Supporting Information:

Reinvestigation of the Mechanism and Selectivity of 1,8-Cineole Synthase using *Terdockin*

Abhay Potluri, Justin B. Siegel, Dean J. Tantillo*

Department of Chemistry, University of California, Davis, 1 Shields Ave, Davis, CA 95616, USA

djtantillo@ucdavis.edu

Table of Contents

1. DFT Studies and Conformational Search…….2
2. Docking Studies……..…….5
3. Docking Scores from Scheme 1 in *Streptomyces clavuligerus*……6
4. Constraint Information for 1,8-cineole Synthase in *Streptomyces clavuligerus*.7
5. Docking Studies: Deprotonation to 1,8-Cineole in *Streptomyces clavuligerus*.10
6. Docking Studies: α-Terpineol Formation in *Streptomyces clavuligerus*……....12
7. Constraint Information for 1,8-cineole Synthase in *Salvia fruticosa*……...17
8. RMSD violin plots of R and S pathways in *Salvia fruticosa*……20
9. Docking Studies: Water Addition Step in *Salvia fruticosa*…….....20
10. Python Scripts Used for Data Extraction and Figure Making……..22

DFT Studies:

Optimization of intermediates and transition structures was initially performed at B3LYP/6-31G(d,p), followed by mPW1PW91/6-311G(d,p). Verification of transition structures was through imaginary frequency analysis and Intrinsic Reaction Coordinate (IRC) calculations. Every structure except water was performed with +1 charge and 1 multiplicity. A data set collection of computational results is available in the ioChem-BD repository[75,76] and can be accessed via https://doi.org/10.19061/iochem-bd-6-611

The inclusion of the diffuse function only impacted the free energy of the water addition (1 + H_2O → A) step of the mechanism by about 5.0 kcal/mol. With the similarities observed in the free energy surface, the diffuse function did not add to the understanding of the mechanism but added to computational costs. The water addition step is also directly catalyzed by the enzyme which was not modeled in QM. The other intermediates of the mechanism show similar trends in free energy which reduces the added value of the diffuse function in this context. With binding orientations in *Salvia fruticosa* showing water adding syn to the alkene, an additional mechanistic pathway was done with mPW1PW91/6-311G(d,p) (Figure S2).

Level of Theory	1 + H_2O	A	TSAB	B	TSBC	C	TSCD	D
mPW1PW91/6-31G(d,p)	0	0.7	4.2	-10.6	-9.3	-10.6	-7.7	-22.3
mPW1PW91/6-31+G(d,p)	0	6.0	9.9	-4.6	-3.7	-5.6	-3.0	-15.4
mPW1PW91/6-311G(d,p)	0	1.8	5.5	-8.5	-7.1	-8.6	-5.8	-18.5
mPW1PW91/6-311+G(d,p)	0	6.6	10.5	-3.5	-2.3	-4.2	-1.6	-13.3

Table S1. Free Energies (in kcal/mol) of Scheme 1 under different levels of theory.

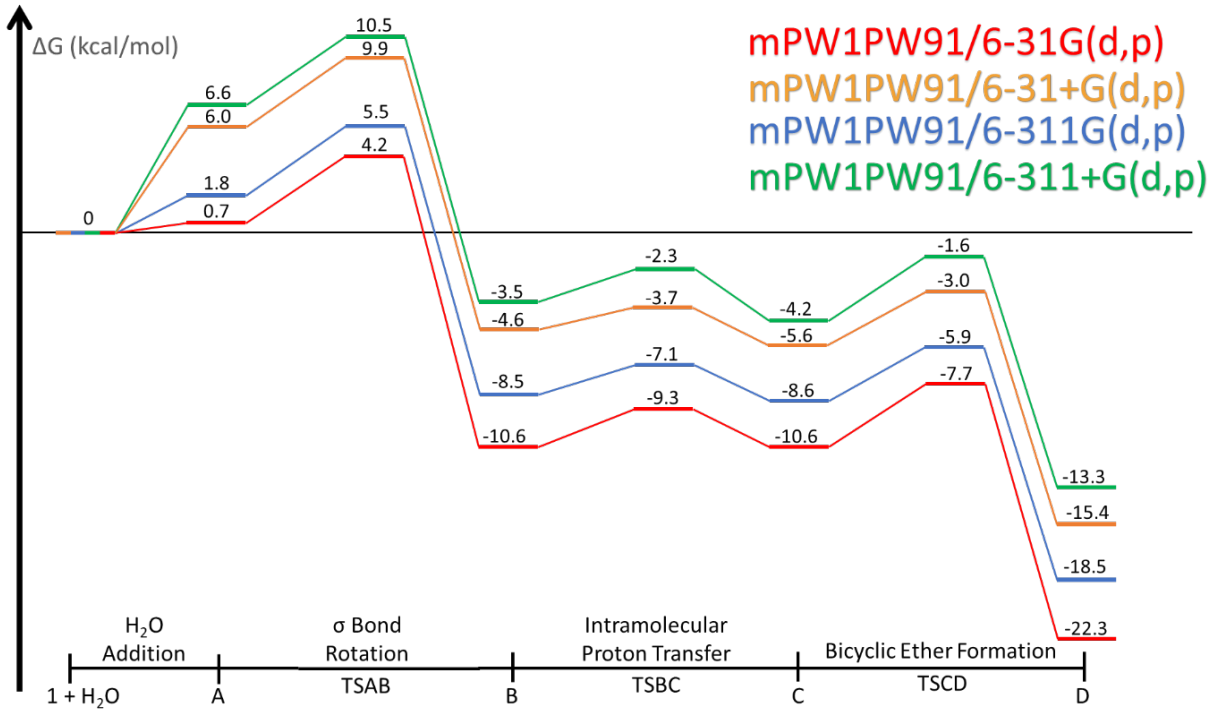

Figure S1. Reaction coordinate diagram from terpinyl cation to protonated 1,8-cineole under multiple levels of theories.

Intermediate	Conformer Number	Free Energy (kcal/mol)
FPP Analog	1-36	N/A
(R)-1_Axial	3, 4, 6	1.04, 0.77, 0.65
(R)-1_Equatorial	2, 5, 7	-0.35, -0.05, 3.30
(S)-1_Axial	2	0.78
(S)-1_Equatorial	3, 4, 5	-0.34, 0.66, 3.32
(R)-A_Axial	2	1.36
(R)-A_Equatorial	3, 4	1.39, -3.12
(S)-A_Axial	4	-0.14
(S)-A_Equatorial	2, 3, 5	1.23, 1.20, -3.30
(R)-B	N/A	N/A
(S)-B	N/A	N/A
C	2, 3, 4	4.28, 2.402, 2.399
D	N/A	N/A

Table S2. Free Energies (in kcal/mol) of conformations kept for docking simulations.

