Polymer Structure Predictor

Contents

General Information

Automation in predicting 3D models of polymers is the key to speed up the discovery of promising organic materials. Polymer structure predictor (PSP) effectively fulfills the current requirements as it can build a hierarchy of polymer models ranging from polymer chain to amorphous models using SMILES strings.

Installation

PSP requires the following packages:

- RDKit (v2019.09.1) (`https://www.rdkit.org/`)
- Open Babel (v3.1.1) (`https://openbabel.org/docs/dev/index.html`)

Note that, both RDKit and Open Babel must are be installed manually. For detailed installation procedure, visit toolkits' websites RDKit and Open Babel. Make sure that all dependencies are installed correctly and working properly.

We recommend using Anaconda python, and creating a fresh conda environment for the install (e. g. `conda create -n MY_ENV_NAME`).

Then use the included setup.py procedure, from the cloned directory.

```
python setup.py install
```

Calling the Program

Polymer structure predictor (PSP) builds a hierarchy of polymer models from SMILES strings. The list of modules available are (1) *MoleculeBuilder* (2) *ChainBuilder* (3) *CrystalBuilder*, (4) *AmorphousBuilder*, and (5) *DimerBuilder*.

3.1 Molecule Builder

MoleculeBuilder generates models for monomer, linear oligomers, and looped oligomers. Note that geometries are minimized using the universal force field (UFF), and output files are provided in POSCAR and XYZ formats. The below code snippet is an example of creating 3D geometries of molecules.

```
import pandas as pd
import psp.MoleculeBuilder as mb


df_smiles = pd.read_csv("molecule.csv", low_memory=False)
mol       = mb.Builder(df_smiles, ID_col="ID", SMILES_col="SMILES")
results   = mol.Build()
```

`df_smiles` is a pandas dataframe with columns `ID` and `SMILES` of having molecule names and their respective SMILES strings. All output files will be stored in the `molecule` directory. An example pandas Dataframe for molecules M1 and M2, and their chemical structures (Figure 3.1) obtained by *MoleculeBuilder* are depicted below.

ID	**SMILES**
M1	C(C)(CC(F)F)(F)F
M2	c1c2c(cc(c1)c1oc3c(n1)cc(cc3))nc(o2)

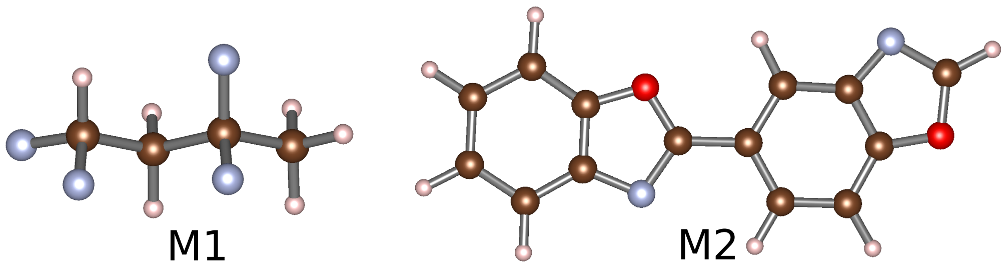

FIGURE 3.1: Chemical structures of M1 and M2. Carbon, hydrogen, and fluorine(M1)/nitrogen(M2) are shown in brown, light brown, and cyan colors, respectively.

3.2 Chain Builder

ChainBuilder builds monomer, oligomers, and polymer chain from a polymer SMILES. In the polymer SMILES, atoms must be assigned as the linking atoms by adding an asterisk [*]. For example (Figure 3.2), SMILES strings of ortho, meta, and para-linked polyphenylene are [*]c1ccccc1[*], [*]c1cccc([*])c1 and [*]c1ccc([*])cc1, respectively.

