

---

# *Polymer Structure Predictor*

---

Version 0.1.0

Ramprasad Group  
School of Materials Science and Engineering  
Georgia Institute of Technology

# Contents

<b>Contents</b>	<b>i</b>
<b>1 General Information</b>	<b>1</b>
<b>2 Installation</b>	<b>2</b>
<b>3 Calling the Program</b>	<b>4</b>
3.1 Molecule Builder . . . . .	4
3.2 Chain Builder . . . . .	5
3.3 Crystal Builder . . . . .	7
3.4 Amorphous Builder . . . . .	9
3.5 Colab notebook . . . . .	10
<b>4 Detailed Documentation</b>	<b>12</b>
4.1 Molecule Builder . . . . .	12
4.2 Chain Builder . . . . .	14
4.3 Crystal Builder . . . . .	16
4.4 Amorphous Builder . . . . .	18
<b>5 Some Tips and Tricks</b>	<b>21</b>

## General Information

Three-dimensional atomic-level models of polymers are necessary prerequisites for physics-based simulation studies. Polymer structure predictor (PSP) is capable of generating a hierarchy of polymer models, ranging from oligomers to infinite chains to crystals to amorphous models, using a simplified molecular-input line-entry system (SMILES) string of the polymer repeat unit as the primary input. The output structures and accompanying force field (GAFF2/OPLS-AA) parameter files are provided for downstream DFT and MD simulations.

## 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>)
- PACKMOL v20.2.2 (<http://leandro.iqm.unicamp.br/m3g/packmol/home.shtml>)
- PySIMM v0.2.3 (<https://pysimm.org>)
- LAMMPS (<https://docs.lammps.org/Manual.html>)
- Ambertools v3.1.1 (<https://ambermd.org/AmberTools.php>)
- LigParGen dependencies (<http://zarbi.chem.yale.edu/ligpargen/>)

It should be noted that all dependencies must be installed separately and tested to ensure that they all function. We recommend using Miniconda python, and creating a fresh conda environment for PSP (e.g. `conda create -n MY_ENV_NAME`).

RDKit and OpenBabel are available as conda packages and can be installed using the instructions provided in the following links (1) <https://anaconda.org/rdkit/rdkit> and (2) <https://anaconda.org/conda-forge/openbabel>.

The detailed instructions for the installation of PACKMOL package can be found at the following URL: <http://leandro.iqm.unicamp.br/m3g/packmol/home.shtml>.

Make sure to include the path for PACKMOL executable as an environment variable "PACKMOL\_EXEC" in ~/.bashrc file.

LAMMPS can be installed separately or along with PySIMM. Make sure to add the PySIMM package to your PYTHONPATH and add PySIMM and LAMMPS command-line tools to your PATH as mentioned in the PySIMM documentation.

Ambertools is available as a conda package and can be installed using the instructions provided in the following links: <https://ambermd.org/AmberTools.php>. Make sure to include the path for the Antechamber executable as an environment variable "ANTECHAMBER\_EXEC" in ~/.bashrc file.

Following that, source your ~/.bashrc file. PSP will look for PATHs for PACKMOL, PySIMM, LAMMPS, and Antechamber while performing its tasks.

LigParGen and its dependencies: LigParGen requires the BOSS executable. Obtain a copy of it and set \$BOSSdir variable in bash. For more information, see <http://zarbi.chem.yale.edu/ligpargen> and <http://zarbi.chem.yale.edu/software.html>. To make LigParGen compatible with PSP, we updated it to include the following features: (1) the ability to store the output files in a user-defined directory; and (2) compatibility with the recent versions of Open Babel (v3.1.1), NetworkX (v2.5), and pandas (v1.2.4). Take note that we have not yet installed NetworkX; ensure that this is done. The updated LigParGen source code is redistributed as part of the PSP package.

Once all dependencies are installed, clone the PSP repository and install it using the *setup.py* included in the package.

```
git clone https://github.com/Ramprasad-Group/PSP
python setup.py install
```

**NOTE:** A colab notebook that demonstrates the step-by-step installation procedure and installs PSP and its dependencies has been provided.