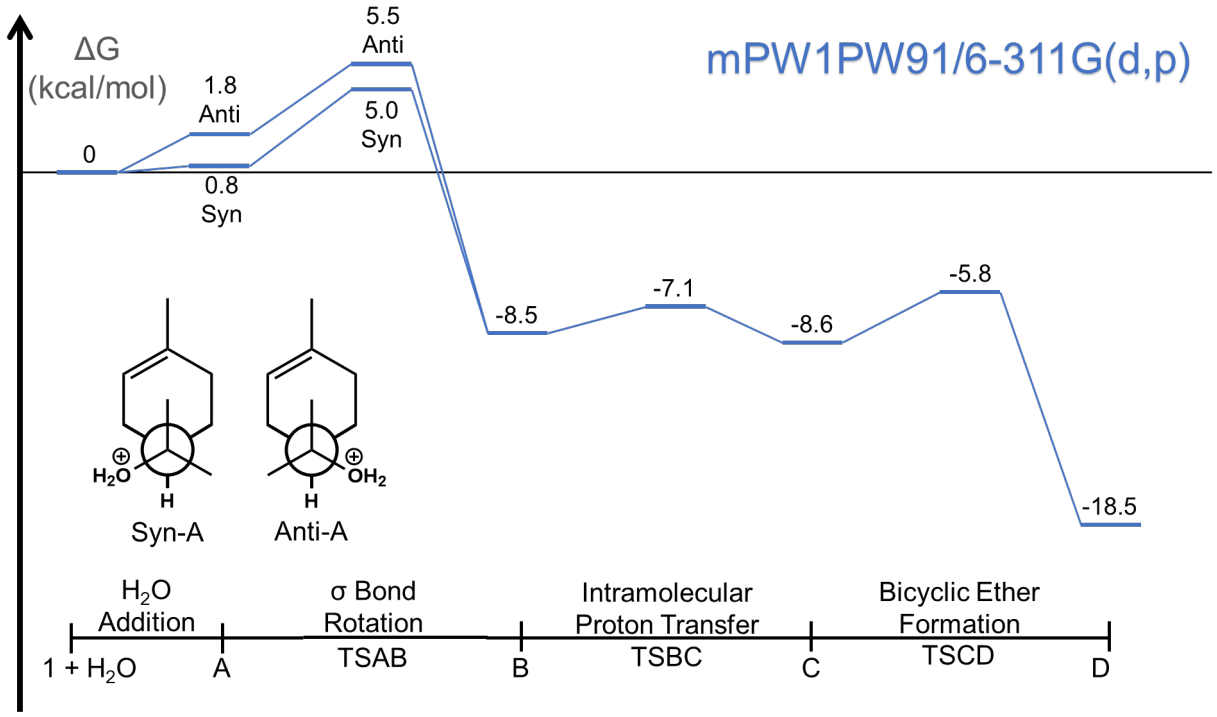

Figure S2. Reaction coordinate diagram depicting both syn addition (Syn-A) and anti-addition (Anti-A) of water to terpinyl cation.

Docking Studies

The Terdockin process highlights the use of chemically meaningful constraints to highlight intermolecular interactions and mechanistically relevant interactions. In this experiment, the constraints are meant to keep the ligands (magnesium/diphosphate group and terpene intermediate) in the active site in an orientation that is supported by previous crystallographic and mechanistic data. Tables S4 – S8 represent constraints used when docking intermediates in 1,8-cineole synthase in *Streptomyces clavuligerus*. Tables S14 – S18 represent constraints used when docking intermediates in 1,8-cineole synthase in *Salvia fruticosa*. Distance constraints are in angstroms (Å) while angle

constraints are in degrees (°). The atom on the residue follows the naming convention as seen in the 5NX7 or 2J5C PDB file.

Docking Scores from Scheme 1 in *Streptomyces clavuligerus*

Table S3 below shows the number of successful poses for each intermediate in each potential orientation. **1** and **A** have two main conformations, noted axial and equatorial with respect to the isopropyl branch on the cyclohexene. **B** is preferably axial due to the strong donor-acceptor interaction. Lastly, **D** is going to be deprotonated by either the enzyme or the diphosphate, which is shown in the last row.

***R* Isomer**			***S* Isomer**			
	$O_A \rightarrow O_A$			**$O_A \rightarrow O_A$**		
(R)-1	Axial: 68	Equatorial: 76	(S)-1	Axial: 58	Equatorial: 82	
A	Axial: 61	Equatorial: 70	A	Axial: 65	Equatorial: 68	
B	80		B	70		
C	80		C	75		
D	42		D	46		
Base	**O_A**		**Base**	**O_A**	**Asn220**	**Asn305**
D	31		D	37	37	37

Table S3. Number of successful docking simulations per intermediate and docking mode tested. For **1**, specific constraints between the reactive water, asparagine, and tryptophan were present. For **D**, the constraint between the oxonium ion and the labelled base was present.

Constraint Information for 1,8-cineole Synthase in *Streptomyces clavuligerus*

Mg_B Constrained to Asp81 (OD2)		
Constraint	Optimal Value	Allowed Deviation
Distance	2.1	0.4
Angle	140.0	12.5
Mg_C Constrained to Asp81 (OD1)		
Constraint	Optimal Value	Allowed Deviation
Distance	2.2	0.4
Angle	130.0	12.5
Mg_A Constrained to Glu228 (OE2)		
Constraint	Optimal Value	Allowed Deviation
Distance	1.9	0.4
Angle	130.0	12.5
O_B Constrained to Arg314 (NH2)		
Constraint	Optimal Value	Allowed Deviation
Distance	3.3	0.4
Angle	90.0	12.5
O_B Constrained to Tyr315 (OH)		
Constraint	Optimal Value	Allowed Deviation
Distance	2.3	0.4
Angle	110.0	20.0

Table S4. Constraints between the magnesium/diphosphate complex and the surrounding residues.

H_2O Constrained to Asn305 (OD1)		
Constraint	Optimal Value	Allowed Deviation
Distance	2.9	0.4
Angle	100.0	10.0
H_2O Constrained to Trp58 (O1)		
Constraint	Optimal Value	Allowed Deviation
Distance	3.0	0.4
Angle	N/A	N/A
H_2O Constrained to Terpene (Carbocation)		
Constraint	Optimal Value	Allowed Deviation
Distance	N/A	N/A
Angle	90.0	15.0

Table S5. Constraints between the reactive water and surrounding residues when docking intermediate (*R*)-**1** or (*S*)-**1**.

