TO: Dr. Michael Shirts
FROM: Yu-Tang Lin
DATE: 15th December, 2022
SUBJECT: Summary of quantifying the performance of OpenFF in optimizing crystal structure by RSMD20 and B-factor calculation

This report summarizes my contribution to quantifying the performance of OpenFF in optimizing crystal structure by RSMD20 and B-factor calculation. Sam built the original workflow, and then I merged the code into Tobias' workflow. This memo will detail the procedures and the result for the following topic.

1. Effect of Constraint forcefield
2. Subprocess Workflow Design
3. Energy Minimization Algorithm
4. Temperature profile extraction
5. RMSD20 algorithm development
6. Centroid and B-factor calculation

Effect of Constraint forcefield

At first, the energy minimization workflow was built by constraint OpenFF forcefield, which means that the system was created with constraints on bonds except for O-H hydrogen bonds. In the energy minimization, our own Scipy minimizer will keep stretching the O-H hydrogen bonds until the Scipy minimizer find minimum energy. Thus, the energy-minimized structure will exist unreasonable long O-H hydrogen bonds in Figure 1.

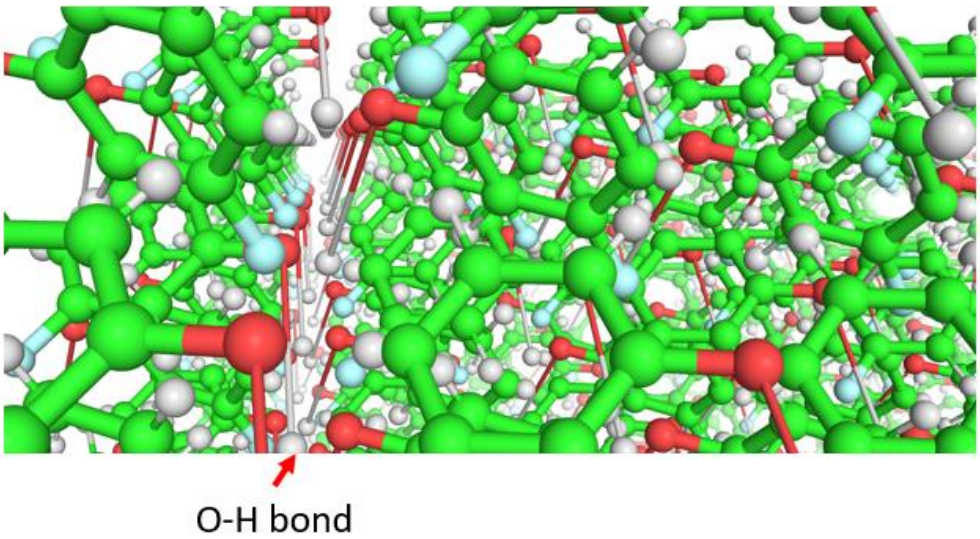

Figure 1: Constraint forcefield example for stretching O-H Bond

Another side effect of using a constraint forcefield is that since our Scipy minimizer is stretching the O-H bonds in a non-reasonable way, the energy gradient will be large, and the energy function may need to be more challenging to converge. The gradient versus iteration figure is shown in Figure 2, where there are a lot of spikes. After the system was switched to an unconstraint forcefield, the Scipy minimizer will stretch O-H Bond and other bonds. There are no unreasonably long O-H bonds in the energy

minimization. The energy versus iteration is shown in figure 3, in which our own Scipy minimizer can easily find energy converge around 100 iterations.