## 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*, and (4) *AmorphousBuilder*.

### 3.1 Molecule Builder

*MoleculeBuilder* generates models for monomer, linear and looped oligomers. First, it joins the fragment SMILES strings in a systematic manner to create the SMILES for the desired molecule/oligomer. Second, it generates three-dimensional geometry, performs conformer search, and optimizes the geometry using a force field. Output files are provided in POSCAR and XYZ formats. The below code snippet is an example of creating 3D geometries of molecules.

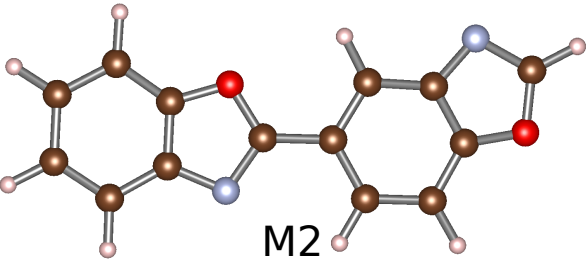
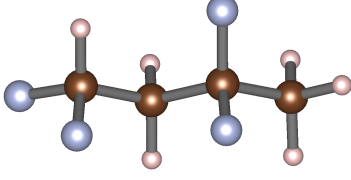
```
import pandas as pd
import psp.MoleculeBuilder as mb

df_smiles = pd.read_csv("molecule.csv")

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 `molecules` 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	<chem>C(C)(CC(F)F)(F)F</chem>
M2	<chem>c1c2c(cc(c1)c1oc3c(n1)cc(cc3))nc(o2)</chem>



**M1****M2**

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 models for infinite polymer chain and oligomers (periodic) using a repeating unit SMILES. The linking atoms in the repeating unit SMILES must be assigned using asterisks [\*]. For example (Figure 3.2), SMILES strings of ortho, meta, and para-linked polyphenylene are [\*]c1ccccc1[\*], [\*]c1cccc([\*])c1 and [\*]c1ccc([\*])cc1, respectively. This module looks for a polymer building block that is thermodynamically preferable while being suitable for imposing a periodic boundary conditions.

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

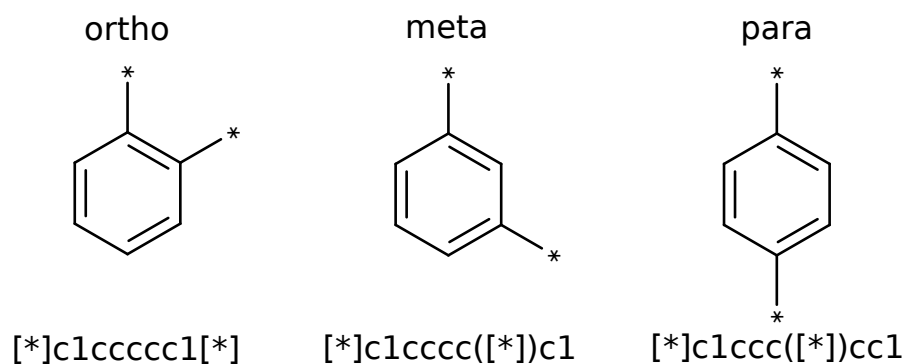


FIGURE 3.2: SMILES strings of polyphenylenes connected in 1,2-fashion (ortho), 1,3-fashion (meta), and 1,4-fashion (para).

```
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	<chem>C(C[*])(CC([*])(F)F)(F)F</chem>
P2	<chem>c1c2c(cc(c1)c1oc3c(n1)cc(cc3)[*])nc(o2)[*]</chem>