H_2O Constrained to Arg174 (NH1)		
Constraint	Optimal Value	Allowed Deviation
Distance	3.1	0.4
Angle	130.0	5.0
H_2O Constrained to Phe179 (N)		
Constraint	Optimal Value	Allowed Deviation
Distance	2.5	0.4
Angle	140.0	5.0

Table S6. Constraints between the structural water residue and the surrounding residues.

Asp221 (OD1) Constrained to Arg174 (NH2)		
Constraint	Optimal Value	Allowed Deviation
Distance	2.8	0.4
Angle	110.0	5.0
Asn220 (OD1) Constrained to Arg174 (NH2)		
Constraint	Optimal Value	Allowed Deviation
Distance	3.2	0.4
Angle	156.0	5.0
Phe179 (CD2) Constrained to Phe77 (CE1)		
Constraint	Optimal Value	Allowed Deviation
Distance	3.4	0.4
Angle	82.5	10.0
Phe179 (CZ) Constrained to Lue183 (CG)		
Constraint	Optimal Value	Allowed Deviation
Distance	3.6	0.4
Angle	N/A	N/A
Asn217 (ND2) Constrained to Asn213 (OD1)		
Constraint	Optimal Value	Allowed Deviation
Distance	2.6	0.3
Angle	N/A	N/A

Table S7. Constraints between amino acids to mimic protein-protein interactions.

Met180 Constrained to Terpene		
Constraint	Optimal Value	Allowed Deviation
Distance	4.0	1.0
Angle	N/A	N/A
Terpene (C_A) Constrained to DPP		
Constraint	Optimal Value	Allowed Deviation
Distance	3.0	0.5
Angle	N/A	N/A
Terpene (C_B) Constrained to DPP		
Constraint	Optimal Value	Allowed Deviation
Distance	3.0	0.5
Angle	N/A	N/A

Table S8. Constraints between the carbon-backbone and surrounding residues to mimic various docking modes.

Docking Studies: Deprotonation to 1,8-Cineole in *Streptomyces clavuligerus*

There are three possible catalytic bases, Asn220, Asn305, and O_A, for the *S* pathway. Based on the pooled filtering process, there is a strong preference towards Asn220 and O_A (Table S9). RMSD analysis (Figure S3) contrasts this, as Asn305 had the lowest (or most favorable) value. This is a debate between kinetics and thermodynamics. When Asn305 is the catalytic base, the heavy atoms of **C** deviate less but are in a less favorable binding orientation. When O_A or Asn220 is the catalytic base, the heavy atoms deviate more but end in a more favorable binding orientation. With little information about the energetic barrier in the enzyme to go between **C**, **D**, and **P**, Terdockin reached its limit in answering this mechanistic question.

S Isomer			
Base	**$O_A \rightarrow O_A$**	**Protein Energy (REU)**	**Interface Energy (REU)**
Asn220	68	-969.53 ± 3.55	-9.74 ± 0.07
Asn305	3	-960.70 ± 1.44	-9.97 ± 0.12
O_A	41	-965.87 ± 2.88	-9.72 ± 0.07

Table S9. Number of successful docking modes for each of the three potential catalytic bases of the *S* pathway. The average Protein Energy and Interface Energy is also present in Rosetta Energy Units (REU).

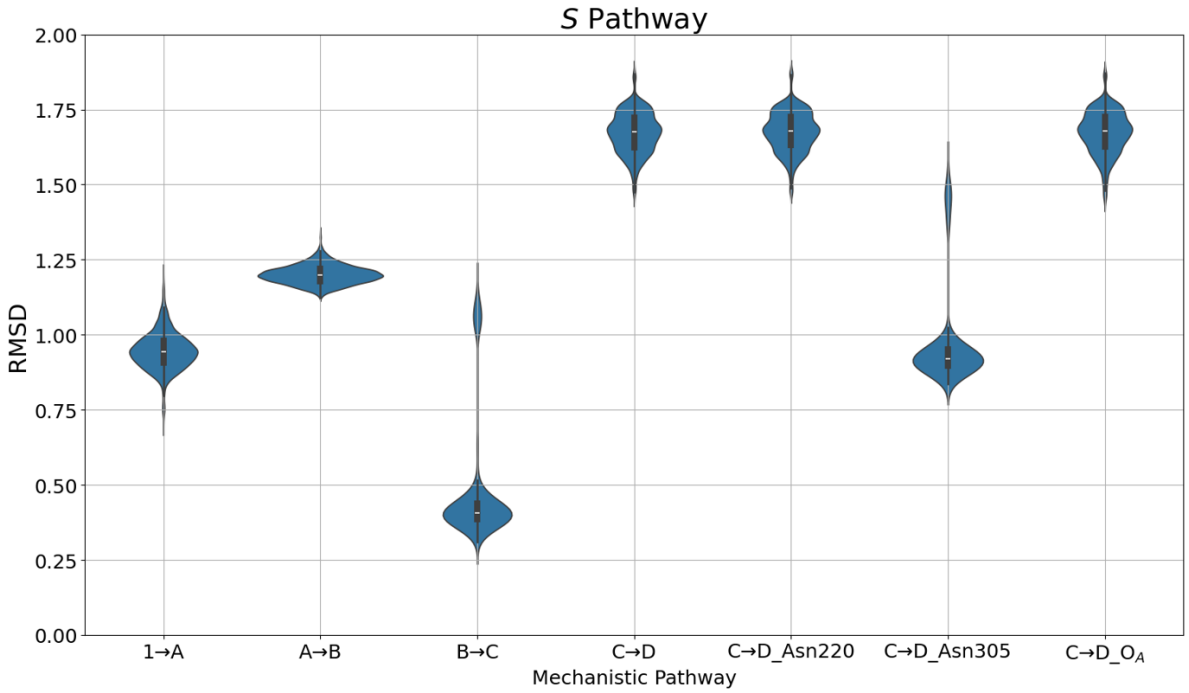

Figure S3. RMSD analysis of the *S* pathway in *Streptomyces clavuligerus* with the $O_A \rightarrow O_A$ carbon-diphosphate binding orientation. The x-axis represents the transition between the specific intermediates. No_Csts represents **D** with no constraints to a catalytic base.

Docking Studies: α-Terpineol Formation in *Streptomyces clavuligerus*

Interestingly, these various bases can also be the cause for the formation of the minor α-terpineol side-product. This led to 24 possibilities since there are 2 mechanistic pathways, 3 acidic protons, and 4 different bases present (Table S10). When referring to the acidic protons on the carbon, Figure S4 shows the labelling pattern used. Based on the heatmap, O_B cannot be the base, and there is a strong preference for O_A and Asn220.