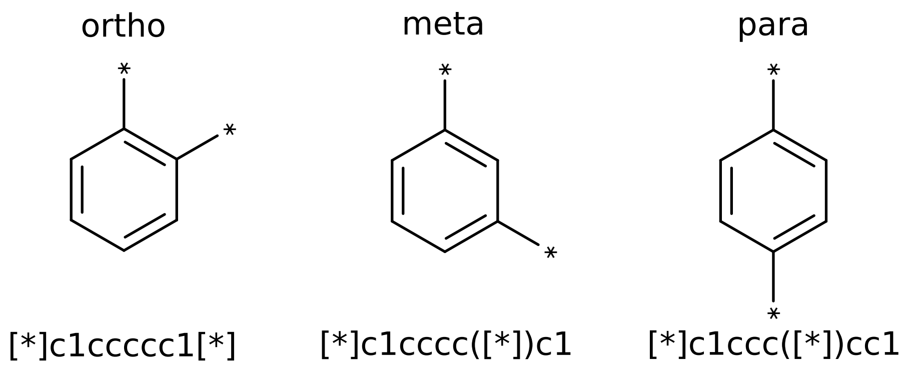

FIGURE 3.2: SMILES strings of polyphenylenes connected in 1,2-fashion (ortho), 1,3-fashion (meta), and 1,4-fashion (para).

The below code snippet is an example of creating polymer chain models from polymer SMILES.

```
import pandas as pd
import psp.ChainBuilder as ChB


df_smiles     = pd.read_csv("chain.csv")
chain_builder = ChB.Builder(df_smiles, ID_col="ID", SMILE_col="SMILES")
results       = chain_builder.BuildChain()
```

Here, `df_smiles` is a pandas Dataframe of input polymer SMILES strings, whose two columns are specified by `ID_col` and `SMILES_col`, respectively. An example Dataframe looks as follows:

ID	**SMILES**
P1	C(C[*])(CC([*])(F)F)(F)F
P2	c1c2c(cc(c1)c1oc3c(n1)cc(cc3)[*])nc(o2)[*]

The above code will create polymer chains (Figure 3.3). Print ‘results’ to see if a SMILES is successfully converted to a polymer chain or not.

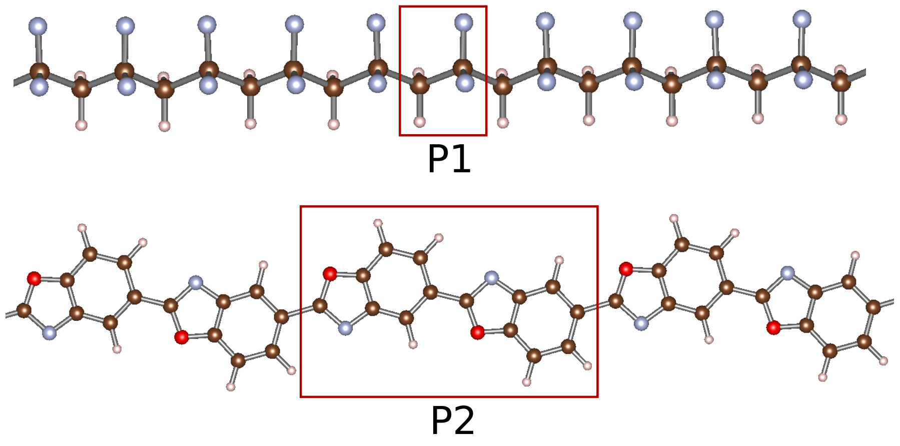

FIGURE 3.3: Polymer chains of P1 and P2, where red box indicates the repeat unit. Carbon, hydrogen, and fluorine(P1)/nitrogen(P2) are shown in brown, light brown, and cyan colors, respectively.

```
print(results)
```

```
ID    Result     Conformers
P1    SUCCESS        1
P2    SUCCESS        1
```

Output geometries for oligomers are provided in both `xyz` and VASP `POSCAR` formats, while infinite polymer chains are exported only in VASP `POSCAR` format. All files are stored in *chains* directory. We can regulate *ChainBuilder* based on our needs. For details, please see to Subsection 4.2.

3.3 Crystal Builder

CrystalBuilder builds crystal models from an oligomer or a polymer chain (VASP `POSCAR` file). To obtain crystal structures, keeping one chain at a fixed position, another chain is translated, rotated around its own axis, and rotated around the other chain.

