After adding more beads, we have solved for all cases that require a new bead

Now we have 110 non-working case that we will have to look closer:

We will use

cd auto_martini

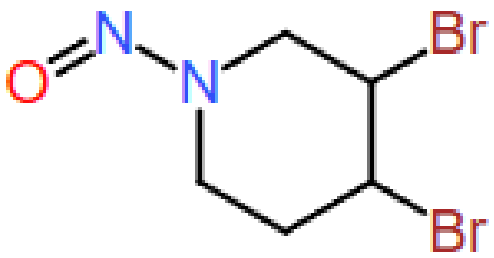
venv\Scripts\activate

python -m auto_martini --smi "....." --mol NAME --top NAME.itp

to see what they got as the solution then we can from there infer

python -m auto_martini --smi "NCC(=O)NCC(=O)NCC(=O)O" --mol 556-33-2 --top 556-33-2.itp

1. 57541-73-8



Solution: ALOGPS can't predict fragment: N=O -> they can’t predict their own 653 picked?

let’s try anyway:

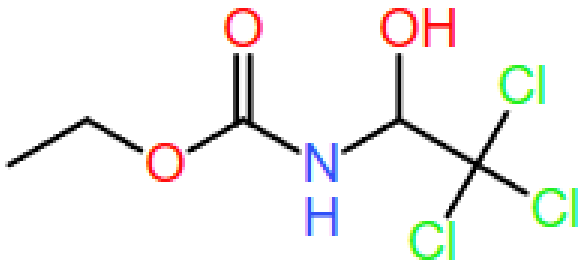
Observation: We have seen N inside of a ring before, but that is the case for a double N=C connection, which we can assign TN6a for

How can we solve this case? 2 solutions

a. assign CN as TN6a regardless
b. merge with the non-ring section as a new section

We should look at more examples for a clearer view

2. 541-79-7



Solution: C(Cl)(Cl)(Cl): C4, CO: P3, NC(O)=O: P2, CC: C5

Observation: This should work in our code, there must be an error that causes this to not work, so let's look closer at my own code

Issue: The issue lies in the fact that our code never accounted for a center atom with 4 branches, so this error expands to large non-ring sections as well.

Fix: We must account for center atom with 4 branches in the non-1-bead-mappable as well

Here is how it is currently working:

1. **Detect "too long"** by calling `is_non_ring_section_1bead_mappable`.
2. **Split off a small connected piece** via `subdivide_non_ring_section` (normal or multi).
3. **Reindex & reset edge flags** on the two new pieces.
4. **Map the first piece with the 1-bead handler** (so it's "done").
5. **Either map the second piece** (if it's now 1-bead-mappable) **or recurse** to split it again.
6. **Update `full_mapping`** so future calls use the new, smaller sections, not the old large one.

We look deeper into the subdivide-non-ring-section and see how it is divided. We can see that the issue is that the code tries to prioritize linear path and will try to make the shortest linear path from the two closest edge points, which is problematic because sometimes doing that will cause 1 atom to be left out if there is a center connection to 3 or more branch path of 1

Fix: To fix this, before we do any breaking, we need to identify all the atoms that are inertly connected to at least 4 other atoms, then, from those atoms, as long as we find an atom that is connected to 3 edges, we know that the ONLY way this will ever work is when this central atom is being combined with all of those edges to form 1 bead

Note that after this step, we have successfully broken this large atom into 2 sections, one is the center atom with the 3 edges that it connects to, and the other one is the rest of the molecule

After that we will do the same thing as we have been doing, like updating the connections, updating mapping, updating edge status, and running is_1_bead_mappable, etc.

def subdivide_non_ring_section_normal(..)

```
# SPECIAL CASE for linear sections (exactly 2 edge nodes)
if len(candidate_edges) == 2:
    if len(section) == 9:...
    elif len(section) == 6:...
```

`We should put this test right after this special case because it is also a special case.`

We have fixed this crucial case, C(Cl)(Cl)(Cl) should now be considered as 1 bead as intended.

Now we have another issue, and this issue is seen a lot in error

The issue is when a linear 5-long bead is considered as 1 bead mappable (which it is not)

To fix this, we must look at how the is-1-bead-mappable is handling this case.