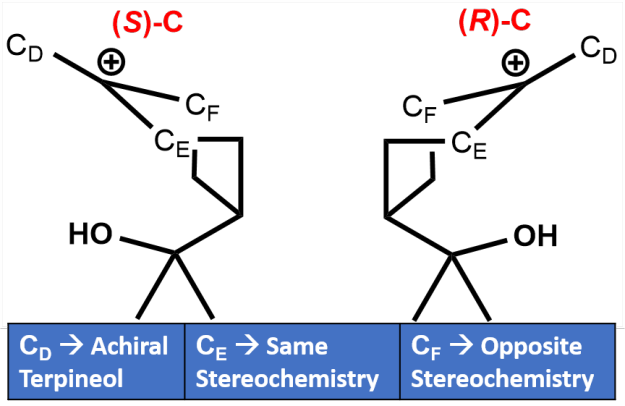

Figure S4. Carbon labelling method for distance and docking analysis

***R* Pathway**				
Diphosphate – Terpene Bond	O_A – O_A			
Acidic Proton on C_D	223	0	65	6
Acidic Proton on C_E	231	0	247	2
Acidic Proton on C_F	186	0	242	85
Basic Oxygen Atom	O_A	O_B	Asn220	Asn305
***S* Pathway**				
Diphosphate – Terpene Bond	O_A – O_A			
Acidic Proton on C_D	183	2	226	0
Acidic Proton on C_E	213	0	231	0
Acidic Proton on C_F	142	0	157	186
Basic Oxygen Atom	O_A	O_B	Asn220	Asn305

Table S10. Heat map of 2500 simulations for the 24 possible docking modes that would lead to the side product α-terpineol.

To reduce the number of possibilities, distances were measured between C_D, C_E, C_F and the basic atoms shown in Table S11. The distance between the alcohol functional group and the carbocation is also shown. From this table, 2 docking modes were chosen from each pathway (one where an asparagine oxygen atom is the base and the second where a diphosphate oxygen atom is the base) for a total of 4 side-product formation possibilities highlighted in orange. From these distances, the *R* pathway appears to lead to mixed stereochemistry, whereas the *S* pathway leads to (*S*)-**T**.

R Pathway			*S* Pathway		
Atoms of Interest	**Distance (Å)**	**Std (Å)**	**Atoms of Interest**	**Distance (Å)**	**Std (Å)**
Oxygen → C^+	3.35	0.00	Oxygen → C^+	3.50	0.34
Asn220 → C_D	5.81	0.08	Asn220 → C_D	4.82	0.56
Asn220 → C_E	4.71	0.13	Asn220 → C_E	4.00	0.89
Asn220 → C_F	4.27	0.11	Asn220 → C_F	5.91	0.59
Asn305 → C_D	7.31	0.06	Asn305 → C_D	6.34	0.30
Asn305 → C_E	6.16	0.04	Asn305 → C_E	5.85	0.87
Asn305 → C_F	4.92	0.06	Asn305 → C_F	5.79	0.76
O_A → C_D	3.80	0.01	O_A → C_D	3.87	0.37
O_A → C_E	2.96	0.01	O_A → C_E	2.91	0.37
O_A → C_F	4.49	0.01	O_A → C_F	4.42	0.33
O_B → C_D	7.56	0.03	O_B → C_D	7.16	0.57
O_B → C_E	5.22	0.03	O_B → C_E	5.02	0.56
O_B → C_F	6.67	0.05	O_B → C_F	6.74	0.41

Table S11. Average distance with standard deviation in Å between carbons with acidic protons and various basic atoms.

To further explore these 4 docking modes, a constraint of 2.7 ± 0.4 Å was set up between C_D, C_E, or C_F, and the basic oxygen atom (Table S12). Observing the active site, the constraint did not automatically determine the catalytic base. In Figure S5, C_F of (*R*)-**C** was constrained to Asn220. This would lead to (*S*)-**T**. However, the distance between O_A and C_E is 0.1 Å shorter and would lead to (*R*)-**T**.

***R* Pathway**		
Diphosphate – Terpene Bond	O_A – O_A	
Acidic Proton on Carbon	C_D	C_D
Basic Oxygen Atom	Asn220	O_A
	57	88
***S* Pathway**		
Diphosphate – Terpene Bond	O_A – O_A	
Acidic Proton on Carbon	C_D	C_D
Basic Oxygen Atom	Asn220	O_A
	54	83

Table S12. Number of successful simulations that could lead to the α-terpineol product.