The below code snippet is an example of building crystal models from oligomer or polymer chains.

```
import psp.CrystalBuilder as CrB


chain_list = ["chains/P1/P1_1.vasp", "chains/P2/P2_1.vasp"]
crystal_builder = CrB.Builder(VASPInp_list=chain_list)
results         = crystal_builder.BuildCrystal()
```

Here, `chain_list` is the list of oligomer or polymer chains. Make sure that *CrystalBuilder* can find them in specified locations. Let's run the above code to create crystal models for P1 and P2 polymer chains and print *results*.

```
print(results)
```

```
 ID     Count   radius
P1_1     125     5.0
P2_1     125     14.0
```

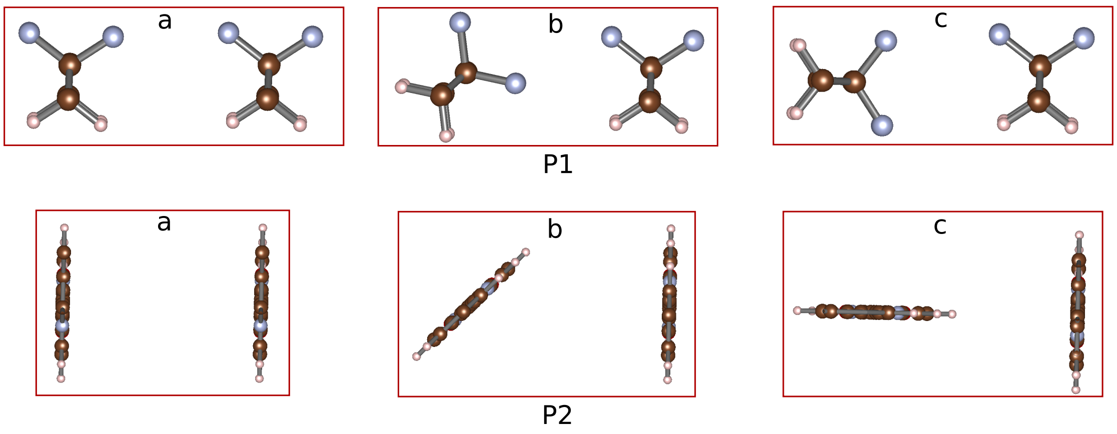

FIGURE 3.4: Examples of crystal models (top views) for P1 and P2, in which one chain is rotated around its axis as well as with respect to the other chain, keeping the latter one fixed. The periodic boundary conditions specified along x and y directions are represented as a box. Carbon, hydrogen, and fluorine(P1)/nitrogen(P2) are shown in brown, light brown, and cyan colors, respectively.

We created 125 crystals for each polymer chain, and a few of them are shown in Figure 3.4. All ouput crystal models are in VASP `POSCAR` format and are stored in *crystals* directory. A probable inter-chain distance (denoted as **radius**) in a unit cell is automatically computed, which is used to rotate one chain with respect to the other fixed one. To further regulate crystal structures, go to Subsection 4.3.

3.4 Amorphous Builder

AmorphousBuilder generates amorphous models from SMILES strings. First, the SMILES strings are used to generate linear/loped models using *MoleculeBuilder*. Second, all molecules are packed inside the box with a given intermolecular distance by minimizing an objective function as implemented in PACKMOL. Third, an output file is generated that can be used directly as an input for MD simulation (LAMMPS).

The below code snippet is an example for building an amorphous model.

```
import pandas as pd
import psp.AmorphousBuilder as ab

input_df = pd.read_csv("input_amor.csv")
amor = ab.Builder(input_df, ID_col="ID", SMILES_col="smiles",
       Length="Len", NumConf="NumConf", Loop="Loop")
amor.Build()
```

Here, `input_df` is a pandas Dataframe of input related to molecules considered in the amorphous model, where columns `ID_col`, `SMILES_col`, `Length` and `NumConf` stand for unique name, SMILES string, length and number of conformers for each molecule, respectively. `Loop` defines whether an oligomer is linear or circular in shape. An example Dataframe looks as follows:

ID	**SMILES**	Len	Num	NumConf	Loop
CC3	[*]CC[*]	3	10	2	False
CC6	[*]CC[*]	6	4	2	False
CC8	[*]CC[*]	8	2	1	True
PVC5	C(C([*])Cl)[*]	5	10	1	False
PVC10	C(C([*])Cl)[*]	10	18	2	True

The code above would generate an amorphous model with a cubic simulation box and a density of 0.65 g/cm^3 (Figure 3.5).