Currently this is the test to see is the section is 1 bead mappable:

1. **Exactly one edge atom** → Return `True`. A single leaf always fits into one bead.
2. **Zero edges** → Error. Every non-ring should have at least one edge.
3. **More than one edge** → For each pair of edges, compute the shortest inner-chain distance (in atom count).
    a. If any pair is "too far apart" (`dist > 4`), return `False` (not 1-bead-mappable).
    b. Otherwise, return `True` (all edge-pairs are at most 4 atoms apart → can collapse into one bead).

After a few trials, it seems that the SMILES in the form:

CCCC(C)CCC will cause the (CCCCC) error

However, CCC(C)CCCC will not and this will detect CCCC, CCCC, perfectly

These look identical; it is not

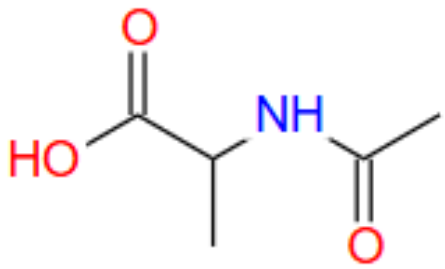
The error seemed to be because somewhere in the code, the CCCCC never went to testing and straight to bead assignment.

Yes, the issue lies at the end of the recursive function, where the first part of the split is considered as guaranteed to be 1-bead-mappable, and it is, EXCEPT FOR THIS CASE, the case where the non-ring section has a center connecting to 3 branches where 2 of the 3 branches has 3 atoms tracing out and the last section only have 1 atom

Our solution: C(Cl)(Cl)(Cl): X2, CO: TN4a, NCO: SN4a, CC=O: SN6 – very different bead assignment but the division is only slightly different

95 CASES LEFT

3. 97-69-8



Solution: CCNC(C)=O: P1, OC=O: P1

Our solution: OC=O: SN4a, CCN: SN4, CC=O: SN4a

This atom proposes a new issue, any bead with the shape CC(C)C(C)C will have the error that is:

Non-ring section with 4 edges does not have an atom with 4 inner connections.

this is because instead of having 4 edges, it has 2 connecting center

Fix: We must add another case in the 1-bead-mappable assignment.

In this case, we will check to see if there are two exactly two atoms that inertly connecting to 3 other atoms where exactly 2 of them are edge atoms

from here, we can inspect 2 parts, the first part is one center connecting to its 2 edge atoms, and the second is the other center connecting to its 2 other edge atoms.

We will have to disconnect the connection between the two center with each other

Then we run the 1-bead mapping function to these 2 new section that we just made

if center_local is None:

three_centers = []

for i, atom in enumerate(section):

if len(atom[4]) == 3:

edge_nbrs = [nbr for (nbr, _) in atom[4] if section[nbr][5]]

if len(edge_nbrs) == 2:

three_centers.append((i, edge_nbrs))

if len(three_centers) == 2:

Unpack the two 3-connected centers

(c1, edges1), (c2, edges2) = three_centers

else:

raise ValueError("Non-ring section with 4 edges has no 4-connected center or valid pair of 3-connected centers.")

I took off the fix edge because we dont need to fix any edge and we dont need to check if the centers are connected, Iwant you to change the following:

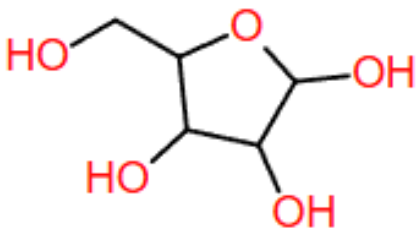
We dont want to call trace linear path here, we can implement our own linear path tracing right here. So we have c1 and edges1, c2 and edges2. We can start with c1 and edges1, we build a string from the one edge to the center to the other edge.

The string will compose of atom_edge1 + bond_edge1_center +atom_center + bond_center_edge2 + atom_edge2

where we will implement the same logic as trace linear path where bond is empty if 1, = if 2 and # if 3. After making the path, we can do the rest as you just did, check for 7 then 11. Apply the same for c2 and edges2

83 CASES LEFT

4. 50-69-1



Solution: CO: P3, OC1CCCO1: SP1, OC1CCOC1: SP1, CO:P3

Our solution: CO: TN6 x3, CO: TN2a (non-ring), CO: TN4a (ring only)

Observation: Well, this solution does not make sense as there are only one ring and it shows that there are 2 ring beads here. I check the score, and it is -3.17 -5.25, off by more than 2 points.