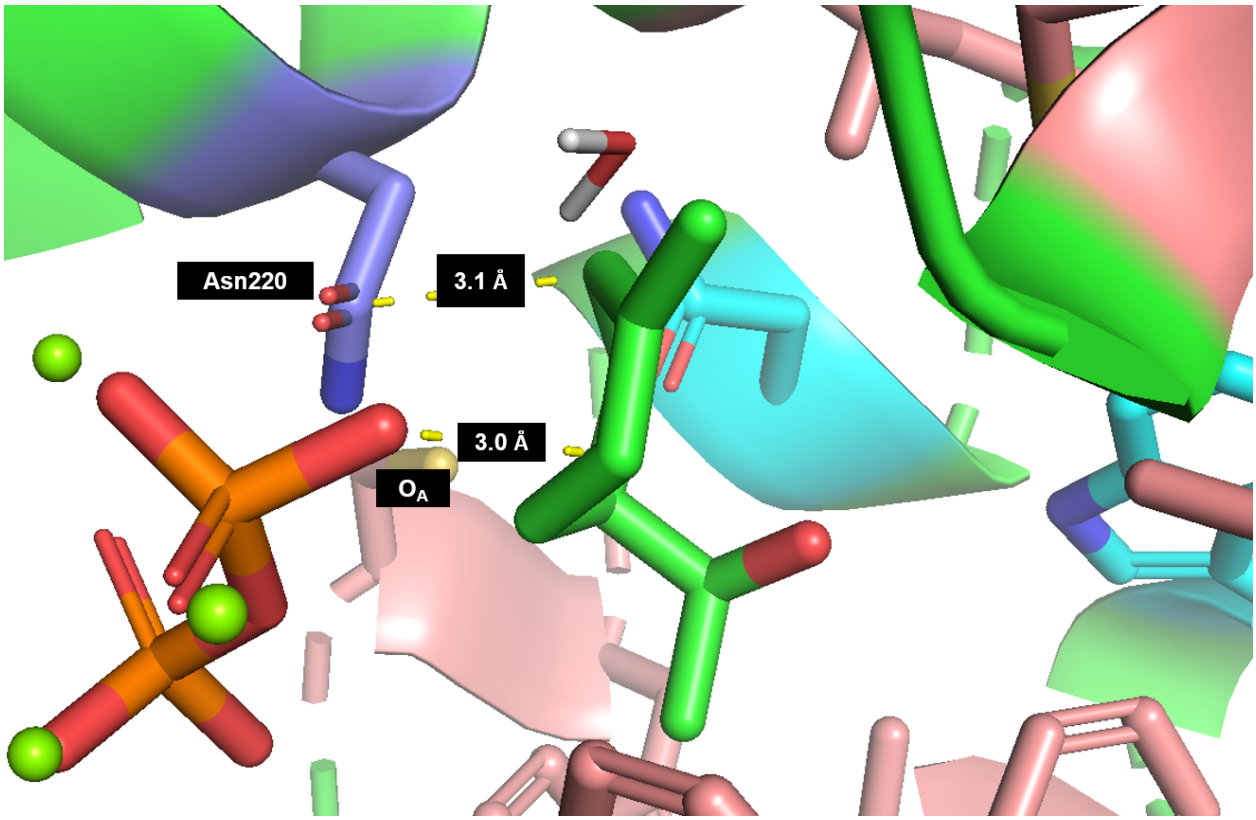

Figure S5. Example of successful docking simulation of (*R*)-**C**.

Phenomena such as these led us to calculate the distance between basic oxygen atoms and carbons with acidic protons for the 4 docking modes to see whether multiple bases present could work simultaneously (Table S13). For the *R* pathway, there can be a mix of (*R*)-**T** and (*S*)-**T**. When analyzing the *S* pathway, there is always a distance preference to form the (*S*)-**T**, which corroborates with previous analysis.

Isomer_Base_Carbon	***S*_Asn220_C_E**						***S*_O_A–C_E**					
Basic Oxygen Atom	Asn220			O_A			Asn220			O_A		
Carbon with Acidic Protons	C_D	C_E	C_F	C_D	C_E	C_F	C_D	C_E	C_F	C_D	C_E	C_F
Distance (Å)	4.25	2.99	4.31	3.94	3.10	5.30	4.81	4.01	5.91	3.82	2.72	4.58
Standard Deviation (Å)	0.22	0.36	0.22	0.01	0.02	0.28	0.19	0.51	0.31	0.11	0.30	0.24
Isomer_Base_Carbon	***R*_Asn220_C_F**						***R*_O_A–C_E**					
Basic Oxygen Atom	Asn220			O_A			Asn220			O_A		
Carbon with Acidic Protons	C_D	C_E	C_F	C_D	C_E	C_F	C_D	C_E	C_F	C_D	C_E	C_F
Distance (Å)	5.05	3.97	3.10	4.05	2.67	4.43	5.79	4.43	4.36	3.76	2.85	4.58
Standard Deviation (Å)	0.15	0.11	0.01	0.26	0.33	0.04	0.08	0.46	0.14	0.07	0.16	0.14

Table S13. Distance between C_D, C_E, C_F and two potential bases that could lead to various isomers of the terpineol side product.

Constraint Information for 1,8-cineole Synthase in *Salvia fruticosa*

Reactive H_2O Constrained to Structural H_2O		
Constraint	Optimal Value	Allowed Deviation
Distance	2.7	0.4
Reactive H_2O Constrained to Asn338 (OD1)		
Constraint	Optimal Value	Allowed Deviation
Distance	2.7	0.5
Reactive H_2O Constrained to Terpene (Carbocation)		
Constraint	Optimal Value	Allowed Deviation
Distance	3.0	0.5

Table S14. Constraints between the reactive water and surrounding residues when docking intermediate (*R*)-**1** or (*S*)-**1**.

Structural H_2O Constrained to Asn338 (ND2)		
Constraint	Optimal Value	Allowed Deviation
Distance	2.2	0.5
Structural H_2O Constrained to Asn338 (OD1)		
Constraint	Optimal Value	Allowed Deviation
Distance	2.7	0.5

Table S15. Constraints between the structural water residue and the surrounding residues.

Asp324 (OD1) Constrained to Arg308 (NE)		
Constraint	Optimal Value	Allowed Deviation
Distance	2.9	0.4
Asp468 (OD1) Constrained to Arg486 (NH1)		
Constraint	Optimal Value	Allowed Deviation
Distance	2.9	0.4

Table S16. Constraints between amino acids to mimic protein-protein interactions.

Ile446 Constrained to Terpene		
Constraint	Optimal Value	Allowed Deviation
Distance	4.2	1.0
Terpene (C_A) Constrained to DPP		
Constraint	Optimal Value	Allowed Deviation
Distance	3.0	0.5
Terpene (C_B) Constrained to DPP		
Constraint	Optimal Value	Allowed Deviation
Distance	3.0	0.5

Table S17. Constraints between the carbon-backbone and surrounding residues to mimic various docking modes.

Mg_A Constrained to Asp345 (OD2)		
Constraint	Optimal Value	Allowed Deviation
Distance	2.4	0.4
Mg_B Constrained to Asp345 (OD1)		
Constraint	Optimal Value	Allowed Deviation
Distance	2.3	0.4
Mg_B Constrained to Asp349 (OD2)		
Constraint	Optimal Value	Allowed Deviation
Distance	2.4	0.4
Mg_C Constrained to Asp489 (OD1)		
Constraint	Optimal Value	Allowed Deviation
Distance	2.6	0.4
Mg_C Constrained to Glu497 (CD)		
Constraint	Optimal Value	Allowed Deviation
Distance	4.2	0.4
O_B Constrained to Arg308 (NH1)		
Constraint	Optimal Value	Allowed Deviation
Distance	2.7	0.4
O_A Constrained to Arg486 (NH2)		
Constraint	Optimal Value	Allowed Deviation
Distance	2.7	0.4
O_C Constrained to Arg486 (NH2)		
Constraint	Optimal Value	Allowed Deviation
Distance	2.7	0.4

Table S18. Constraints between the magnesium/diphosphate complex and the surrounding residues.

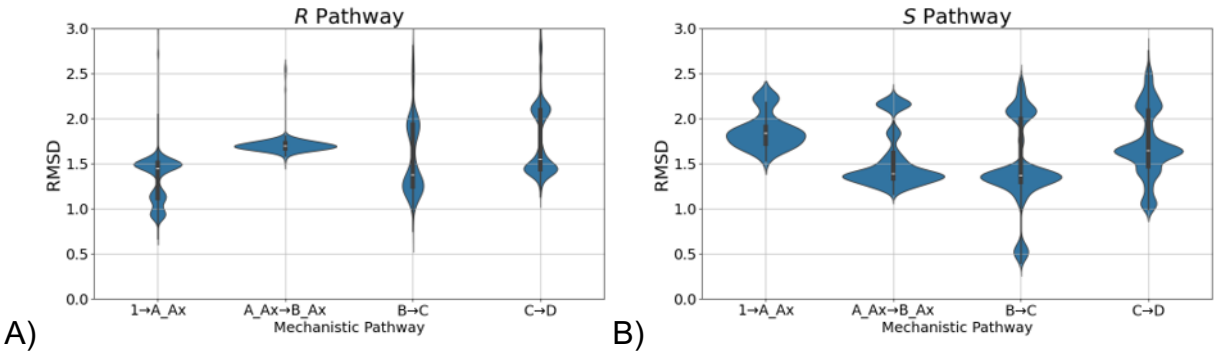

Figure S6. RMSD analysis of *R* and *S* pathway in *Salvia fruticosa* with the O_B → O_B carbon-diphosphate binding orientation. The x-axis represents the transition between the specific intermediates.

