Grafting fragments onto molecules in rdkit - babysteps

Peter Schmidtke

2021-01-23

Table of contents

Context

In this quick walkthrough I describe the first steps to attach fragments from a fragment library onto a molecule of interest. The main idea here is to prepare code snippets for the integration of all of this into the web-based 3d-editor project I started with Daniel Alvarez some time ago. As I'm learning a lot of new things about rdkit I prefer to write it up here, as I found most of the relevant information in the mailing list & the rdkit documentation.

The molecule to modify

I'll go for the same molecule as the one used for now in our BRD4 structure of the 3d editor project - that just comes from one of the official openforcefield examples here.

The fragment to add

I wanted to go for something easy for now … so let's start with a methyl group ;) I made a quick shoutout on twitter on current available 3D fragments that could be helpful to write such a sketcher. Geoff Hutchison (Mr Avogadro) gave me a great hint to this library here. It contains way more fragments than I'd initially considered for this project and the coordinates seem reasonable & compatible with the systems we'll manage in the 3d-editor.

So I just took the methane from there (for now), ran it through Marvinsketch to replace one of the protons with an R group and then exported the whole thing as mol file (to the best of my knowledge rdkit doesn't support the chemistry markup language yet).

Preliminary steps

```
# https://sourceforge.net/p/rdkit/mailman/message/34922663/ procedure extracted from this
import rdkit
from rdkit import Chem
from rdkit.Chem.Draw import IPythonConsole
from rdkit.Chem import Draw
IPythonConsole.ipython_useSVG=True
from rdkit.Chem import AllChem
import copy
from rdkit.Chem.rdMolAlign import AlignMol
```

Below, the molecule we want to add the methyl group to. We'll attach it on the triazole ring.

```
//| code-fold: true
//| echo: false

// Create drawing area
div_mol = html`<div style="width:400px;height:400px;position:relative"></div>`;
```

```
//| code-fold: true
//| output: false
//| echo: false

// Create viewer
  NGL = require("ngl@next");

   molBlock = await FileAttachment("ligand.sdf").blob();
```

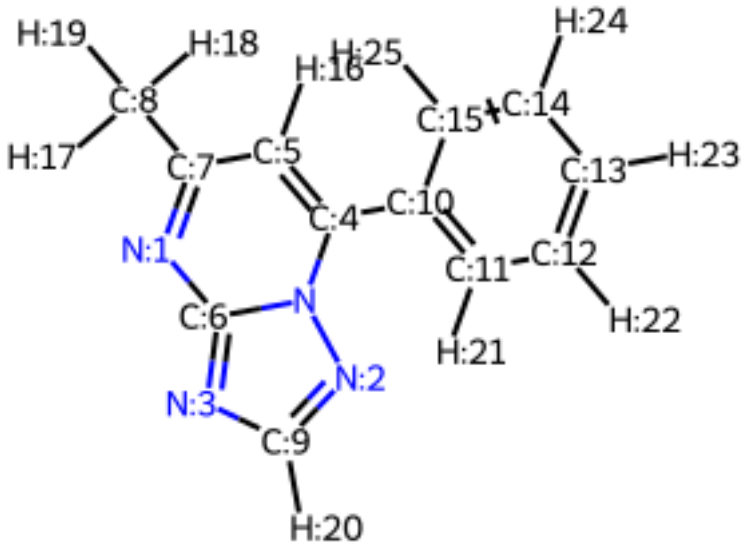
```
stage = new NGL.Stage(div_mol, { backgroundColor: "black",cameraType: "orthographic"});
structure = await stage.loadFile(molBlock, {ext: "sdf", asTrajectory: false})
structure.addRepresentation("licorice",{ "sele": "all", "color": "element"});
structure.autoView();
```

As the aim is to integrate that into the 3D-editor, the user ultimately will be able to click on the proton where he wants to place the fragment. Thus, we know which exact atom we want to attach it to. Below I'm determining this showing plain atom indices in the rdkit molecule

```
suppl = Chem.SDMolSupplier('ligand.sdf',removeHs=False)
for mol in suppl:
    mblock = Chem.MolToMolBlock(mol)

for atom in mol.GetAtoms():
        atom.SetAtomMapNum(atom.GetIdx())
mol2d=copy.deepcopy(mol)
AllChem.Compute2DCoords(mol2d)    #do this on a copy here

Draw.MolToImage(mol2d, includeAtomNumbers=True)
```