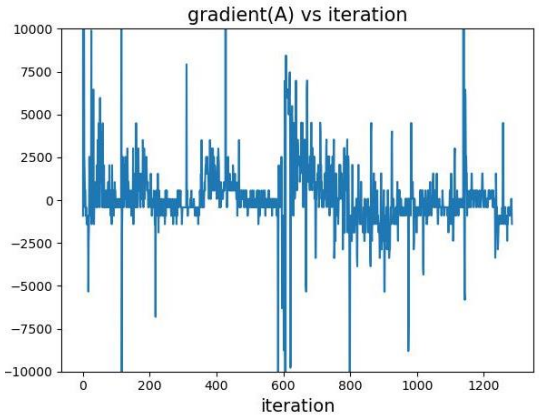

Figure 2: Gradient vs. iteration in constraint forcefield

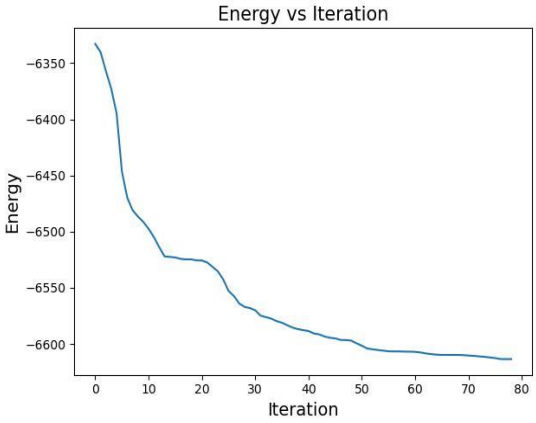

Figure 3: Energy vs. iteration in unconstraint forcefield

Subprocess Workflow Design

Since there are 286 CIF files from the COD database, we will need large computation resources to run all the workflow. In this case, our workflow can be controlled by the python subprocess package and add input for argparse in the current. For building systems (supercell and forcefield) and MD simulation, subprocess.run would be applied into the workflow. For energy minimization subprocess.Popen would be applied, which the program will run until the previous program is finished. Subprocess.Popen is another function in Subprocess package, which can run the target python files parallelly. Previously, it would take us more than 1000 hours for all CIF files to finish the energy minimization by subprocess.run. However, the computation time can be reduced to 8 hours by subprocess.Popen. The program will keep adding another energy minimization to run multiple energy minimizations parallelly. The only thing that needs to control is memory usage, and a multi-core CPU will allocate for the computation.

	subprocess.run	subprocess.Popen
Process time (hours)	1000	8

Table 1: subprocess.run and subprocess.Popen process in energy minimization

Energy Minimization Algorithm and Experiment

For 286 CIF files from the COD database, around 200 structures can be performed on energy minimization, and each structure can be built by 5 different forcefields (OpenFF 1.0.0/OpenFF 1.1.0/ OpenFF 1.2.0/ OpenFF 1.3.0/ OpenFF 2.0.0). Thus, there are around 1000 structures that can be tested to do energy minimization. There are two different ways to run the energy minimization (a) With OpenFF minimization first, then Scipy minimization (b) Only Scipy minimization. For the (a) workflow, it can find a lower energy minimum, which is another crystal structure. Both methods work perfectly for Energy Minimization in this relatively similar energy minima. Energy vs. iteration in 2100147_parsley_1.1.0 for workflow (a) With OpenFF minimization first, then Scipy minimization and energy vs. iteration in 2100147_parsley_1.1.0 for workflow (b) Only Scipy minimization is shown in Figure 4 and Figure 5. The plots are different because, for workflow (a), the initial guess is after OpenMM minimization, OpenMM minimization would not plot in the figure. For workflow (b), the initial guess is with OpenMM minimization, so it would show the whole minimization. Even though the minimization looks good now, it is still good to discuss what different bonds contributed to different energy for different forcefields or why different forcefield exists so different minimized energy with a relatively similar RMSD20.

(a) With OpenFF minimization first, then Scipy minimization						
COD ID	RMSD	RMSD20 Mean	method	Original Energy	Minimized Energy (OpenMM)	Minimized Energy (Cell Minimization)
2100147_parsley_1.0.0	0.01	0.00	L-BFGS-B	63052.95	17198.86	17198.86
2100147_parsley_1.1.0	0.01	0.01	L-BFGS-B	59251.02	13466.21	13389.25
2100147_parsley_1.2.0	0.01	0.01	L-BFGS-B	35177.56	-11070.77	-11131.28
2100147_parsley_1.3.0	0.01	0.01	L-BFGS-B	26201.04	-19621.58	-19773.02
2100147_sage_2.0.0	0.01	0.00	L-BFGS-B	21194.09	-23807.50	-23871.51
(b) Only Scipy minimization						
COD ID	RMSD	RMSD20 Mean	method	Original Energy	Minimized Energy (OpenMM)	Minimized Energy (Cell Minimization)
2100147_parsley_1.0.0	0.00	0.00	L-BFGS-B	63052.95	None	17277.53
2100147_parsley_1.1.0	0.00	0.01	L-BFGS-B	59251.02	None	13893.09
2100147_parsley_1.2.0	0.01	0.00	L-BFGS-B	35177.56	None	-11093.28
2100147_parsley_1.3.0	0.01	0.00	L-BFGS-B	26201.04	None	-19606.95
2100147_sage_2.0.0	0.01	0.00	L-BFGS-B	59251.02	None	13547.42

