d-an
CS3     CD0      CD1       d-an
AA      CD1      CD0       d-an

[ impropers ]
AA      CD1      CD0      ZZ      c-id
AA      CD1      CD0      CS3     t-id

[ dihedrals ]
+AA     ZZ      CD0      CD1     d-dh
-ZZ     AA      CD1      CD0     d-dh
CD0     ZZ      +AA     +CD1     s-dh

[ nterminal ]
AA              CH3
[ cterminal ]
ZZ              CH3

```

3.1.3 Force filed file

This is a single file uniting a set of gromacs force field files, namely: `ffbonded.itp`, `ffnonbonded.itp` and `forcefield.itp`. For more detail of these files see Gromacs manual Tables 5.4 and 5.5.

In this file the order is important. First part describes all bonded interactions. The keyword `[bondedtypes]` is followed by four numbers in the next line, indicating the interaction type for bonds, angles, dihedrals and impropers, according to the Table 5.5 in Gromacs manual. The following entries are of a `ffbonded.itp` format and gives the name of the forcefield parameter type, as in 3.1.2, followed by the parameters of the desired force field.

The second part of the file provides parameters for non-bonded interactions, following the format of a `ffnonbonded.itp` file. The keyword is `[atomtypes]`, followed by a row per atom, in the given format: atom type, atom number, mass, charge, type and, if using Lennard-Jones, σ and ϵ .

The last entry is of a `forcefield.itp` format and provides the combination rules for non-bonded interactions. The keyword `[defaults]`, followed by one

line formatted as follows: non-bonded type, combination rule, generate pairs (yes/no), scaling factor for Lennard-Johnes, scaling factor for Coulomb. Detailed information on these values can be found in Gromacs manual Chapter 5.3.2 .

Below is an example of the `forcefield.ff.txt` file for a polyisoprene polymer, that can be created from cis-polyisoprene given above (figure 2), and trans-polyisoprene included in the folder `database`, using TraPPE [2] force-field:

```
;BONDED INTERACTIONS
[ bondedtypes ]
; bonds  angles  dihedrals  impropers
      1      1          3          2
;
; Bond
d-bn      0.133  1.500e+05
s-bn      0.154  1.500e+05
;
; Angle
s-an      114      519.65388125
d-an      119.7    585.504421082
;
; Improper
c-id      0.0      206.19866008
t-id      180.0    222.82758428
;
; Dihedral
s-dh      8.39881329788865  ... .. -0.0179106949224495
d-dh      2.27265025391004  ... .. 0.0190345532675874
;
;
;Non-Bonded
[ atomtypes ]
;name  at.num  mass      charge  ptype      sigma      epsilon
DC0     6      12.011    0.000    A          0.385      0.166289242
DC1     6      13.0186   0.000    A          0.373      0.3907797187
DC2     6      14.0266   0.000    A          0.3675     0.70672928
CH1     6      13.0186   0.000    A          0.465      0.08314462
CH2     6      14.0266   0.000    A          0.395      0.3824652566
CH3     6      15.0345   0.000    A          0.375      0.8148172858
```

```

;
;combination rules
[ defaults ]
; nbfunc      comb-rule      gen-pairs      fudgeLJ fudgeQQ
1             2             yes             0         0.5

```

3.1.4 Database file

A database file stores information about molecular subunits. Here follows an example:

```

C ./database/cis-PBD-monomer.pdb ./database/cis-PBD-monomer.txt
T ./database/trans-PBD-monomer.pdb ./database/trans-PBD-monomer.txt
V ./database/vinyl-PBD-monomer.pdb ./database/vinyl-PBD-monomer.txt
S ./database/Styrene.pdb ./database/Styrene.txt

```

Every line in the file contains, in order: a single character acting as label for the molecule, a path to a PDB file and a path to a topology file. Paths can be absolute or relative (with respect of *Assemble!* installation directory). Lines starting with the # symbol will be ignored.

3.1.5 Parameter file

Assemble! behavior is controlled via a text file. The file is structured as a series of keywords (one per line) having one or more corresponding values. Keywords are case sensitive, and their order is irrelevant. Lines starting with the # symbol will be ignored. The following keywords are available:

- database < *molecules file name* >
Acceptable values: string
Description: Path to database file containing information about molecular building blocks (see Section 3.1.4)
- mode < *assemble working mode* >
Acceptable values: gromacs or pdb
Default value: gromacs
Description: In case the gromacs mode is selected, *Assemble!* will create all files needed to execute a molecular dynamics simulation in Gromacs (see Section 3.2). To run this mode, the database file must provide information about molecules topology as well, and a force field file must also be provided.

- ForceField < *Force Field file name* >
Acceptable values: string
Description: Path to the force field file.
- default_bond < *default hook distance* >
Acceptable values: float
Default value: 1.5
Description: If not specified in the force field, this value determines the distance of hooking points from termini (in Å). This is useful only if the bond potential in the force field has no analytical minimum (i.e. when using a tabulated potential, so fftype is not between 1 and 7). Note: this keyword is available in text mode only.
- default_angle < *default hook angle* >
Acceptable values: float
Default value: 114
Description: If not specified in the force field, this value determines the angle of hooking points from termini and a previous atom (in degrees). This is useful only if the angle potential in the force field has no analytical minimum (i.e. fftype is not 1 or 2). Note: this keyword is available in text mode only.
- default_dihedral < *default hook dihedral* >
Acceptable values: float
Default value: 120
Description: if not specified in the force field, this value determines the dihedral angle placing the hooking point with respect of termini and two previous atoms (in degrees). This value is typically useful when the dihedral potential used in the force field has no analytical minimum (i.e. fftype is not equal to 1 or 2). Note: this keyword is available in text mode only.
- clash_threshold < *threshold distance for clash detection* >
Acceptable values: float
Default value: 0.9
Description: When adding a new molecule to an existing polymer chain, *Assemble!* makes sure that no heavy clash is generated. If one is detected, the dihedral angles of current and next hooking points are

perturbed until a clash free arrangement is found (grid search using 5 degrees step). This parameter determines the minimal distance (in Å) between atoms of newly added molecule and the ones already part of the polymer chain, below which a heavy clash is detected. In the unlikely event that no clash free arrangement can be found, execution will continue anyway and a polymer featuring one or more heavy clashes will be produced. In this case, three courses of actions can be taken: (1) modify the `default_dihedral` keyword, and try running *Assemble!* again, (2) manually modify the generated polymer or (3) run gromacs anyway, and hope the clash can be solved with a sufficiently long minimization round. Note: this keyword is available in text mode only.

- molecule *< molecule names to generate >*
Acceptable values: list of string
Description: list of polymers names to generate, space separated. These names will be used as labels in other commands referring to the specific polymers (see `chain`, `length`, `composition`, `concentration`).

- chain *< sequence of building blocks >*
Acceptable values: two strings
Description: two values are expected: a polymer name (defined in keyword `molecule`) and a character sequence defining the polymer composition. The sequence is composed of letters referring to molecular subunits, as defined in the loaded database file (see keyword `database`).

- composition *< concentration of building blocks >*
Acceptable values: list
Description: If a polymer sequence is not explicitly defined via the keyword `chain`, this can be randomly generated. To do so, the keyword `composition` must be used in conjunction with the keyword `length`. `composition` defines the concentration of individual subunits within the polymer. The first string is the polymer's name, as defined in the keyword `molecule`, the following lines define concentrations. In the line `composition t2 T 75 C 25`, the polymer `t2` is composed by 75% of T molecules, and 25% of C subunits. Note that the sum of all concentration must equal to 100.

- length *< length of polymer >*

Acceptable values: integer

Description: If a polymer sequence is not explicitly defined via the keyword `chain`, this can be randomly generated. To do so, the keyword `length` must be used in conjunction with the keyword `composition`. `length` defines the length of the random chain to be generated.

- `box_grid_shape < grid dimensions >`

Acceptable values: three positive integers

Description: Multiple copies of defined polymers will be arranged on a three dimensional grid. `box_grid_shape` defines the number of voxels along the x, y and z axes of the grid. Every polymer is aligned along its inertia tensor to minimize its bounding box size (x being the major axis, y the second and z the third). In the final system, every voxel's dimension will be equal to the largest dimension within the polymer's bounding boxes.