Next I'm loading the fragment and extract the position of the R-group and the connected atom as atom indices (I'll need that later). I know this is very limited and ugly for now, but it serves the purpose here & now ;)

```
def getAttachmentVector(mol):
    """ for a fragment to add, search for the position of the attachment point and extract
    mol: fragment passed as rdkit molecule
    return: tuple (atom indices)
    """

    rindex=-1
    rindexNeighbor=-1
    for atom in mol.GetAtoms():
        if(atom.GetAtomicNum()==0):
            rindex=atom.GetIdx()
            neighbours=atom.GetNeighbors()
            if(len(neighbours)==1):
                rindexNeighbor=neighbours[0].GetIdx()
```

```
            else:
                print("two attachment points not supported yet")
                return None
    return((rindex,rindexNeighbor))



fragment=Chem.MolFromMolFile('methyl.mol',removeHs=False)
ret=getAttachmentVector(fragment)

for atom in mol.GetAtoms():
        atom.SetAtomMapNum(atom.GetIdx())
if(ret):
    fragIndex1,fragIndex2=ret

print(fragIndex1, fragIndex2)
Chem.MolToMolFile(fragment,"molblock.mol")
```

```
4 1
```

```
//| code-fold: true
//| echo: false

// Create drawing area
div_mol_2 = html`<div style="width:400px;height:400px;position:relative"></div>`;
```

```
//| code-fold: true
//| echo: false
//| output: false


// Draw new molecule

molBlock2 = await FileAttachment("molblock.mol").blob();
stage2 = new NGL.Stage(div_mol_2, { backgroundColor: "black",cameraType: "orthographic"});
structure2 = await stage2.loadFile(molBlock2, {ext: "sdf", asTrajectory: false})
structure2.addRepresentation("licorice",{ "sele": "all", "color": "element"});
structure2.autoView();
```

Now let's extract also the atom index of the proton, and also the carbon the proton is attached to on the molecule.

```
queryAtomIndex=20 #defined by clicking on an atom in the sketcher -> needs to be a proton

atom=mol.GetAtomWithIdx(queryAtomIndex)
neighbours=atom.GetNeighbors()
if(len(neighbours)==1):
    rindexNeighbor=neighbours[0].GetIdx()
else:
    print("two attachment points not supported yet")

molIndex1=queryAtomIndex
molIndex2=rindexNeighbor
print(molIndex1,molIndex2)
```

```
20 9
```

Aligning the fragment onto the molecule

Now I have the bond of the carbon to proton selected in the molecule and the carbon to R-group in my fragment. These bonds can be aligned onto each other to position the 3D-fragment correctly versus the molecule. This can be conveniently done using the AlignMol function available in rdkit.

```
AlignMol(fragment,mol,atomMap=((fragIndex2,molIndex1),(fragIndex1,molIndex2))) #important
```

```
0.02749767976054038
```

```
#the rest is just to display things here
Chem.MolToMolFile(mol,"molblock3.mol")
Chem.MolToMolFile(fragment,"fragblock3.mol")
```

```
//| code-fold: true
//| echo: false

// Create drawing area
div_mol_3 = html`<div style="width:400px;height:400px;position:relative"></div>`;
```

```
//| code-fold: true
//| echo: false
```

```
//| output: false


// Draw new molecule

molBlock3 = await FileAttachment("molblock3.mol").blob();
fragBlock3 = await FileAttachment("fragblock3.mol").blob();
stage3 = new NGL.Stage(div_mol_3, { backgroundColor: "black",cameraType: "orthographic"});
smol3=await stage3.loadFile(molBlock3, {ext: "sdf", asTrajectory: false});
smol3.addRepresentation("licorice",{ "sele": "all", "color": "element"});
sfrag3=await stage3.loadFile(fragBlock3, {ext: "sdf", asTrajectory: false});
sfrag3.addRepresentation("licorice",{ "sele": "all", "color": "element"});
sfrag3.autoView();
```

As you can see, the fragment gets placed correctly on top of the proton. Now we have however a few overlapping atoms in place. We can use the rdkit edition functions to address this and combine both overlapping molecules now into a final single molecule.

```
def connectMols(mol1, mol2, atom1, atom2):
        """function copied from here https://github.com/molecularsets/moses/blob/master/mo
        combined = Chem.CombineMols(mol1, mol2)
        emol = Chem.EditableMol(combined)
        neighbor1_idx = atom1.GetNeighbors()[0].GetIdx()
        neighbor2_idx = atom2.GetNeighbors()[0].GetIdx()
        atom1_idx = atom1.GetIdx()
        atom2_idx = atom2.GetIdx()
        bond_order = atom2.GetBonds()[0].GetBondType()
        emol.AddBond(neighbor1_idx,
                     neighbor2_idx + mol1.GetNumAtoms(),
                     order=bond_order)
        emol.RemoveAtom(atom2_idx + mol1.GetNumAtoms())
        emol.RemoveAtom(atom1_idx)
        mol = emol.GetMol()
        return mol

finalMol=connectMols(mol,fragment,mol.GetAtomWithIdx(molIndex1),fragment.GetAtomWithIdx(fr
Chem.SanitizeMol(finalMol)
```

```
rdkit.Chem.rdmolops.SanitizeFlags.SANITIZE_NONE
```

```
#the rest is just to display things here
Chem.MolToMolFile(finalMol,"finalmol.mol")
```

```
//| code-fold: true
//| echo: false

// Create drawing area
div_mol_4 = html`<div style="width:400px;height:400px;position:relative"></div>`;
```

```
//| code-fold: true
//| echo: false
//| output: false


// Draw new molecule

molBlock4 = await FileAttachment("finalmol.mol").blob();
stage4 = new NGL.Stage(div_mol_4, { backgroundColor: "black",cameraType: "orthographic"});
finalmol=await stage4.loadFile(molBlock4, {ext: "sdf", asTrajectory: false});
finalmol.addRepresentation("licorice",{ "sele": "all", "color": "element"});
finalmol.autoView();
```

And voilà. Our methyl is nicely placed and oriented. That's one of the easiest cases and we'll have to consider torsion angles & protein environment at a later stage as well, but this should provide first basic steps for simple additions like the one done here.