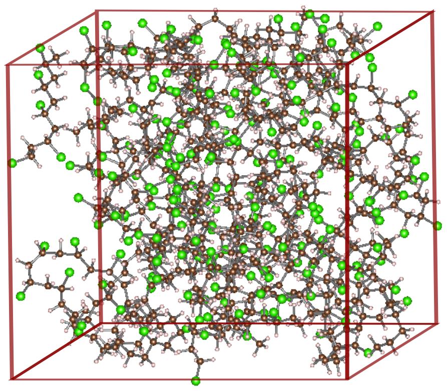

FIGURE 3.5: Amorphous model with a cubic simulation box and a density of 0.65 g/cm^3.

3.5 Dimer Builder

DimerBuilder generates dimer models for any two molecules. The below code snippet is an example of creating dimer models for the Nafion chain and water.

```
import pandas as pd
import psp.DimerBuilder as db


input_df = pd.read_csv("input_DM.csv")
dim = db.Builder(input_df, ID_col="ID", SMILES_col="smiles")
results = dim.Build()
```

Here, `input_df` is a pandas Dataframe for SMILES strings of molecules considered in a dimer model, where columns `ID_col` and `SMILES_col` stand for unique name and SMILES string, respectively. An example Dataframe looks as follows:

ID	**SMILES**
Nafion	[*]C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(F)C(F)(OC(F)(F)C(F)(C(F)(F)(F))OC(F)(F)C(F)(F)S(=O)(=O)O)[*]
Water	O([H])[H]

Let's run the above code to create dimer models for Nafion and water molecules (Figure 3.6).

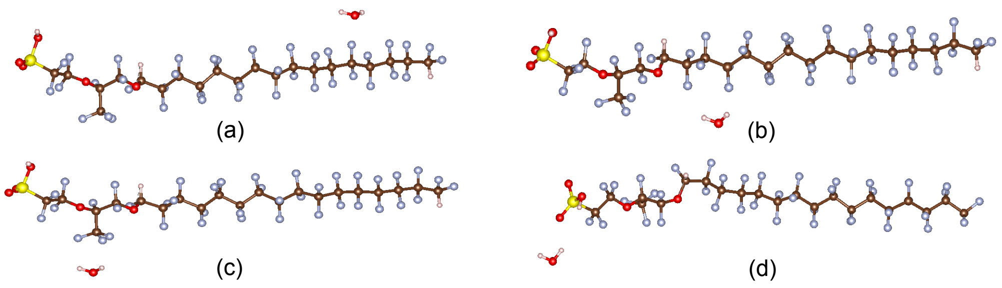

FIGURE 3.6: Four dimer models for Nafion and water molecules.

Detailed Documentation

3D models can be further tuned to meet specific requirements. A complete list of available functionalities for each module is included below, along with examples.

4.1 Molecule Builder

The following code snippet illustrates the full list of features available for tuning monomer and oligomer models.

```
mb.Builder(Dataframe, ID_col='ID', SMILES_col='smiles',
           LeftCap='LeftCap', RightCap='RightCap',
           OutDir='molecules', Inter_Mol_Dis=6,
           Length=[1], NumConf=1, loop=False, NCores=0)
```

1. `ID_col` [default option='ID']: Column name for polymer/molecule names in the Dataframe.

2. `SMILES_col` [default option='smiles']: Column name for the polymer SMILES strings in the Dataframe.

3. `LeftCap` [default option='LeftCap']: Column name for the SMILES string of left cap molecular fragment in the Dataframe.

4. `RightCap` [default option='RightCap']: Column name for the SMILES string of right cap molecular fragment in the Dataframe.

5. `OutDir` [default option='molecules']: The output directory, which contains output models of all molecules in XYZ (*.xyz*), POSCAR (*.vasp*), and PDB (*.pdb*) formats.

6. `Inter_Mol_Dis` [default value=6Å] When periodic boundary conditions are applied (POSCAR format), it determines the minimum distance between two molecules in x, y, and z directions.

7. `Length` [default option=[1]]: Monomer and oligomers can be generated by setting `Length` as desired. For example, if `Length`=[1,2,5], monomer, dimer, and pentamer chain will be created.

8. `NumConf` [default value=1]: PSP can create N number of conformers if `NumConf`=N is assigned, where N is an integer.

9. `loop` [default option=False]: It specifies whether to produce a linear or circular oligomer model. For circular one, set `loop`=True.