Reflect to our code, we are met with the error: 1-edge non-ring not mappable (non-C)

[lone-O] couldn’t map O@6, neighbor beads = ['N60a&lm_']

our final before the error looks like ['TN4a8W]` ^A', "TP1d'N%e0T", "TP1d'N%e0T", 'N6c7a]H"', 'N6c7a]H"', 'N6c7a]H"', '', 'N6c7a]H"', '', 'TN4a8W]` ^A']

Observation: the 5-bead mapping does not have as diverse way of mapping as the general 6-bead mapping because there are too few of them

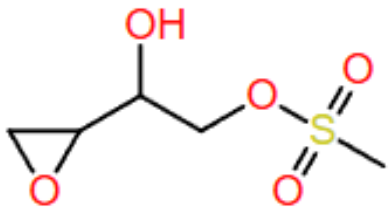
Fix: do a big try with the 5-bead mapping then it that fails then we move on to try to use the 6-bead mapping instead

This is tricky because the final bead will be save even if the trial fail -> we need to make sure that is not the case

We also added 4 more beads as new beads show up when new errors are fixed

77 CASES LEFT

5. 30031-63-1



Solution: C1CO1: SP1, CO: P3, CO: P3, CS(=O)=O: P3

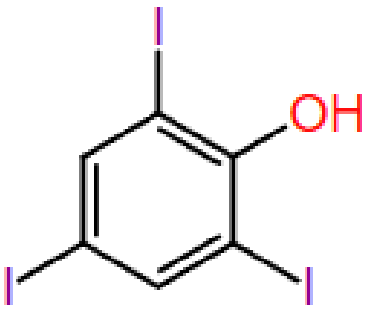
Observation: this one has the 3-ring that we have very limited information about.

All we need to do here is adding the next case for 3-ring: COC: SN4a (as used for 6-ring nodes)

There is a new bead that arises from that (CS(=O)=O: P3) the one that they assigned as P3, we will assign it as SP3 as well to match

Our solution: C1CO1: SN4a, OCCO: P4, CS(=O)=O: SP3

6. 609-23-4



This one does not run using their algorithm

Observation: This one should be able to run if it is non-benzene. The test for benzene is somewhat limited.

Error: mapping section 0 failed: Benzene ring section not fully mappable!

Benzene should already have a section that accounts for this and all cases similar to this.

We need to dive deeper into how benzene maps for a better understanding

Here is the current logic flow of benzene:

- **Step 1 (array0):** Find adjacent carbon-pairs that each connect to some non-ring fragment (section_type ≠ 0). Assign them "TC5e…" or "TC5…" immediately.
- **Step 2:** Actually record those adjacent pairs as `pairs0` and do the bead assignment for each.
- **Step 3 (array1):** Among the remaining unmapped atoms, collect those that either have no outer connections or have "big" outer connections (size ≥ 3 or 2-atom that's not pure CO). If that list is odd, move one atom out → `array2`.
- **Step 4:** Pair up `array1` according to inner-bond connectivity (2, 4, or 6 atoms) and assign either "TC5…" (for C–C pairs) or "TN6a…" (if a nitrogen is involved).
- **Step 5 (array2):** Keep the "leftover" from Step 3; we don't assign them now.
- **Step 6 (array3):** Look for benzene atoms whose outer connection is into a single-atom fragment (section_type 0). For each such atom, pair it either with a neighbor in `array2` (if one exists) or with a neighbor that has "big outer" connections, or else assign a T-type bead if nothing qualifies.

- **Step 7 (array4):** Look for benzene atoms whose outer connection is into a two-atom fragment. If the foreign atom is oxygen and the benzene atom is carbon, assign "SN2a…" to that triplet (benzene C, O, and O's neighbor).
- **Step 8 (array5):** Pair up any still-unmapped atoms in the ring. The logic depends on whether 6, 4, or 2 remain.
- **Final check:** Ensure no atom in the section remains unmapped. If so, raise an error; otherwise, return the updated `final`.

