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) toolkit effectively fulfills the current requirements as it can build a hierarchy of polymer models starting from oligomer and polymer chain to crystal models from a SMILES string.

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 along with 3D coordinates of a molecule from its SMILES string. The list of modules available are (1) *MoleculeBuilder* (2) *ChainBuilder* (3) *CrystalBuilder*.

3.1 Molecule Builder

MoleculeBuilder generates XYZ coordinates of a molecule from its SMILES string. Note that geometries are minimized using the universal force field (UFF) and output files are provided in XYZ format. 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',
            OutDir='molecule')
results   = mol.Build3D()
```

`df_smiles` is a pandas dataframe with columns `ID` and `SMILES` of having molecule names and their respective SMILES string. All output files will be stored in the `molecule`

directory. An example pandas dataframe for M1 and M2 molecules and 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)

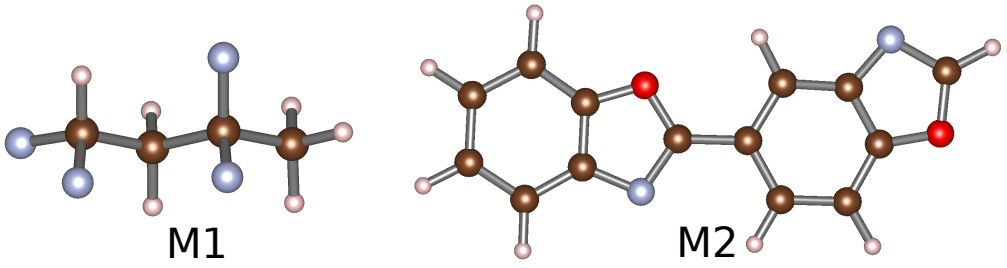

FIGURE 3.1: Chemical structures of M1 and M2. Carbon, hydrogen, and fluorine(M1)/nitrogen(M2) are shown in brown, light brown, and cyan colors, respectively.

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.

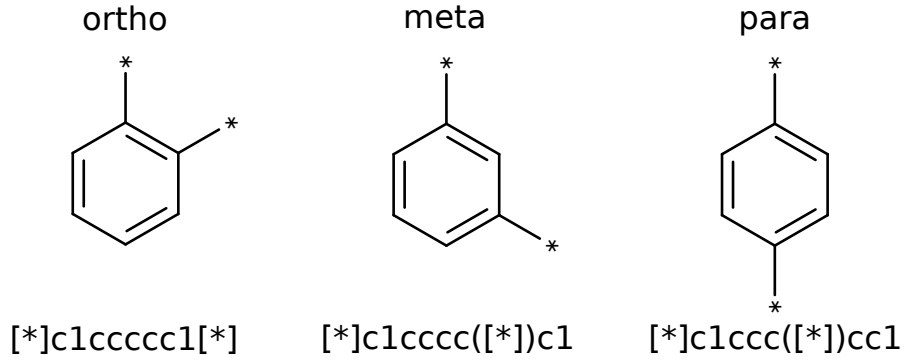

FIGURE 3.2: SMILES strings of polyphenylenes connected in 1,2-fashion (ortho), 1,3-fashion (meta), and 1,4-fashion (para).

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.

```
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 in VASP `POSCAR` format. All files are stored in

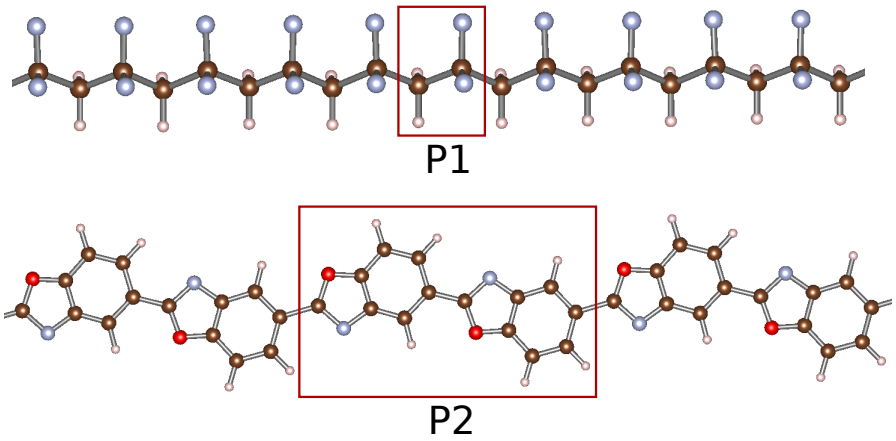

FIGURE 3.3: Polymer chains of P1 and P2, where red box indicates the repeat unit. Carbon, hydrogen, and fluorine(P1)/nitrogen(P2) are shown in brown, light brown, and cyan colors, respectively.

chains directory. We can regulate *ChainBuilder* based on our needs. For details, please see to Subsection 4.0.1.