4.2 Chain Builder

The PSP provides a set of functionalities to balance the trade-off between computational cost and possibly quality of single chains. *ChainBuilder* module with the complete list of keywords looks as follows:

```
ChB.Builder(Dataframe=df_smiles, ID_col='ID', SMILE_col='smiles',
            Length=[1,2,5,'n'], Method='SA', Steps=25, Substeps=20,
            MonomerAng='medium', DimerAng='medium', NumConf=1,
            IntraChainCorr=1, Tol_ChainCorr=50,
            OutDir='chain-models', NCores=4)
```

1. `Dataframe` : A pandas Dataframe having polymer IDs and SMILES strings.

2. `ID_col` [default option='ID']: Column name for the polymer IDs in the Dataframe.

3. `SMILE_col` [default option='smiles']: Column name for the polymer SMILES strings in the Dataframe.

4. `Length` [default option='n']: Monomer and oligomers can be generated by setting `Length` as desired. For example, if `Length`=[1,2,5,'n'], monomer, dimer, pentamer, and infinite polymer chain will be created.

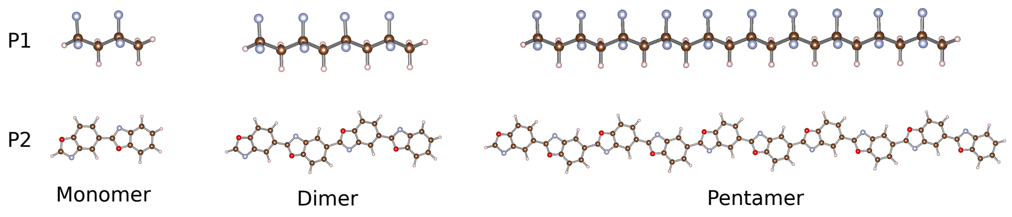

FIGURE 4.1: Monomers, dimers and pentamers of P1 and P2.

5. `Method` [default option='SA']: The default option for *method* is *'SA'*, which means that simulated annealing (SA) method will be used. If *method='Dimer'*, then the PSP will create dimers by rotating one monomer unit while keeping the other one fixed. This can quickly create polymer chains; however, the quality of chains may not be good, and the number of atoms in a unit cell will be high.

6. `Steps` [default value=20]: The computational cost of PSP mainly depends on how many steps and substeps are used in the SA method. `Steps` is the outer loop of the SA, and we recommend to set a large value (`Steps` = 100 or more) if there are many rotatable single bonds in a molecule. To avoid unnecessary calculations, if the SA doesn't find a better geometry after three steps, it is terminated.

7. `Substeps` [default value=10]: `Substeps` is the inner loop of the SA, and `Substeps` = 20 should be sufficient for most of the cases.

8. `MonomerAng` [default option='medium']: This is an array of angles used in the SA process. The user can define his own array of angles or use the predefined parameters. Three predefined parameters are:

    - `low` : [0, 45, -45, 60, -60, 90, 120, -120, 180]

- `medium` : [0, 30, -30, 45, -45, 60, -60, 90, 120, -120, 135, -135, 150, -150, 180]
- `intense` : [-180, -170, -160, ... , 160, 170, 180]

9. `DimerAng` [default option='low']: This is an array of angles used to build a flexible or non-periodic dimer (if PSP fails to get an acceptable repeating unit from a monomer to make a polymer chain). Similar to `MonomerAng`, the user can define his own array of angles or use the predefined parameters.

10. `NumConf` [default value=1]: PSP can create N number of conformers if *num_conf* = N is assigned, where N is an integer. Out of N conformers, the first conformer is the most stable one. If PSP does not find N number of conformers, then it will abolish your request. Note that most of the conformers look very similar because PSP captures them from the last few simulated annealing steps.

11. `IntraChainCorr` [default option=1]: PSP sometimes generates a twisted polymer (often helical) backbone. If `IntraChainCorr`=1, PSP makes the backbone straight by gradually moving two sets of the dummy and linking atoms towards opposite directions on the z-axis.

12. `Tol_ChainCorr` [default value=50]: While straightening a monomer, we apply a cutoff for a sudden energy bump (in KJ/mol) defined by `Tol_ChainCorr`. Once it is reached, PSP doesn't stretch the molecule further.