Step 1 and step 2 regards to the TC5e

Step 3-5 is actually crucial to this debug as it will try to map all of the atoms that does not have an outer connection and are connected to another one of their kind.

Here, there is a safety net that prevents this from happening where the two atoms with no outer connections not connected, hence they are moved to the array2 which store these atoms for the next step

Step 7 and beyond is working with the CO special case as well as the rest of the unmapped atoms, not related to this bug

Step 6 is the step where the C=CI bead will be assigned. We will find an issue here.

After a long time of debugging, we have found the issue:

```
print('Im picking bead')

print('element:' + element)

print('Im picking bead')

print('element:' + element)

        print("c")        print(bead)        print("b")        print(bead)        print("a")
print(bead)      print(final)
```

These are all debugging print statements, We have found out that the CC(I) does not start with S but X instead

We need to add this line:

```
    if not key.startswith(kind):
```

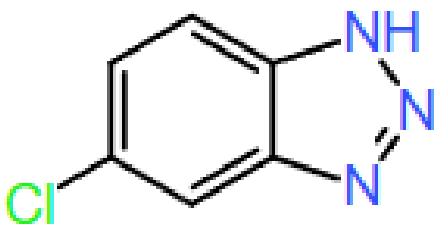
```
        if key.startswith('X') and kind == "S":
            key
        else:
            continue
```

it is now working as intended

Our solution: cO: TN6, ccI: X1 x2, cI: TX1

64 CASES LEFT

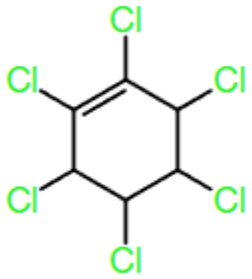
7. 94-97-3



The code from GitHub does not give a result again in this case. This is getting very odd.

Observation: The issue for this case is clear: We can't map N=N. And we don't know what bead to assign for it so this case we will move on.

8. 59229-56-0



Solution: CCl: N0 x3, ClC1CCC=CC1: SC3, ClC1CCCC=C1: SC4, ClC1=CCCCC1:SC3

Our solution: CCl: TX3 x6

Observation: again, this itp file given by the algorithm does not make sense as they duplicate the ring 3 times. It seems like they do not have a method to determine if the ring is already mapped. The score is 5.63 4.37, off by 1.26.

This should not be an issue to the code, it should be able to run smoothly

Here is what the final is at the moment of the error:

['TX3>2Zt;!', 'TX3>2Zt;!', 'TX3?mqA4z', 'TX3?mqA4z', 'SX3:`tma>', 'SX3:`tma>', 'SX3:`tma>', '', 'SX3L].]~s', 'SX3L].]~s', 'SX3L].]~s', '']

This is very odd, and we must investigate further

This is how the current non-benzene-6-ring is laying out:

- **Helper functions** allow us to look up a "size-1 foreign" atom and check "does this atom live in a non-ring size-1 section?"
- **Step A** scans for any ring–ring borders (atoms that join two different ring sections). Let any such bonded pair or triplet be assigned a bead using Cases B1/B2/B3.
- **Step B** scans for any *inner* double bond within the remaining unmapped ring atoms. Each time it finds a pair of unmapped atoms connected by a double-bond, it either bundles them (plus their size-1 foreign partner, if present) into an S-type or T-type bead, or falls back to `"TC5"/"TN6a"`.
- **Step C** does two passes. It looks for any still-unmapped ring atom that has **two** "size-1, non-ring" neighbors (step C1), assigning either `SC3`, `SN5a`, `SN1`, or `SX4e` if that central atom is carbon, or `SN3a` if that atom is oxygen.
- If a ring atom has **exactly one** size-1 neighbor, we group it (plus a carefully chosen inner neighbor) under either an S-type (if a neighbor is found) or T-type (if no good neighbor) bead.
- **Step C.5** catches any atom that is neither C nor O (e.g. maybe an S, N, or Cl) and has exactly one size-1 foreign neighbor of element C; it assigns them both a T-type bead.
- **Step D** is the "last resort." It counts how many unmapped atoms remain in the ring and handles each possible cardinality (6 → special oxygen-centric assignments, 5 → "promote a carbon" trick, 4 → two-atom T-pairs, 3 → three-atom SC or merge logic, 2 → two-atom T-pair, 1 → merge/promote if carbon, error if oxygen). By the end of Step

D, every ring atom in `section` should be assigned to `final[global_index]` with a nonempty bead string. If at any point no bead can be found, a `ValueError` is raised.