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

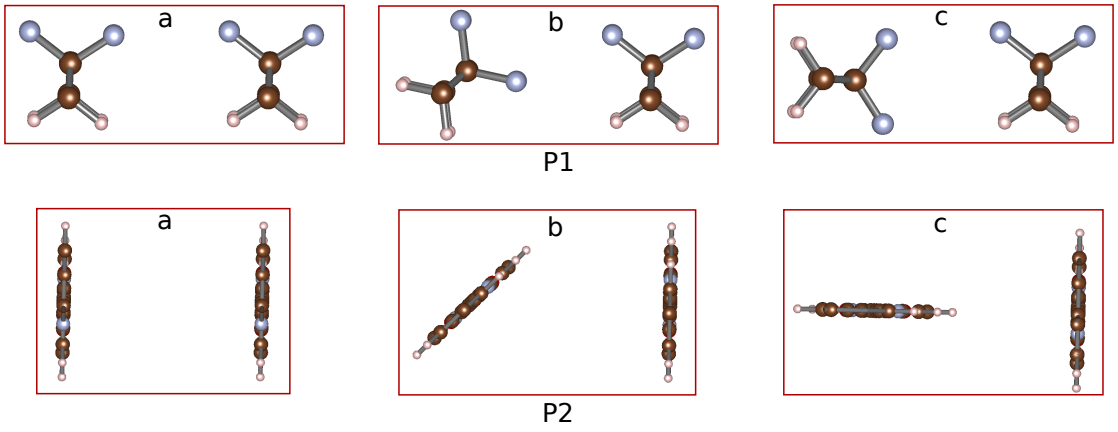

FIGURE 3.4: Examples of crystal models (top views) for P1 and P2, in which one chain is rotated around its axis as well as with respect to the other chain, keeping the latter one fixed. The periodic boundary conditions specified along x and y directions are represented as a box. Carbon, hydrogen, and 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 4.0.2.

Detailed Documentation

PSP allows you to control *ChainBuilder* and *CrystalBuilder* further. Below, a complete list of functionalities is provided, along with examples.

4.0.1 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,
            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.

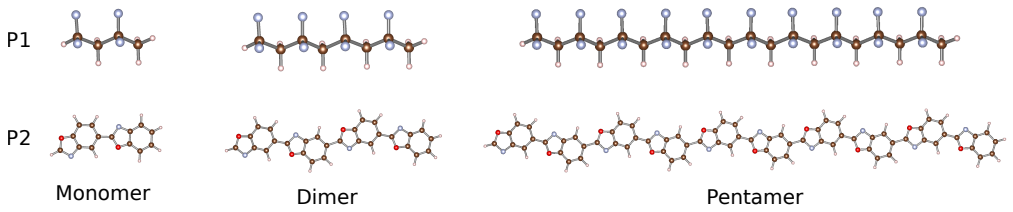

FIGURE 4.1: Monomers, dimers and pentamers of P1 and P2.

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

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

4.0.2 Crystal Builder

The PSP is flexible enough to create desired crystal structures from single chains.

```
CrB.Builder(VASPInp_list = chain_list, NSamples=5,
            InputRadius='auto', MinAtomicDis=2.0,
            OutDir='crystals-models', 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 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 $NSamples \times NSamples \times NSamples$, 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. `OutDir` [default option='crystals']: This key allows users to define a directory where they want to store output files.

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

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.

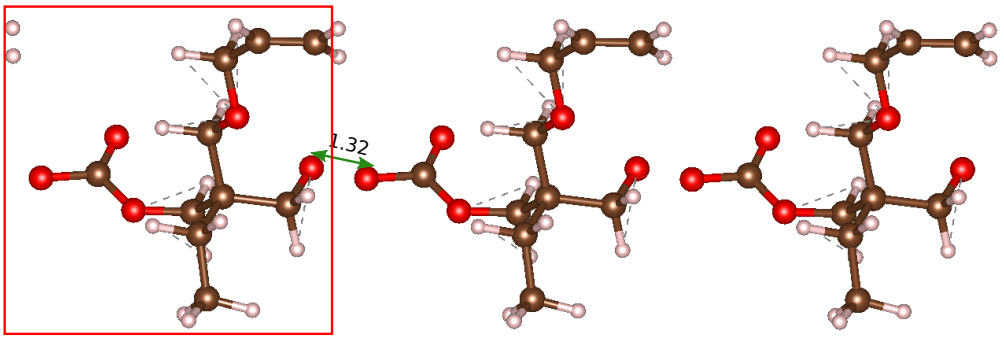

FIGURE 5.1: Polymer chain model for [*]OCC(CC)(COCC=C)COC(=O)O[*]