Table 2: Minimized energy for (a) With OpenFF minimization first, then Scipy minimization (b) Only Scipy minimization

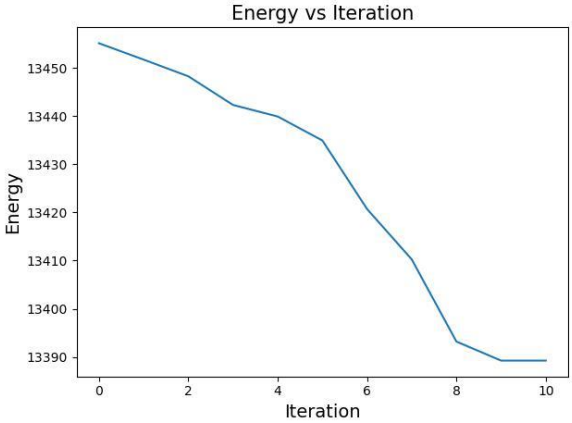

Figure 4: Energy vs. iteration in 2100147_parsley_1.1.0 for workflow (a) With OpenFF minimization first, then Scipy minimization

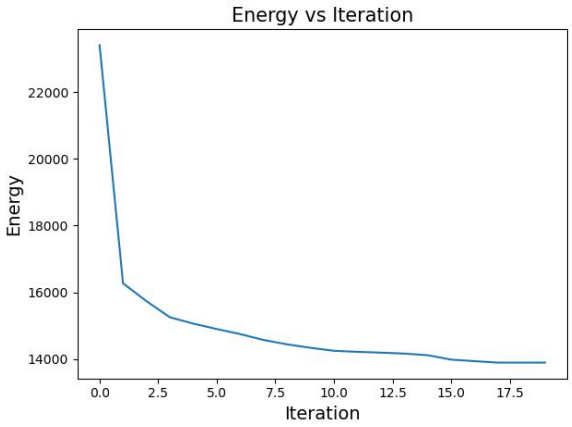

Figure 5: Energy vs. iteration in 2100147_parsley_1.1.0 for workflow (b) Only Scipy minimization

For 200 structures with 5 different forcefields (OpenFF 1.0.0/OpenFF 1.1.0/ OpenFF 1.2.0/ OpenFF 1.3.0/ OpenFF 2.0.0), around 1000 structures can be tested to do energy minimization. OpenFF 1.2.0 have the smallest average RMSD20, which means it fits the experiment structure the most. However, in this small difference, OpenFF 1.0.0/OpenFF 1.1.0/ OpenFF 1.2.0/ OpenFF 1.3.0/ OpenFF 2.0.0 all perform a good simulation for the crystal structure by energy minimization. RMSD20 Distribution for different OpenFF was shown is Figure 6, and the RMSD20 for most of structures' is lower than 0.02.

OpenFF	1.0.0	1.1.0	1.2.0	1.3.0	2.0.0
RMSD20	0.0116	0.0122	0.0108	0.0113	0.0115

Table 3: Average RMSD 20 of OpenFF 1.0.0/OpenFF 1.1.0/ OpenFF 1.2.0/ OpenFF 1.3.0/ OpenFF 2.0.0