13. `OutDir` [default option='chains']: This key allows users to define a directory where they want to store output files.

14. `NCores` [default value=0, Number of CPU Cores-1]: By default, PSP uses all the cores (minus one) available in a computer.

4.3 Crystal Builder

The PSP is flexible enough to create desired crystal structures from a single polymer or oligomer chain. The code snippet below shows all keywords available in *CrystalBuilder*.

```
CrB.Builder(VASPInp_list=chain_list, NSamples=5, InputRadius='auto',
            MinAtomicDis=2.0, OutDir='crystals', Polymer=True,
            Optimize=False, NumCandidate=50, NCores=4)
```

1. `VASPInp_list`: A list of names of oligomer or polymer chains with their PATH. Make sure that *CrystalBuilder* can find them in the specified locations.

2. `NSamples` [default value=5]: It stands for the number of samples for each degree of freedom. Total number of crystal models generated by PSP is defined by *NSamples*×*NSamples*×*NSamples*, which is related to one translational and two rotational motions.

3. `InputRadius` [default option='auto']: This key defines the distance between z-axes of two chains, which is used to rotate one chain with respect to another fixed one. If `InputRadius`='auto', *CrystalBuilder* will calculate an approximate distance by considering x and y coordinates of the input geometry and the minimum atomic distance defined by `MinAtomicDis`. To define your own value, change this to a float or integer (in Å).

4. `MinAtomicDis` [default value=2.0] This key defines the minimum distance between any two atoms of two chains in Å. Note that this key works if `InputRadius`='auto'.

5. `Polymer` [default option='True']: In the case of polymers, fixing the position of the first polymer, the second one is moved in three distinct ways, i.e., (1) translated along the z-axis, (2) rotated around its own axis, (3) rotated around the first polymer chain. However, for oligomers, more degrees of freedom must be considered to capture every possible crystal models, which can be activated by setting `Polymer`='False'.

6. `Optimize` [default option='False']: PSP generated crystal models can be further optimized using UFF and conjugate gradient method by setting `Optimize`='True'.

7. `NumCandidate` [default value=50]: This keyword is activated when `Optimize`='True', determining the number of crystal models selected based on the total energy computed by UFF. Note that high-energy models are discarded.

8. `OutDir` [default option='crystals']: This key allows users to define a directory where they want to store output files.

9. `NCores` [default value=0, Number of CPU Cores-1]: By default, PSP uses all the cores (minus one) available in a computer.

4.4 Amorphous Builder

Numerous amorphous models can be generated by varying type of molecules, density, size, and shape of the simulation box. All of these parameters can be tuned by *AmorphousBuilder*.

```
ab.Builder(Dataframe, ID_col='ID', SMILES_col='smiles',
           NumMole='Num', Length='Len', NumConf='NumConf',
           LeftCap = 'LeftCap',RightCap = 'RightCap',
           Loop = 'Loop', OutFile = 'amor_model',
           OutDir='amorphous_models', OutDir_xyz='molecules',
           density=0.65, tol_dis=2.0, box_type='c',
           box_size=[0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
           incr_per=0.4)
```

1. `ID_col` [default option='ID']: Column name for molecule IDs in the Dataframe.

2. `SMILES_col` [default option='smiles']: Column name for SMILES strings in the Dataframe.

3. `NumMole` [default option=‘Num’]: Column name for the number of each molecule in the Dataframe.
4. `Length` [default option=‘Len’]: Column name for the length of each molecule in the Dataframe.
5. `NumConf` [default option=‘NumConf’]: Column name for the number of conformers for each molecule in the Dataframe.
6. `LeftCap` [default option=‘LeftCap’]: Column name for the SMILES string of left cap molecular fragment in the Dataframe.
7. `RightCap` [default option=‘RightCap’]: Column name for the SMILES string of right cap molecular fragment in the Dataframe.
8. `Loop` [default option=‘Loop’]: Column name for selecting linear or circular chains.
9. `OutFile` [default option=‘amor_model’]: The names of the output files exported in POSCAR and LAMMPS data formats.
10. `OutDir` [default option=‘amorphous_models’]: The output directory, which contains all files, including POSCAR and LAMMPS data files.
11. `OutDir_xyz` [default option=‘molecules’]: The directory in which individual molecules are stored in .xyz and .pdb file formats.
12. `density` [default value=0.65 g/cm^3]: The density of a simulation box.
13. `tol_dis` [default value=2.0]: The minimum distance between any two molecules in a simulation box.
14. `box_type` [default option=‘c’]: The shape of a simulation box. Here, ‘c’ and ‘r’ denote cubic and rectangular, respectively.
15. `incr_per` [default value=0.4]: If the shape of a box is rectangular, `incr_per` determines the length of z axis (l_z). l_z = Volume$^{1/3}$ + `incr_per` $\times$ Volume$^{1/3}$.