Here we will focus on Step B and C

Issue: double bond is prioritized and therefore messed with the results (here, there should be 6 beads representing all 6 connections

Fix: double bond is still prioritized but not when both atoms have a foreign non-ring section of size 1

The issue is because I check for or before and where I should have done the other way around

However there still is another error

The updated final table is now:

['TX3rHW|VP', 'TX3rHW|VP', 'TX3\vV>h[', 'TX3\vV>h[', 'SX3e5big3', 'SX3e5big3', 'SX3e5big3', '', 'TX3_>H;.a', 'TX3_>H;.a', 'TX3od)*w{', 'TX3od)*w{']

there is only 1 part left I need to fix the SX3 SX3 SX3 blank which should be TX3 TX3 TX3 TX3

We must look at why this is still happening

We are now focusing on the neighbor logic:

- The code scans each inner neighbor `nbr` of the current ring atom.
- It first requires that `nbr` itself is still unmapped.
- Then it checks whether `nbr` has at least one inner neighbor already mapped (`not inner_unmapped`). If *all* of `nbr`'s inner neighbors were unmapped, `nbr` is discarded here.
- Next, it gathers `nf_tups`, the list of `nbr`'s unmapped "size-1 outer" neighbors.
- Finally, it ensures `nbr` is carbon and does not have more than one unmapped size-1 neighbor to juggle. Precisely, it allows `nbr` if:

3. `nbr` has exactly one unmapped foreign (so it can group in a triple), **or**
4. `nbr` has two outers but neither is that unmapped size-1 type, **or**
5. `nbr` has zero unmapped size-1 foreign neighbors.

if nbr[1].upper() == 'C' and not inner_unmapped and not nf_tups: array1.append(nbr)

here is the new rule for considering neighbor, and it works for this case but there appears immediately another case that it cannot work for

elif len(array1) == 2:

if i == 1:

raise ValueError("Non–ring section mapping too complex (foreign/inner connection).")

for this case let's try the following:

if i == 1:

so now we have run through the code twice to the same issue.

Here, we need to check the inner neighbors of each of the candidates to see what their final is.

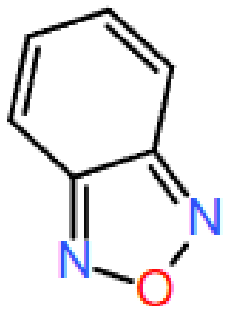
we will prioritize any candidate that is not connected to an inner neighbor that has final[index] that starts with T and use that candidate immediately once we find one.

if both are connected to one that starts with T, just use any. And do bead assignment like the len(array) == 1 case

Now everything works as intended.

39 CASES LEFT

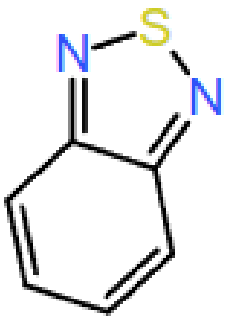
9. 273-09-6



Clear issue as before, we don't have that flexibility for the inner of a bead

Solution: No solution

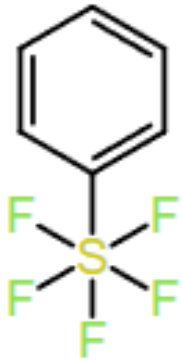
10. 273-13-2



Solution: c1cnsn1: SNa x2, c1ccccc1: SC5 x2 – does not make sense again. Score: 2.75 2.20 - off by 0.55

Same issue but now come to the look of it, this is solvable! We need to have a different approach on the ring however, as the 5 ring here needs to be mapped as C=N and C=N for it to work, same with the other case. These are fixable but will require major changes to the code structure so we will do them at the end