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.

```
print(results)
```



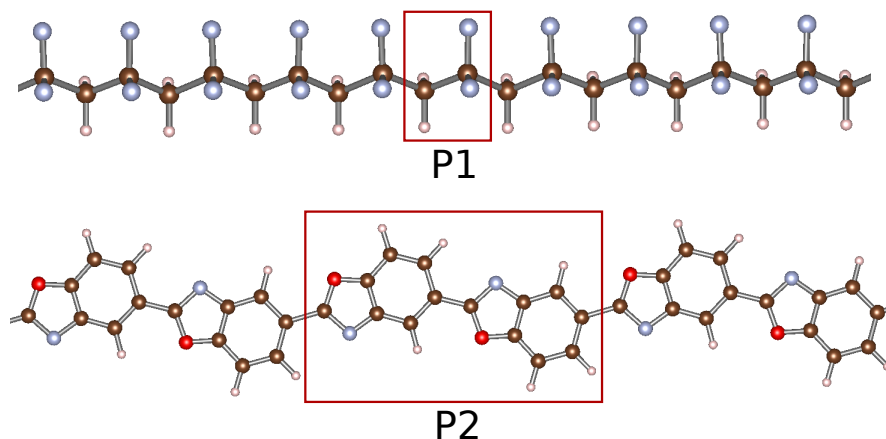


FIGURE 3.3: Polymer chains of P1 and P2, where red box indicates the repeat unit. Carbon, hydrogen, and fluorine(P1)/nitrogen(P2) are shown in brown, light brown, and cyan colors, respectively.

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 Subsection 4.2.

### 3.3 Crystal Builder

*CrystalBuilder* builds crystal models from an oligomer or a polymer chain (VASP `POSCAR` file). *CrystalBuilder* builds crystal models from an oligomer or a polymer chain (VASP `POSCAR` file). A polymer chain, aligned along the  $z$  direction, is placed far from the original chain, rotated by a given angle around itself, rotated by another angle around the original chain, translated along the  $z$  direction by a given distance, and rigidly moved toward the original chain so that the distance between any two atoms is roughly 2 Å. By specifying the number of samples for each rotating angle and translating distance, i.e.,

3 degrees of freedom, a number of candidate crystal structures are obtained. In order to explore different crystal structures for small molecules, in addition to these movements, rotations of both chains around their  $x$ ,  $y$ , and  $z$  axes are considered, allowing for 8 degrees of freedom. These candidate structures are either provided as output or optionally optimized with UFF, and low-energy structures are chosen as the output of the package.

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

We created 125 crystals for each polymer chain, and a few of them are shown in Figure 3.4. All output 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.

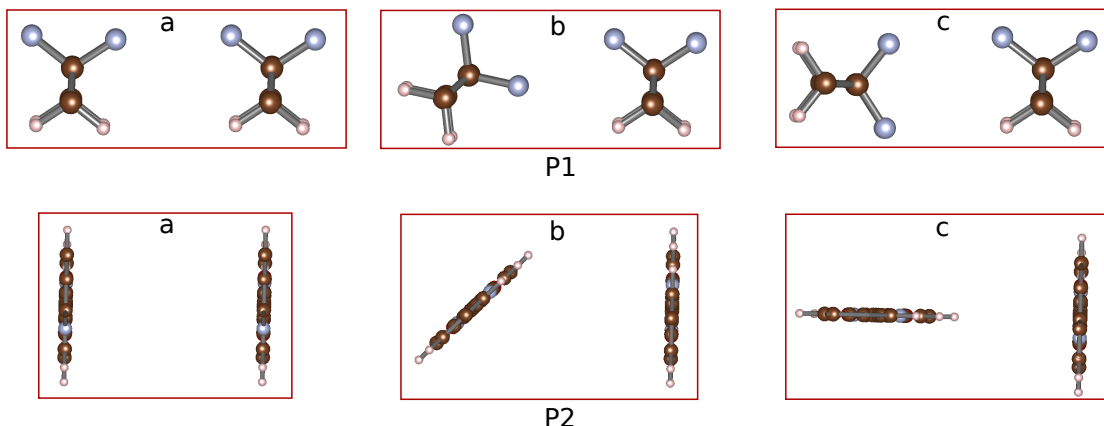


FIGURE 3.4: Examples of crystal models (top views) for P1 and P2, in which one chain is rotated around its axis as well as with respect to the other chain, keeping the latter one fixed. The periodic boundary conditions specified along  $x$  and  $y$  directions are represented as a box. Carbon, hydrogen,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

### 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  $0.65 \text{ g/cm}^3$  (Figure 3.5).

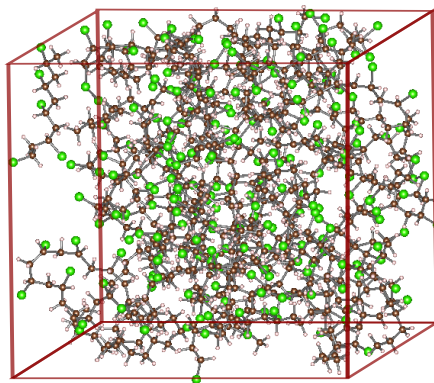


FIGURE 3.5: Amorphous model with a cubic simulation box and a density of  $0.65 \text{ g/cm}^3$ .