16. `box_size` [No default value]: It is a list of six numbers that offers an alternative method of explicitly defining the box size. [xmin, xmax, ymin, ymax, zmin, zmax]

4.5 Dimer Builder

Dimer models can be tuned to meet user requirements using the keywords available in *DimerBuilder*.

```
db.Builder(Dataframe, ID_col=‘ID’, SMILES_col=‘smiles’,
              Length=[1], NumConf=1, Ndimer=10, ABdis=2.0,
              Loop=False, OutFile=‘AB’, OutDir=‘dimer_models’,
              OutDir_xyz=‘molecules’)
```

1. `ID_col` [default option=‘ID’]: Column name for molecule IDs in the Dataframe.
2. `SMILES_col` [default option=‘smiles’]: Column name for SMILES strings in the Dataframe.
3. `Length` [default option=[1]]: Monomer and oligomers can be generated by setting `Length` as desired.
4. `OutDir` [default option=‘dimer_models’]: The directory in which dimer models molecules are stored in the XYZ (*.xyz*) file format.
5. `Ndimer` [default value=10]: It determines the number of dimers to be generated.
6. `ABdis` [default value=2.0]: It defines the minimum distance between two molecules in each dimer model.
7. `OutFile` [default option=‘AB’]: It specifies the prefix for output file names stored in `OutDir`.

Some Tips and Tricks

- If your output shows `REJECT`, *ChainBuilder* can't handle the input polymer SMILES.
- If your output shows `FAILURE`, try again. Note that *ChainBuilder* uses the SA method, which randomly picks rotatable bonds and angles while searching for a suitable conformer. If your molecule is big, you may need to increase the number of Steps and Substeps in the SA.
- Keep *Substeps=20*, most of the time it works. Set steps as high as possible, for example, *Step=100.*
- If you provide more freedom in SA by considering more angles for rotation, increase the number of steps.
- Sometimes VESTA doesn't show a bond between two atoms even if they are at an acceptable distance (see Figure 5.1). If VESTA shows a broken chain, calculate the distance between two atoms. If required, add another unit cell and verify the model again.
- In crystal models, if `InputRadius`=‘auto’, the inter-chain distance is sufficiently large and all crystal structures will be accepted. However, if you need more compact models, you have to define the inter-chain distance (`InputRadius`). However, in few crystal models, some parts of two chains may overlap. These crystals are considered unacceptable and will be rejected.

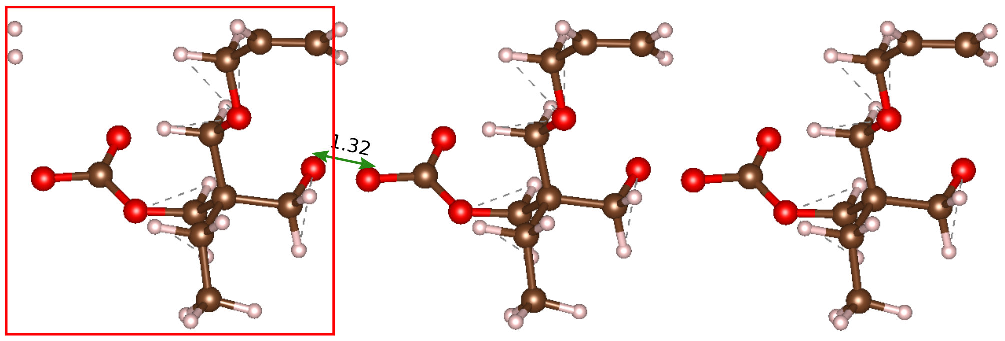

FIGURE 5.1: Polymer chain model for [*]OCC(CC)(COCC=C)COC(=O)O[*]