Docking Studies: Water Addition Step in *Salvia fruticosa*

Another phenomenon of interest occurred when docking the equatorial and axial conformers of (*S*)-**1**. Some successful orientations showed the reactive water molecule facing the empty p orbital on the side opposite to the alkene, while others showed the reactive water molecule more on the syn side. (Figure S7) Water attaching on the syn side relative to the alkene was not seen in the (*R*)-**1** compounds, nor was it seen in any of the terpinyl cation docking with 1,8-cineole synthase in *Streptomyces clavuligerus*. This interesting observation is a potential mechanistic difference between isozymes, but it is speculative due to the use of generative models and *Terdockin*. The σ-bond rotation of Syn-**A** to **B** has a barrier of 4.2 kcal/mol, which is feasible in the terpene active site (Figure S2). Docking of Intermediate (*S*)-**A** with the alcohol moiety syn and anti to the alkene exclusively showed the anti-orientation. While both orientations can fit in the active site consistently, the enzymatic preference of the alcohol being anti to the alkene in

Intermediate (*S*)-**A** while being syn to the alkene in Intermediate (*S*)-**1** may be an artifact of *TerDockin* and using generative models.

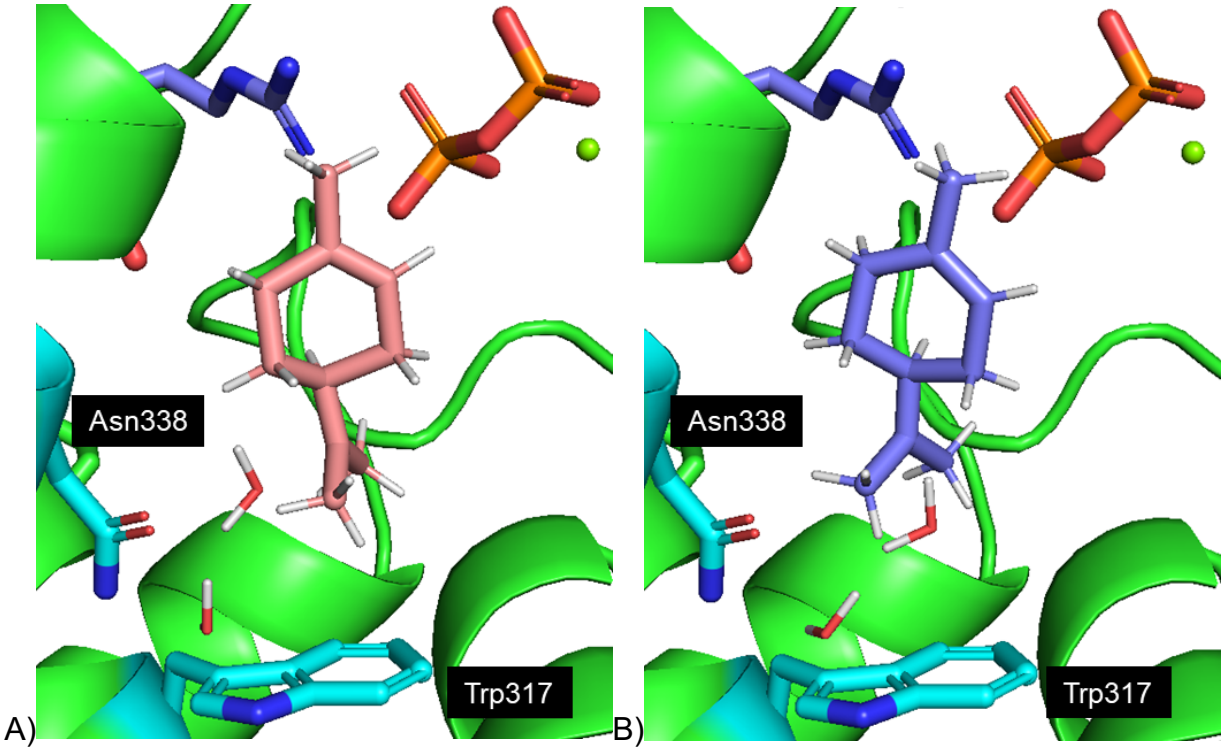

Figure S7. Docked poses of equatorial conformers of (*S*)-**1**. A) Reactive water is closer to the carbocation on the face anti to the alkene. B) Reactive water is closer to the carbocation on the face syn to the alkene.

Python Scripts Used for Data Extraction and Figure Making:

run_rms_parallel.py

```
import os
import subprocess
from itertools import product
from multiprocessing import Pool

def run_calc_rms(pair):
    model1, model2 = pair
    try:
        subprocess.run(
     ["python", "calc_rms.py", model1, model2],
     check=True,
    )
    except subprocess.CalledProcessError:
        print(f"Failed: {model1} {model2}")

def main():
    folder1 = "[Input_Folder_Name]"
    folder2 = "[Input_Folder_Name]"

    pdbs1 = [os.path.join(folder1, f) for f in os.listdir(folder1) if
f.endswith(".pdb")]
    pdbs2 = [os.path.join(folder2, f) for f in os.listdir(folder2) if
f.endswith(".pdb")]

    pairs = list(product(pdbs1, pdbs2))
    print(f"Submitting {len(pairs)} pairs for processing...")

    with Pool(processes=[Input_Value]) as pool:
        pool.map(run_calc_rms, pairs)

if __name__ == "__main__":
    main()
```

