Polymer Structure Predictor

Contents

General Information

In the case of organic oligomers/polymers, preparing 3D-geometries is the first and crucial step for high-level ab-initio calculations. Building 3D structures by hand is not only a tedious job but sometimes challenging as well. The PSP makes your life a lot easier by automatically generating proper 3D structures from SMILES strings.

Installation

The PSP require the following packages:

- RDKit (https://www.rdkit.org/)
- Open Babel (https://openbabel.org/docs/dev/index.html)

Note that, both RDKit and Open Babel are need to be installed manually. For detailed installation procedure visit toolkits' website: 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 develop
```

Calling the Program

The PSP has two components:

- Polymer Chain Builder
- Crystal Builder

3.1 Polymer Chain Builder

This module builds monomer, oligomers, and polymer from a SMILES string. First, the 3D structure of a molecule is generated and optimized using the universal force field (UFF). Second, the molecule is converted to a repeating monomer (periodic) unit using the simulated annealing method. The repeating structure is further optimized using the steepest descent approach.

3.1.1 Steps to create polymer chains from SMILES

Import this module with the following command

```
import PSP.PolymerBuilder as PB
```

Create a DataFrame of polymer SMILES (df_smiles),

ID	SMILES
P1	[*]CC[*]
P2	[*]C(=O)C1C(c2ccccc2)C(C(=O)N2CCN([*])CC2)C1c1ccccc1

To get polymer chains, we could do something like:

```
chain_builder = PB.Builder(df_smiles)
results = chain_builder.BuildPolymer()
```

This will create polymer chains (Figure 3.1) and put inside directory "*output*" . Print 'results' to see if SMILES are successfully converted to a polymer chain or not.

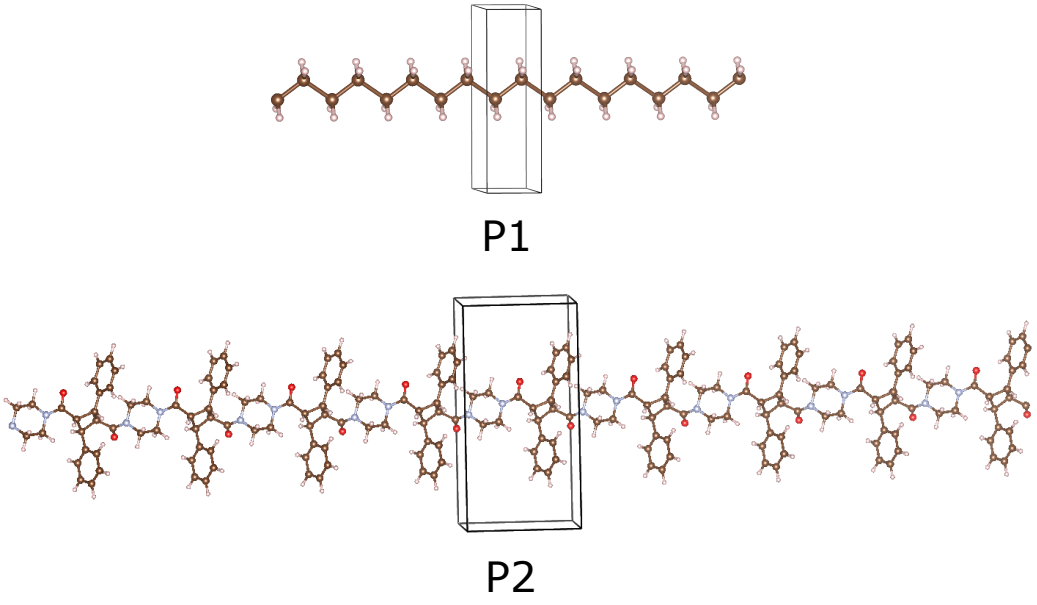

FIGURE 3.1: Polymer chains of P1 and P2.

```
print(results)
```

ID	Result	Conformers
P1	SUCCESS	1
P2	SUCCESS	1

We can regulate Polymer Chain Builder based on our needs. For details, please go to Subsection 4.0.1.

3.2 Crystal Builder

This module builds crystal structures from a monomer, oligomer, or polymer. 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.

3.2.1 Steps to create crystal structures from single chains

Import this module with the following command

```
import PSP.CrystalBuilder as CB
```

First, prepare a list of single-chain polymers. Make sure that Crystal Builder can find polymer chains in specified locations. Here, we are going to create crystal structures for P1 and P2 polymer chains.