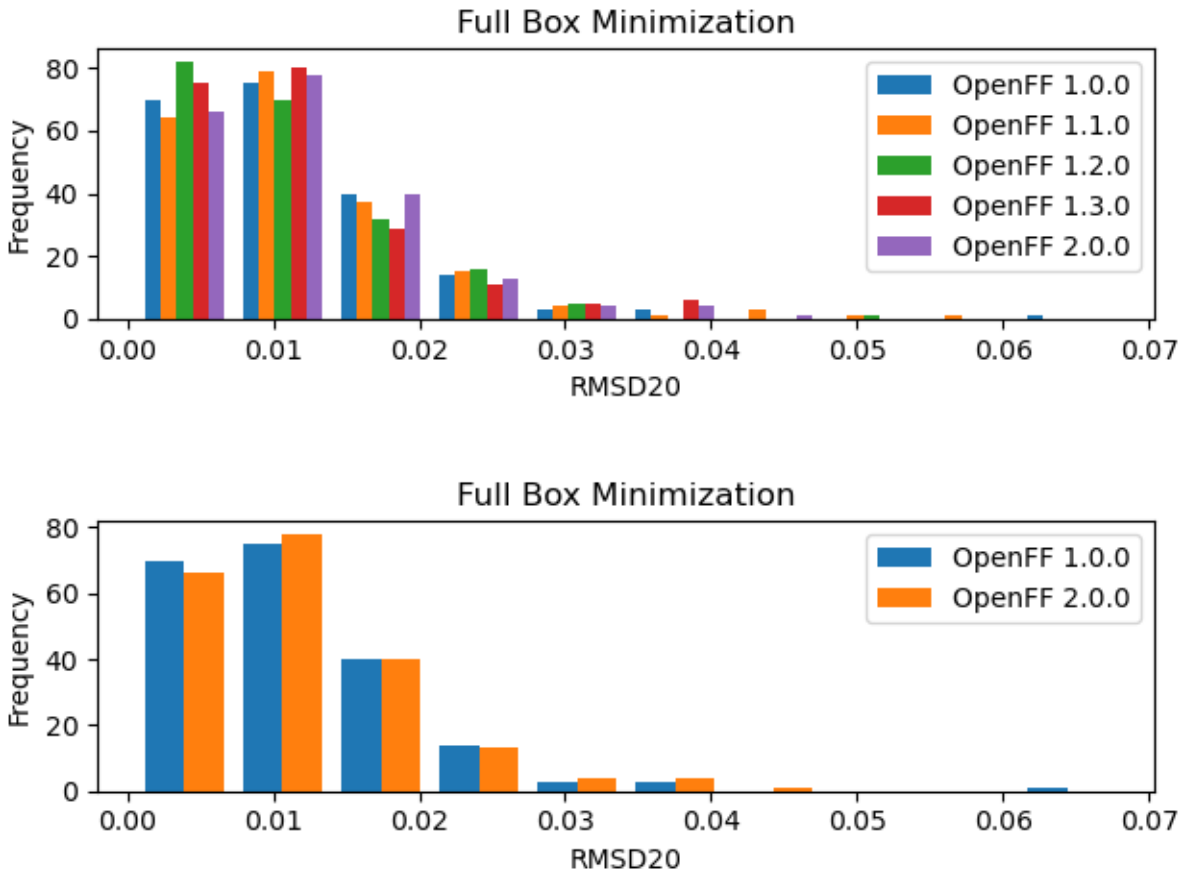

Figure 6: RMSD20 Distribution for different OpenFF

Temperature profile extraction

For 286 CIF files from COD database, only 17 CIF files, including some organic compounds, H2O, and CO2, do not exist in temperature. Most of the CIF file without temperature is really old (19XX). There are two kinds of temperature distribution in COD database in Figure 7. The first one is around 100K, which is for most of gas XRD measurement because gas was needed to crystal under low temperature for XRD measurement. The second temperature distribution is around 300K for the room temperature XRD measurement. The temperature profile can let us know the temperature distribution and as an input temperature for the MD simulation, which is also an important parameter when we run the MD. In this case, since most of the files existed in temperatures 290 - 300 K, I will first use 298.15K as the default temperature (there is no right answer because it depends on the compound or their experiment).