calc_rms.py

```
import sys
import Bio.PDB
import numpy as np

# Top : Bottom
mymappingdict = {
[Input_Atom_Number_from_Folder_1]:[Input_Atom_Number_from_Folder_2] }

# based on code from http://combichem.blogspot.com/2013/08/aligning-pdb-
structures-with-biopython.html?m=1
# give the function the pdb to align (pdb1) the structure to align to (orig)
and the name you want the new structure to write out as (can be overridden
for each iteration)
def align_prot(pdb1, orig):
    atoms_to_be_aligned_SgCS = range(1, [Input_Range])
    pdb_parser = Bio.PDB.PDBParser(QUIET=True)
```

```
    ref_structure = pdb_parser.get_structure("ref", orig)
    sample_structure = pdb_parser.get_structure("sample", pdb1)

    ref_model = ref_structure[0]
    sample_model = sample_structure[0]

    ref_atoms = []
    sample_atoms = []

    for ref_chain in ref_model:
        if ref_chain.get_id()[0] == '[Input_Info]':
            for ref_res in ref_chain:
                if ref_res.get_id()[1] in atoms_to_be_aligned_SgCS:
                    ref_atoms.append(ref_res['CA'])

    for sample_chain in sample_model:
        if sample_chain.get_id()[0] == '[Input_Info]':
            for sample_res in sample_chain:
                if sample_res.get_id()[1] in atoms_to_be_aligned_SgCS:
                    sample_atoms.append(sample_res['CA'])

    super_imposer = Bio.PDB.Superimposer()
    super_imposer.set_atoms(ref_atoms, sample_atoms)
    super_imposer.apply(sample_model.get_atoms())

    return sample_structure

#Function that calculates the RMS of two PDB's that have both been aligned to
another structure
def rms(struct1, struct2):
    lig1 = {}
    lig2 = {}

    for atom in struct1.get_atoms():
        if atom.get_parent().get_resname() == '[Input_Info]' and atom.element
in ('[Input_Element_Type]', '[Input_Element_Type]'):
            lig1[atom.get_name()] = atom.coord

    for atom in struct2.get_atoms():
        if atom.get_parent().get_resname() == '[Input_Info]' and atom.element
in ('[Input_Element_Type]', '[Input_Element_Type]'):
            lig2[atom.get_name()] = atom.coord

    rms_tot = 0
    count = 0
    for key in lig1:
        try:
            target = mymappingdict[key]
            rms_i_j = distance(lig1[key], lig2[target])
            rms_tot += rms_i_j
            count += 1
        except KeyError:
            print(f"Missing mapping for atom {key} skipped.")
            continue

    return rms_tot / count if count > 0 else float('nan')
```

```
def distance(coords1, coords2): # helper function that calculates distance
between 2 points in 3D space
    difx = coords1[0] - coords2[0]
    dify = coords1[1] - coords2[1]
    difz = coords1[2] - coords2[2]
    return np.sqrt(difx**2 + dify**2 + difz**2)


def main():
    model1_path = sys.argv[1]
    model2_path = sys.argv[2]
    crystal = '[Input_Info].pdb' #Change if Neccessary

    struct1_aligned = align_prot(model1_path, crystal)
    struct2_aligned = align_prot(model2_path, crystal)

    my_rms = rms(struct1_aligned, struct2_aligned)

    with open('[Input_Info].txt', 'a') as fh: #Change if Neccessary
        fh.write('%s %s %s \n' % (model1_path, model2_path, my_rms))

main()
```

Distance_Pymol_Script.py

```
import os
import math

def get_pdb_files():
    """Returns a list of all PDB files in the current directory."""
    return [f for f in os.listdir() if f.endswith(".pdb")]

def extract_atom_by_serial(pdb_file, serial_number):
    """Returns the line matching a given atom serial number from ATOM or
HETATM lines."""
    serial_str = f"{serial_number:>5}"  # PDB serial number column (7-11)

    with open(pdb_file, "r") as file:
        for line in file:
            if line.startswith(("ATOM  ", "HETATM")) and line[6:11] ==
serial_str:
                return line.strip()
    return None

def extract_xyz_from_line(line):
    """Extracts XYZ coordinates from a PDB line using fixed column
positions."""
    try:
        x = float(line[30:38].strip())
        y = float(line[38:46].strip())
        z = float(line[46:54].strip())
        return x, y, z
    except ValueError:
```

```
        return None

def calculate_distance(coords1, coords2):
    return math.sqrt(sum((a - b) ** 2 for a, b in zip(coords1, coords2)))

def main():
    # Get user input for serial numbers
    try:
        serial1 = int(input("Enter first atom serial number: "))
        serial2 = int(input("Enter second atom serial number: "))
    except ValueError:
        print("Invalid input. Please enter integer serial numbers.")
        return

    pdb_files = get_pdb_files()
    if not pdb_files:
        print("No PDB files found.")
        return

    distances = []

    for pdb_file in pdb_files:
        line1 = extract_atom_by_serial(pdb_file, serial1)
        line2 = extract_atom_by_serial(pdb_file, serial2)

        if not line1 or not line2:
            print(f"{pdb_file}: Missing atom {serial1} or {serial2}.
Skipping...")
            continue

        coords1 = extract_xyz_from_line(line1)
        coords2 = extract_xyz_from_line(line2)

        if not coords1 or not coords2:
            print(f"{pdb_file}: Error extracting coordinates. Skipping...")
            continue

        distance = calculate_distance(coords1, coords2)
        distances.append(f"{pdb_file}: {distance:.2f}")

    print("\nFinal Distances:")
    print("\n".join(distances))

if __name__ == "__main__":
    main()
```

Violin_Plots.py

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Define the folder path containing Excel files
```

```
folder_path = r"[Path_to_Excel]"

# List all Excel files in the folder
excel_files = [file for file in os.listdir(folder_path) if
file.endswith(".xlsx") or file.endswith(".xls")]

# Read data from each Excel file and concatenate into a single DataFrame
df_list = []
for file in excel_files:
    file_path = os.path.join(folder_path, file)
    df = pd.read_excel(file_path)
    df_list.append(df)

# Combine all data into a single DataFrame
if df_list:
    data = pd.concat(df_list, ignore_index=True)
else:
    raise ValueError("No Excel files found in the specified folder.")

# Specify the columns to plot (modify this based on your dataset)
columns_to_plot = ["[Column_Name]"]  # Replace with actual column names in
Excel

# Check if columns exist in the data
missing_cols = [col for col in columns_to_plot if col not in data.columns]
if missing_cols:
    raise ValueError(f"Columns {missing_cols} not found in the dataset.")