```
list = ['output/P1/P1_1.vasp', 'output/P2/P2_1.vasp']
```

```
crystal_struct = CB.Builder(VaspInp_list=list)
results = crystal_struct.BuildCrystal()
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 3.2. A probable inter-chain distance (denoted as **radius**) in a unit cell is automatically

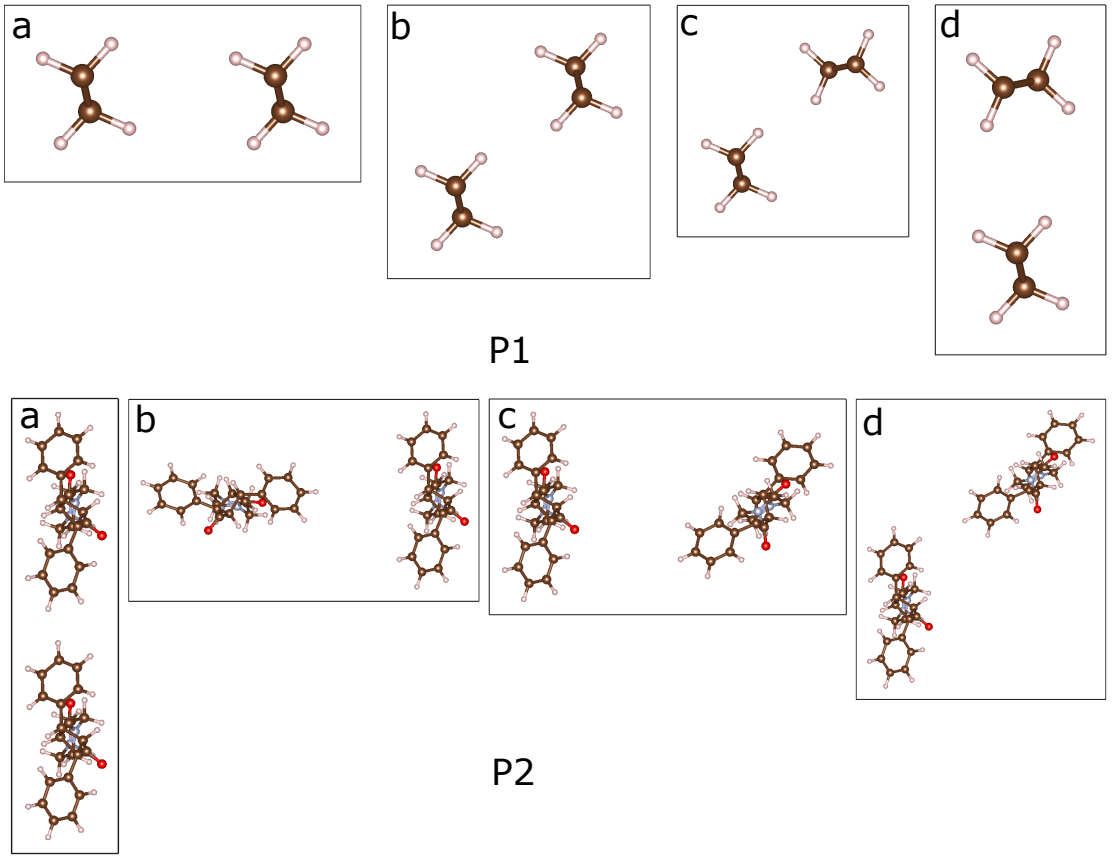

FIGURE 3.2: Crystal structures of P1 and P2.

computed, which is used when we rotate one chain with respect to the other. To further regulate crystal structures, go to Subsection 4.0.2.

Detailed Documentation

The PSP allows you to control Polymer Chain builder and Crystal Builder further. Below, a complete list of functionalities is provided, along with examples.

4.0.1 Polymer Chain Builder

The PSP provides a set of functionalities to balance the trade-off between computational cost and quality of single chains.

The default option for *method* is *'SA'*, which means that the simulated annealing 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.

```
PB.Builder(df_smiles, method='Dimer')
```

The 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. Default value for $num\_conf = 1$.

```
PB.Builder(df_smiles, num_conf=5)
```

To generate monomer or oligomers, define *length* as shown below:

```
PB.Builder(df_smiles, length=[1,2,5,'n'])
```

This will create monomer, dimer, pentamer, and polymer for all samples (Figure 4.1). Monomer and oligomers are saved in both *.xyz* and *.vasp* file formats. *length=['n']* is the default option.