### 3.5 Colab notebook

We have included a Colab notebook with the PSP package that provides several examples for each builder. By modifying input SMILES and parameters, users can build

desired polymer models, visualize them, and download them for downstream DFT or MD simulations.

## Detailed Documentation

PSP allows us to set some parameters for keeping a balance between the computational cost and (possibly) the quality of polymer models. A complete list of available functionalities for each module is provided 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. **Dataframe** : A pandas Dataframe that contains polymer IDs and SMILES strings representing repeating and endcap units.
2. **ID\_col** [default option='ID'] : Column name for IDs of polymers/molecules in the Dataframe.

3. **SMILES\_col** [default option='smiles'] : Column name for the polymer SMILES string (repeating unit) in the Dataframe.
4. **LeftCap** [default option='LeftCap'] : Column name for the SMILES string of left cap molecular fragment in the Dataframe.
5. **RightCap** [default option='RightCap'] : Column name for the SMILES string of right cap molecular fragment in the Dataframe.
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 chains will be constructed.
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.
10. **IrrStruc** [default option=False] : If **IrrStruc**=True, a short MD simulation will be performed to generate a more relevant structure. \*This feature is not available here.\*
11. **OPLS** [default option=False] : If **OPLS**=True, an OPLS FF parameter file will be generated for each model.
12. **GAFF2** [default option=False] : If **GAFF2**=True, an GAFF2 FF parameter file will be generated for each model.
13. **GAFF2\_atom\_typing** [default option='pysimm'] : Use *pysimm* or *antechamber* for atom typing.

14. **OutDir** [default option='molecules'] : The output directory, which contains output models of all molecules in XYZ (*.xyz*), POSCAR (*.vasp*), and PDB (*.pdb*) formats, including FF parameter files.
15. **NCores** [default value=0, Number of CPU Cores-1] : By default, PSP uses all the cores (minus one) available in a computer.

## 4.2 Chain Builder

The following code snippet illustrates the full list of features available for tuning infinite polymer chain and oligomer models (periodic).

```
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, Inter_Chain_Dis=12,
             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 (repeating unit) in the Dataframe.
4. **Length** [default option='n'] : It defines the lengths of oligomers. For example, if **Length**=[1,2,5,'n'], monomer, dimer, pentamer, and infinite polymer chain will be created. Note that 'n' stands for infinite polymer chains.
5. **Method** [default option='SA'] : The default option for **method** is 'SA', which means that the simulated annealing (SA) method will be used. If **method**='Dimer', then the *ChainBuilder* will create dimers by rotating one monomer unit while keeping



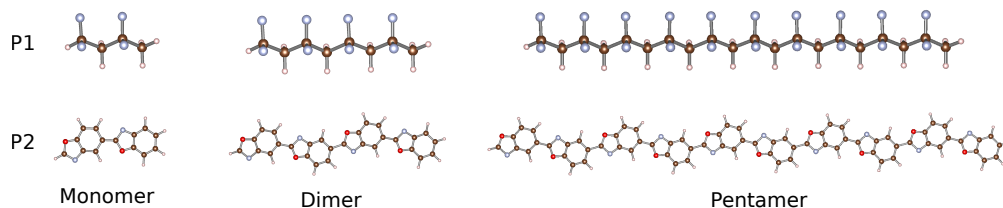


FIGURE 4.1: Monomers, dimers and pentamers of P1 and P2.

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 *ChainBuilder* 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. **IntraChainCorr** [default option=True] : *ChainBuilder* sometimes generates a twisted polymer (often helical) backbone. If **IntraChainCorr**=True, *ChainBuilder* makes the backbone straight by gradually moving two sets of the dummy and linking atoms towards opposite directions on the  $z$ -axis.
9. **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, *ChainBuilder* doesn't stretch the molecule further.
10. **OutDir** [default option='chains'] : This key allows users to define a directory where they want to store output files.
11. **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]
12. **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.
  13. **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.
  14. **Inter\\_Chain\\_Dis** [default option=12Å] : The distance between two periodical images.
  15. **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 determines the number of crystal models to be generated. It can be an integer or a list of lists. If it is an integer, it will be considered as the number of samples for each degree of freedom. For polymer and small molecules, the total number of crystal models will be **NSamples**<sup>3</sup> and **NSamples**<sup>8</sup>, respectively. A specific set of crystal models can be generated by specifying translation distance and rotation angles in a list of lists.
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 an 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. *AmorphousBuilder* allows the following parameters or keywords for fine-tuning of amorphous models:

```
ab.Builder(Dataframe, ID_col='ID', SMILES_col='smiles',
           NumMole='Num', Length='Len', NumConf='NumConf',
           LeftCap = 'LeftCap', RightCap = 'RightCap',
           Loop = 'Loop', NumModel=1, 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. **Dataframe** : A pandas Dataframe that contains polymer IDs, SMILES strings representing repeating and endcap units, length of oligomers, number of oligomers/molecules, chain type (linear or loop), etc.
2. **ID\_col** [default option='ID'] : Column name for molecule/oligomer IDs in the Dataframe.
3. **SMILES\_col** [default option='smiles'] : Column name for polymer SMILES strings (repeating unit) in the Dataframe.
4. **NumMole** [default option='Num'] : Column name for the number of each molecule in the Dataframe.

5. **Length** [default option='Len'] : Column name for the length of each oligomer 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. **NumConf** [default option='NumConf'] : Column name for the number of conformers for each molecule in the Dataframe.
9. **Loop** [default option='Loop'] : Column name for selecting linear or circular chains.
10. **NumModel** [default option=1] : The number of amorphous models that will be generated.
11. **OutFile** [default option='amor\_model'] : The names of the output files exported in POSCAR and LAMMPS data formats.
12. **OutDir** [default option='amorphous\_models'] : The output directory, which contains all files, including POSCAR and LAMMPS data files.
13. **OutDir\_xyz** [default option='molecules'] : The directory in which individual molecules are stored in .xyz and .pdb file formats.
14. **density** [default value=0.65 g/cm<sup>3</sup>] : The density of a simulation box.
15. **tol\_dis** [default value=2.0Å] : The minimum distance between any two molecules in a simulation box.
16. **box\_type** [default option='c'] : The shape of a simulation box. Here, 'c' and 'r' denote cubic and rectangular, respectively.
17. **incr\_per** [default value=0.4] : If the shape of a box is rectangular, **incr\_per** determines the length of z axis ( $l_z$ ).  $l_z = \text{Volume}^{1/3} + \text{incr\_per} \times \text{Volume}^{1/3}$ .

18. **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]

## Some Tips and Tricks

- If your output shows **REJECT**, PSP can't handle the input polymer SMILES.
- *ChainBuilder*: If your output shows **FAILURE**, try again. Note that PSP 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.
- *ChainBuilder*: Keep *Substeps*=20; most of the time it works. Set steps as high as possible, for example, *Step*=100.
- *ChainBuilder*: 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.

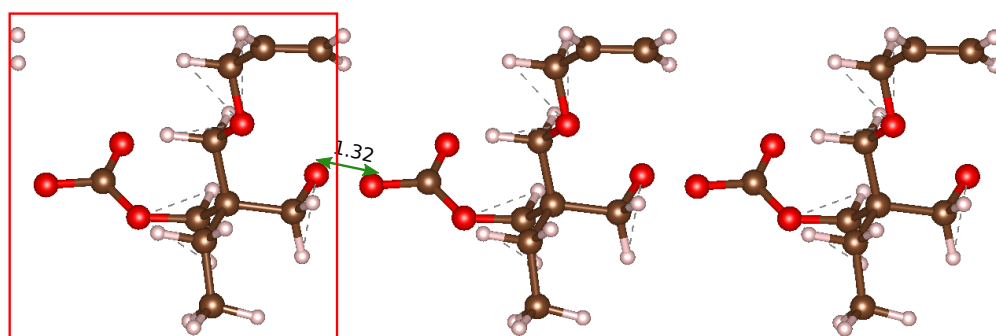


FIGURE 5.1: Polymer chain model for [\*]OCC(CC)(COCC=C)COC(=O)O[\*]