# Rename columns for display (supports mathtext subscripts)
display_names = {"[Column_Name_from_Excel]:[Column_Name_In_Graph]"} #$_{A}$
is subscript
data = data.rename(columns=display_names)
columns_to_plot = [display_names.get(c, c) for c in columns_to_plot]

# Reshape data for Seaborn
df_melted = data.melt(value_vars=columns_to_plot, var_name="Category",
value_name="Value")

# Create a violin plot with multiple columns
plt.figure(figsize=(10, 6))
sns.violinplot(x="Category", y="Value", data=df_melted)

# Customize the plot
plt.ylim([0, 4.0])
plt.title("[Title_Name]", fontsize=26) # $Value$ is italic
plt.xlabel("[X-Axis]", fontsize=22)
plt.ylabel("[Y-Axis]", fontsize=22)
plt.tick_params(axis='both', which='major', labelsize=18)
plt.grid(True)

# Show the plot
plt.show()
```

Rosetta all_data.py

```
import re
import os

def get_data(directory):
    """Read and parse the score.sc file in the specified directory."""
    with open('score.sc', 'r') as file:
        header_line = file.readline().strip()
        categories = re.split(r'\s+', header_line)
        data = [re.split(r'\s+', line.strip()) for line in file]
    return categories, data

def extract_data(categories, data, category_names):
    """Extract all rows of data based on category names."""
    indices = [categories.index(category) for category in category_names]
    return [[row[i] for i in indices] for row in data]

def main():
    # Loop through folders numbered 1 to [INPUT_HERE]
    for x in range(1, [INPUT_HERE]):
        directory = str(x)

        # Check if the directory exists before proceeding
        if os.path.isdir(directory):
            os.chdir(directory)  # Navigate to the folder

            categories, data = get_data(directory)

            # Extract all data for the categories of interest
            my_categories_cst = ['all_cst']
            my_categories_total_energy = ['total_score']
            number = [INPUT_HERE]  # Adjust the interface number if necessary
            my_categories_interface_energy = [f'SR_{number}_interf_E_1_2']

            cst_data = extract_data(categories, data, my_categories_cst)
            total_energy_data = extract_data(categories, data,
my_categories_total_energy)
            interface_energy_data = extract_data(categories, data,
my_categories_interface_energy)

            # Loop through the extracted data and print the values
            for i in range(len(cst_data)):
                total_energy = total_energy_data[i][0]  # Get the
corresponding total energy for the current row
                cst_value = cst_data[i][0]  # Get the CST value for the
current row
                interface_energy = interface_energy_data[i][0]  # Get the
interface energy for the current row

                # Simplified output: just print the values separated by
commas
                print(f"{total_energy}, {cst_value}, {interface_energy}")

            os.chdir('..')  # Go back to the parent directory
        else:
            print(f"Directory {directory} does not exist, skipping.")
if __name__ == '__main__':
    main()
```

File_Renaming_Script.py

```
import os

def rename_and_cleanup_files():
    parent_directory = os.getcwd()  # Get the script's current directory

    # Get a sorted list of subdirectories, sorting them numerically
    folders = sorted(
        [f for f in os.listdir(parent_directory) if
os.path.isdir(os.path.join(parent_directory, f))],
        key=lambda x: int(x) if x.isdigit() else float('inf')  # Sort
numerically, non-numeric last
    )

    # List of filenames to exclude from renaming
    exclude_files = {"[Input_File].pdb", "[Input_File].pdb"}

    pdb_files_to_rename = []  # Store (old_path, new_name) tuples
    current_number = 1  # Start numbering from 1

    # Collect file paths for renaming and deletion
    for folder in folders:
        folder_path = os.path.join(parent_directory, folder)
        all_files = os.listdir(folder_path)

        # Filter .pdb files for renaming, excluding specific files
        rename_candidates = sorted([f for f in all_files if
f.endswith(".pdb") and f not in exclude_files])

        for file in rename_candidates:
            old_file_path = os.path.join(folder_path, file)
            new_file_name = f"{current_number}.pdb"
            new_file_path = os.path.join(folder_path, new_file_name)
            pdb_files_to_rename.append((old_file_path, new_file_path))
            current_number += 1  # Increment file number


    # Show summary and ask for confirmation
    if pdb_files_to_rename:
        print(f"\nThis script will rename {len(pdb_files_to_rename)} .pdb
files.")

        # Show a preview of renaming
        if pdb_files_to_rename:
            print("\nFiles to be renamed (first 10 shown):")
            for old_path, new_path in pdb_files_to_rename[:10]:
                print(f"  {os.path.basename(old_path)} ->
{os.path.basename(new_path)}")
            if len(pdb_files_to_rename) > 10:
                print("  ...and more.")


        confirm = input("\nProceed with renaming? (Y/N): ").strip().lower()
        if confirm != 'y':
            print("Operation canceled. No changes were made.")
```

```
            return

        # Rename files
        for old_path, new_path in pdb_files_to_rename:
            os.rename(old_path, new_path)
            print(f"Renamed: {old_path} -> {new_path}")


        print("\nRenaming and cleanup complete!")

    else:
        print("No files found to rename.")

# Run the function
rename_and_cleanup_files()
```

File_Moving_Script.py

```
import os
import shutil

# Get the directory where the script is located
script_dir = os.path.dirname(os.path.abspath(__file__))

# Define the destination folder
destination_folder = os.path.join(script_dir, "[INPUT_HERE]")

# Ensure the destination folder exists
os.makedirs(destination_folder, exist_ok=True)

# Read file names (numbers) from a text file
file_list_path = os.path.join(script_dir, "files_to_move.txt")  # Ensure this
file exists

if not os.path.exists(file_list_path):
    print(f"Error: {file_list_path} not found. Please create the file and add
numbers.")
    exit()

files_to_move = set()

# Read and extract the last number from each line
with open(file_list_path, "r") as f:
    for line in f:
        line = line.strip()
        if not line:
            continue  # Skip empty lines
        parts = line.split('_')
        if parts and parts[-1].isdigit():
            number = str(int(parts[-1]))  # Remove leading zeros
            files_to_move.add(number + ".pdb")

# Loop through folders 1 to []
for filename in list(files_to_move):  # Note, there must be an external .txt
file named files_to_move
```

```
    found = False  # Flag to indicate if the file has been found
    for i in range(1, [INPUT_HERE]):
        folder_path = os.path.join(script_dir, str(i))

        if os.path.exists(folder_path):  # Ensure the folder exists
            file_path = os.path.join(folder_path, filename)
            if os.path.exists(file_path):  # Check if the file exists in the
folder
                shutil.move(file_path, os.path.join(destination_folder,
filename))
                print(f"Moved: {filename} from folder {i} to a new folder")
                found = True
                break  # Stop searching once the file is found

    if not found:
        print(f"Not found: {filename}")

print("File moving process completed!")
```

References

(75) C. Bo, F. Maseras, N. López, *Nat. Catal*, 2018, **1**, 809–810. https://doi.org/10.1038/s41929-018-0176-4.

(76) M. Álvarez-Moreno, C. de Graaf, N. López, F. Maseras, J. M. Poblet, C. Bo, *J. Chem. Inf. Model*, 2015, **55**, 95–103. https://doi.org/10.1021/ci500593j.