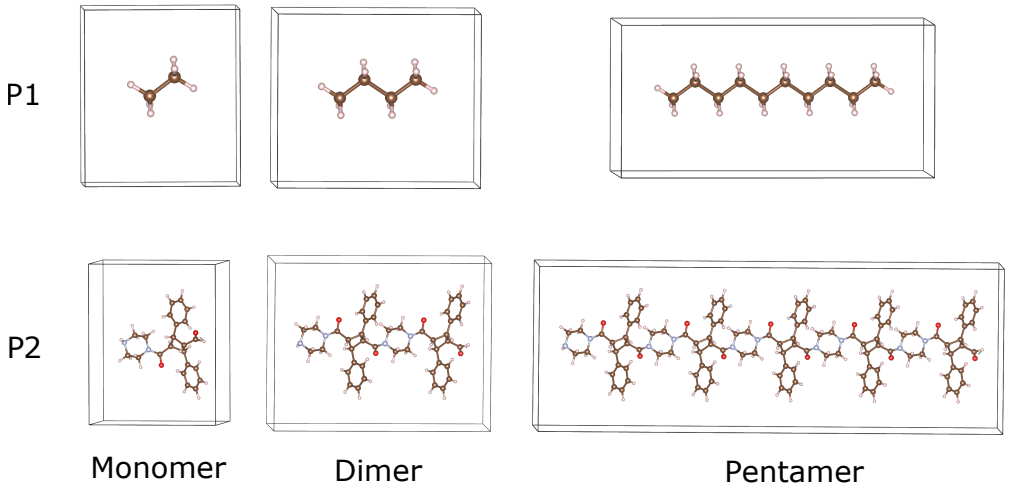

FIGURE 4.1: Monomer, dimer and pentamer of P1 and P2.

The computational cost of the PSP mainly depends on how many steps and substeps are used in the simulated annealing method. You can define them, as shown below.

```
PB.Builder(df_smiles, Steps=100, Substeps=20)
```

This will use 100 steps and 20 sub-steps for each step in the simulated annealing. It is always advisable to keep more steps in comparison to the number of sub-steps. To avoid unnecessary calculations, if the SA doesn't find a better geometry after three steps, it is terminated.

In the simulated annealing method, we rotate all possible single bonds. The allowed angles for rotation can be provided by *input_monomer_angles*, as shown below:

```
PB.Builder(df_smiles, input_monomer_angles='intense')
```

There are three sets of angles available in PSP, i.e., low, medium, and intense. The user can also provide a list of desired angles there (*input_monomer_angles=[5, 10, 15, ...]*).

In the case *method='Dimer'* or when the SA doesn't get a suitable geometry for monomer, we create a dimer (non-periodic). For such a case, we can define allowed angles by *input_dimer_angles*.

```
PB.Builder(df_smiles, input_dimer_angles='medium')
```

By default, the PSP uses all the cores available in your machine. You define cores, as shown below.

```
PB.Builder(df_smiles, n_cores=5)
```

4.0.2 Crystal Builder

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

The *Nsamples* is an important parameter to define in the Crystal Builder. Total number of crystals generated by the PSP is defined by $Nsamples \times Nsamples \times Nsamples$. There are three degrees of freedom considered in the Crystal Builder, i.e., translational and two rotational motions. *Nsamples* stands for the number of samples for each degree of freedom.

```
crystal_struct = CB.Builder(VaspInp_list=list, Nsamples=5)
```

Input_radius defines the distance between two chains, where one is rotated with respect to the other. If *Input_radius='auto'*, the PSP will calculate an approximate

distance by considering XYZ coordinates of two chains. To define your own value, change this to float or integer.

```
crystal_struct = CB.Builder(VaspInp_list=list, Input_radius='auto')
```

Define *OutDir* to store crystal structures:

```
crystal_struct = CB.Builder(VaspInp_list=list, OutDir='Crystals/')
```

By default, the PSP uses all the cores available in your machine. You define cores as shown below.

```
crystal_struct = CB.Builder(VaspInp_list=list, n_cores=5)
```

Some Tips and Tricks

- 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.
- In crystal structures, if *Input_radius='auto'*, the inter-chain distance is sufficiently large and all crystal structures will be accepted. However, if you need more compact structures, you have to define the inter-chain distance. As the center chains are considered for the distance calculation, in few crystal structures, some parts of two chains may overlap. These crystals are considered unacceptable and will be rejected.

Bibliography