11. 2557-81-5

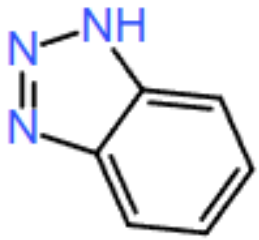


Solution: SC3 for the whole thing which clearly does not make sense. Score 4.60 2.87 off by 1.73

Obvious reason why it does not work

We need to fix the 5 edges bead case which we have not seen before, again not a code issue so we will skip and work on it later

12. 95-14-7



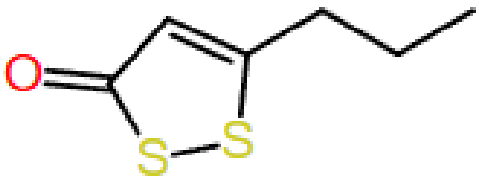
Solution: c1cnnn1: SP1 x2, c1ccccc1: SC5 x2

Observation: same as case 7 – no N=N bead

We will talk about the theory of them thinking of rings as a whole

score: 1.97 1.17 off by 0.8

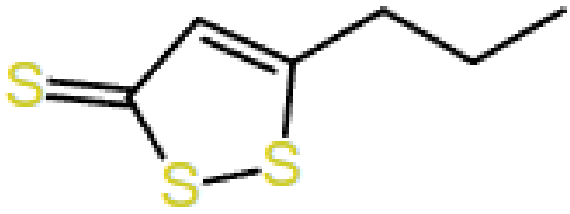
13. 164584-62-7



Solution: Does not converge

Observation: Clear SS blockade

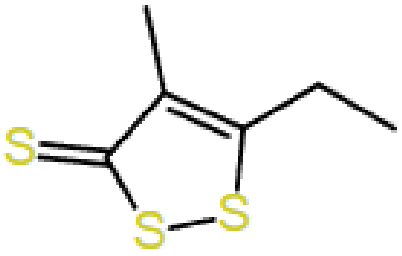
14. 146252-77-9



Solution: Does not converge

Observation: Clear SS blockade

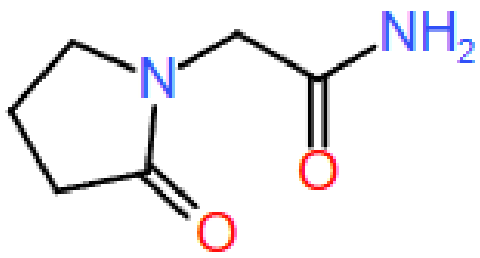
15. 6125-90-2



Solution: Does not converge

Observation: Clear SS blockade

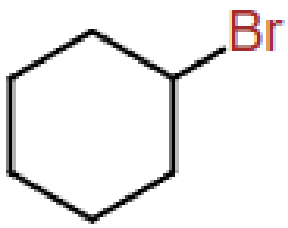
16. 7491-74-9



Solution: P2: CC(N)=O, SP1: C1CCNC1, SP2: O=C1CCCN1

Score: -2.10 -1.78 off by 0.32, for us we have the clear N blockade with no bead to assign

17. 108-85-0



Solution: N0: CBr, 2x SC2: C1CCCCC1

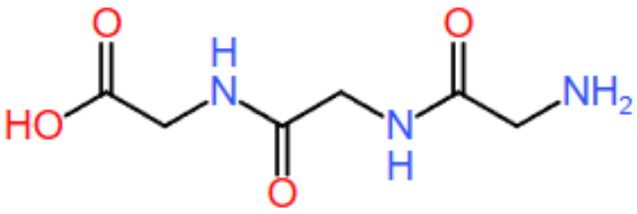
This case is interesting because we have not seen the 5 unmapped atom in a non_6_ring before, but this is fixable:

Fix handling for 5 unmapped atoms

Our solution: SX2: CCBr, TC3: CC x2

36 CASES LEFT

18. 556-33-2



Observation: This formation CCC(C)CCC(C)CCC(C)C causes issues

Solution: P5: NCC=O, P2: CN, P3: CNC=O, P1:OC=O

The solution makes sense; we must investigate why this is causing issues for our own code. I will go to sleep now.