- `concentration < polymer concentration >`

Acceptable values: list

Description: Every polymer will exist in multiple copies within the final system. This keyword allows to control how many copies of each polymers will be placed in the system. Two values are expected: the polymer's name as defined with the `molecule` keyword, and a floating point number between 0 and 100, defining the percentage of this polymer within the system. This keyword can be called multiple times, to define the concentration of different polymers. The sum of all defined concentration must be equal to 100%.

- `system_name < output files basename >`

Acceptable values: string

Default value: `system`

Description: `basename` of Gromacs-ready output files (see section 3.2). This will also be the name of the directory *Assemble!* will create to store all the produced files.

- `output_folder < output folder path >`

Acceptable values: string

Default value: current working directory

Description: path for folder where the *Assemble!* results will be

dumped (note that *Assemble!* will create a new directory within the output folder).

3.2 Output files

When `gromacs` mode is chosen, *Assemble!* produces all the necessary structure and topology files to run a molecular dynamics simulation with Gromacs. For every defined polymer, coordinate files in gromacs (.gro) and protein database (.pdb) format, as well as include topology file (.itp) are generated. We encourage the user to inspect the resulting structures.

If generation of a system box is chosen, a coordinate file (.gro) for the final system and associated topology file (.top) are also generated, along with a Gromacs index file (.ndx) allowing the selection of polymers by kind. Aiming for modularity, system topology (.top) only defines atomtypes, and imports individual polymer include topology files (.itp) as source of topology information. During its execution, *Assemble!* communicates useful information on the terminal. This information is also stored into a logfile having as base-name the system name. All these files are written into a selected folder.

4 Graphical User Interface

Assemble! comes with a Graphical User Interface, aimed at easily creating and submitting input files to the *Assemble!* engine. The whole GUI is shown on Figure 4.

The GUI is conceived to be used from top to bottom. Initially, database and force field information are provided, then polymers are defined, and finally the desired system is described and generated. At any moment, the state of the interface can be saved with the *File > Save* top menu. This will save a parameter file. A parameter file can be directly used to execute *Assemble!* via console, or loaded again in the GUI with the *File > Load* menu to continue working with it in a second moment. Below is a description of a typical usage.

- **load a database.** Select a database file in your system using the *Select* button, and load it with the *Load* button. A database can be also edited, modified or even created from scratch using the buttons *Edit*, *Add*, *Remove*. The created database can be saved for further usage with the button *Save*.

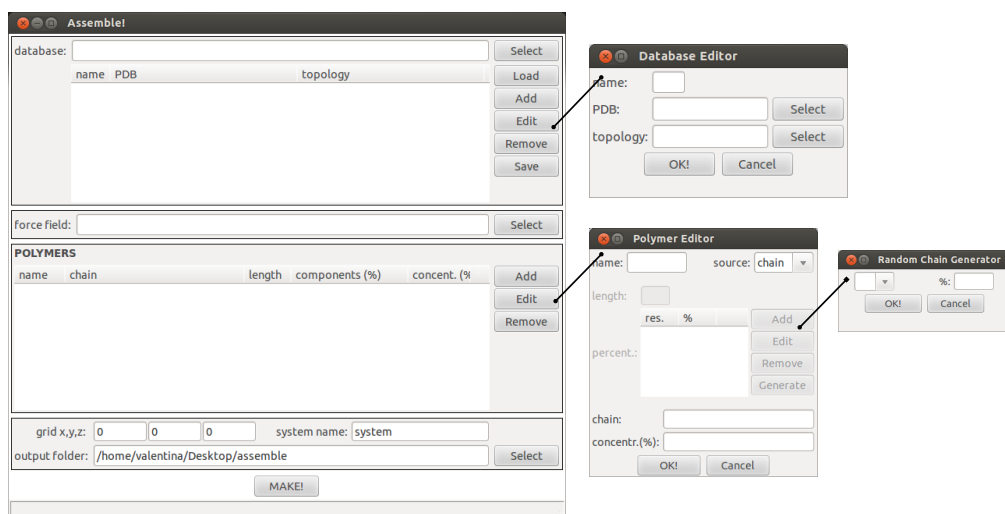


Figure 4: Assemble! GUI

- **load individual monomer components.** When a pre-made database is not available, it is possible to add individual monomers to the database with *Add*, *Remove* buttons.
- **load a force field.** Select a force field file using the button *Select*. Notice that the topology files defining the molecular subunits in the database should match this force field.
- **create polymers.** To create a new polymer, press the button *Add*. A popup menu will appear. First, define a polymer name in the *name* textbox. Then define the polymer's sequence. This can be done in two ways, explicitly or randomly. For an explicit definition, select *chain* in the *source* dropdown menu, and manually provide the desired sequence in the active *chain* textbox below. For a random definition, select *percentage* in the *source* dropdown menu. First, provide a chain length in the active *length* textbox. Subunits percentages can then be added, edited and removed with the *Edit*, *Add* and *Remove* buttons. Once all percentages have been defined, a random chain can be created by pressing the button *Generate*. Notice that the GUI will verify that all provided parameters are consistent. In particular, notice that sequence letters should exist in the database. The last step is to define this polymer's concentration in the final system, by providing a percentage value in the *concentr. (%)* textbox. Once this is done, pressing *OK* will save the polymer definition and display it in the list box on the main interface. Created polymers can be edited and removed with the

Edit and *Remove* buttons.

- **define box.** In the bottom part of the GUI the system grid size can be defined in the three *grid x,y,z* fields. These should be positive integers, defining how many voxels along the x, y and z axis will compose the system.
- **generate files.** Define the basename of the final system files in the *system name* textbox, and the folder where all the generated files will be located. Launch data generation by pressing the button *MAKE!*. After a parameter's consistency check, all the files will be produced in a folder having the same name name of the system, within the selected target directory.

References

- [1] Sander Pronk, Szilárd Páll, Roland Schulz, Per Larsson, Pär Bjelkmar, Rossen Apostolov, Michael R Shirts, Jeremy C Smith, Peter M Kasson, David van der Spoel, et al. Gromacs 4.5: a high-throughput and highly parallel open source molecular simulation toolkit. *Bioinformatics*, page btt055, 2013.
- [2] Collin D Wick, Marcus G Martin, and J Ilja Siepmann. Transferable potentials for phase equilibria. 4. united-atom description of linear and branched alkenes and alkylbenzenes. *The Journal of Physical Chemistry B*, 104(33):8008–8016, 2000.