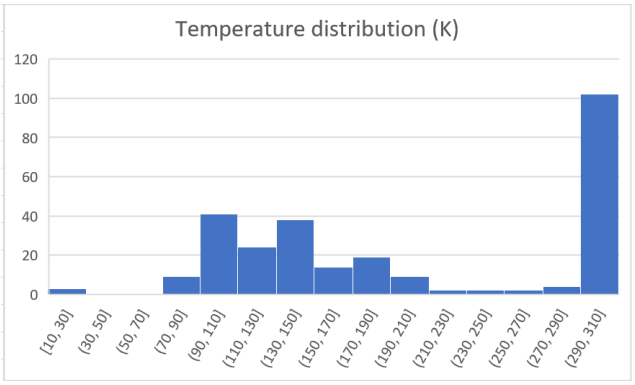

Figure 7: Temperature profile from COD database

RMSD20 algorithm development

Since periodic boundary conditions were applied to OpenFF, which would make a lattice of copies of the system to remove surface effects. Thus, molecules on one side of the box can interact with those on the other side & if a molecule leaves the box, it re-enters on the other side. However, this would make a derivation when we use RMSD to quantify the result because when the molecule leaves the box and re-enters on the other side, the RMSD would increase because of periodic boundary conditions. One easy way to use RMSD20 is to quantify the result, which means that we quantify the result for the clusters of the 20 closest molecules. Thus, RMSD20 can emit the calculation error from periodic boundary conditions. The logistics for RSMD20 is that if atoms and molecules move to more than 0.5*box vector size, it will be determined to jump out of the other side. Thus, the position must subtract a box vector size for the axis. The atom position should be copied and modified by deep copy rather than shallow copy, and the detailed code can be found on the RMSD.py on the GitHub repo.

Centroid and B-factor calculation

The workflow for centroid and B-factor calculation can be finished by mdtraj. The centroid is the most representative molecular structure in the MD simulation and uses the centroid to calculate root mean square fluctuation (RMSF). The following formula can derive B-factor by RMSF.

$$B\_factor = 8/3*pi*pi*(rmsf)**2 \quad (1)$$

The workflow is to load the dcd file from the MD simulation, set up the mdtraj, and then calculate the centroid and RMSF to get a B-factor for each atom, which as shown in Figure 8. Then, B-factor can be compared by the COD database (CIF files). The term for B-factor in the CIF files is “atom_site_B_iso_or_equiv” and “atom_site_U_iso_or_equiv”. To estimate the B-factor result, further effort is needed to observe how atoms move in MD simulation.

```python
for f in os.listdir('./run_0/'):
    if f.endswith("_1.dcd"):
        # load the file
        print(f)
        # setup traj and topology
        traj = md.load('./run_0/' + f, top="../build_system/MM/" + f.split("_")[0]+"_"+f.split("_")[1]+"_"+f.split("_")[2] + ".pdb")
        topology = traj.topology
        # Calculate centroid
        atom_indices = [a.index for a in traj.topology.atoms if a.element.symbol != 'H']
        distances = np.empty((traj.n_frames, traj.n_frames))
        for i in range(traj.n_frames):
            distances[i] = md.rmsd(traj, traj, i, atom_indices=atom_indices)
        beta = 1
        index = np.exp(-beta*distances / distances.std()).sum(axis=1).argmax()
        centroid = traj[index]
        # Calculate RMSD
        rmsf = md.rmsf(traj, centroid, 0)
        # Calculate B_factor
        B_factor = 8 / 3 * math.pi * math.pi * ((rmsf)**2)
        # Zip atom and B_factor to tables
        atom = [str(atom).split("-")[1] for atom in topology.atoms]
        B_factor_table = list(zip(atom, B_factor))
        print('B factor table: ', B_factor_table)
```

Figure 8: Demo code of B-factor calculation

Current Issues

1. Figure out different bonds can contribute to different energy for different forcefields and why different forcefield exists so different minimized energy with a relatively similar RMSD20.
2. Further B-factor data analysis to estimate the performance of OpenFF is by observing how atoms move in MD simulation. This can help us estimate the performance of OpenFF in the crystal structure prediction.

Conclusions

Most of the major previous issues in energy minimization are the transposition of molecules in the crystal cell across the periodic boundary leading to high RMSD values, and the violation of the periodic boundary conditions being in “reduced” form, which causes the energy minimization to fail already been fixed by RMSD20 calculation and trust-constr boundary constraint. This project’s future challenge would be centroid and B-factor calculation. This definitely needs better insight by observing how atoms move in MD simulation, which GROMACS can do or other commercial software can be utilized.