RDkit

## Installation RDkit

Instructions to install RDkit:

<http://rdkit.org/RDKit_Docs.current.pdf>

But first install conda:

<http://wiki.ab.wurnet.nl/index.php/How_can_I_install_software_for_my_own_use>

Activate environment:

source activate my-rdkit-env

source deactivate

Matplotlib is required for some functions:

conda install matplotlib

Getting started with the RDKit in Python

<http://www.rdkit.org/docs/GettingStartedInPython.html>

## SMARTS and SMILES

The most important differences between SMARTS and SMILES are shown in table 1. For more information, see: <http://www.daylight.com/dayhtml/doc/theory/theory.smarts.html>

Smart viewer: <http://smartsview.zbh.uni-hamburg.de>

Table 1. Differences between SMARTS and SMILES

|  |  |  |
| --- | --- | --- |
|  | SMARTS | SMILES |
| fragments $(…) | allowed | Not allowed |
| C | aliphatic | Aromatic or aliphatic |
| c | aromatic | Aromatic or aliphatic |
| cc (empty bond) | single or aromatic | aromatic if it belongs to a ring and has both end atoms aromatic (lower case), otherwise a single bond |
|  | c1=cc=cc=c1  RDkit recognizes this structure but officially it is not correct:  C1=CC=CC=C1 | c1ccccc1  C1=CC=CC=C1  c1-c=c-c=c-c=1 |
|  | COc | COC |

## Reading molecules

Smiles C1=CC=CN=C1, c1cccnc1 and n1ccccc1 represent the aromatic molecule below:

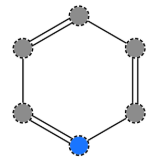


Figure . Molecule

>>> from \_\_future\_\_ import print\_function

>>> from rdkit import Chem

>>> from rdkit.Chem import AllChem

Conversion needed for RDkit:

>>> m = Chem.MolFromSmiles('C1=CC=CN=C1')

Also SMART patterns can be used:

>>> m = Chem.MolFromSmarts('c1=cc=cn=c1')

Return a SMILE:

>>> Chem.MolToSmiles(m)

c1ccncc1

The kekule form of SMILE:

>>> Chem.Kekulize(m)

>>> Chem.MolToSmiles(m, kekuleSmiles=True)

C1=CC=NC=C1

Add and remove hydrogens

>>> m2 = Chem.AddHs(m)

>>> Chem.MolToSmiles(m2)

[H]c1c([H])c([H])c([H])c([H])n1

>>> m = Chem.RemoveHs(m2)

>>> Chem.MolToSmiles(m)

c1ccncc1

Calculating mol weight

>>> from rdkit.Chem import Descriptors

>>> Descriptors.ExactMolWt(m)

79.04219916

Creating a molfile:

Add compound name:

>>> m.SetProp("\_Name","Aromatic molecule")

With 2D coordinates

>>> AllChem.Compute2DCoords(m)

With 3D coordinates

>>> AllChem.EmbedMolecule(m, useExpTorsionAnglePrefs=True, useBasicKnowledge=True)

>>> Chem.MolToMolBlock(m)

Aromatic molecule

RDKit 3D

6 6 0 0 0 0 0 0 0 0999 V2000

-1.1028 0.8475 0.0035 C 0 0 0 0 0 0 0 0 0 0 0 0

0.1720 1.3675 -0.0393 C 0 0 0 0 0 0 0 0 0 0 0 0

1.2453 0.5069 -0.0417 C 0 0 0 0 0 0 0 0 0 0 0 0

1.0802 -0.8571 -0.0027 C 0 0 0 0 0 0 0 0 0 0 0 0

-0.1671 -1.3423 0.0385 N 0 0 0 0 0 0 0 0 0 0 0 0

-1.2276 -0.5224 0.0417 C 0 0 0 0 0 0 0 0 0 0 0 0

1 2 2 0

2 3 1 0

3 4 2 0

4 5 1 0

5 6 2 0

6 1 1 0

M END

looping over atoms and bonds (in the same order as the entered SMILE)

>>> for atom in m.GetAtoms():

atom.GetAtomicNum()

6

6

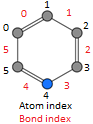
6

6

7

6

Get bound type, three different ways:

>>> m.GetBonds()[0].GetBondType()

DOUBLE

>>> m.GetBondBetweenAtoms(0,1).GetBondType()

DOUBLE

>>> m.GetBondWithIdx(0).GetBondType()

DOUBLE

Get symbol:

>>> m.GetAtomWithIdx(0).GetSymbol()

'C'

Figure . molecule with indexes

Get explicit valence:

>>> m.GetAtomWithIdx(0).GetExplicitValence()

3

Get begin atom index:

>>> m.GetBondWithIdx(0).GetBeginAtomIdx()

0

Get end atom index:

>>> m.GetBondWithIdx(0).GetEndAtomIdx()

1

Highlighted = atom or bond index, see fig 2.

## Substructure matching

New molecule:

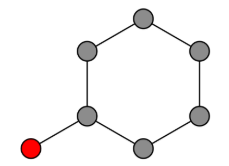
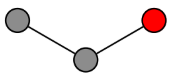


Figure : Molecule



>>> m = Chem.MolFromSmiles('c1ccccc1O')

>>> patt = Chem.MolFromSmarts('ccO') # see fig. 4

Figure : patt/ substructure

Boolean; if substructure matches:

>>> m.HasSubstructMatch(patt)

True

Atoms in structure which match:

>>> m.GetSubstructMatch(patt)

(0, 5, 6)

If the substructure matched multiple times:

>>> m.GetSubstructMatches(patt)

((0, 5, 6), (4, 5, 6))

## Chemical transformations

Deleting a substructure:

>>> rm = AllChem.DeleteSubstructs(m, patt)

>>> Chem.MolToSmiles(rm) #resulting atom not shown

New molecule:

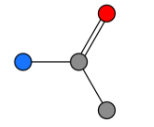
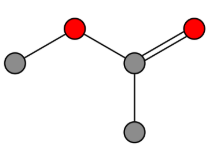


Figure : Start molecule

Figure : patt/ substructure

Figure : replacement

Figure : End molecule

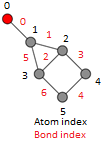
Replacing a substructure:

>>> m2 = Chem.MolFromSmiles('CC(=O)N') # fig. 5

>>> patt = Chem.MolFromSmarts('[$(NC(=O))]') # fig. 6

>>> repl = Chem.MolFromSmiles('OC') # fig. 7

>>> rms = AllChem.ReplaceSubstructs(m7,patt2,repl) # fig. 8



## Ring information

New molecule, see fig 9:

>>> m = Chem.MolFromSmiles('OC1C2C1CC2')

Boolean if atom is in ring:

>>> m.GetAtomWithIdx(0).IsInRing() # atom index

False

Figure : New molecule with atom and bond indices

Boolean if atom is in ring with ringsize (bonds determine the size):

>>> m4.GetAtomWithIdx(2).IsInRingSize(4))

True

More detail about the smallest set of smallest rings (SSSR) is available

>>> ssr = Chem.GetSymmSSSR(m)

Number of rings in molecule:

>>> len(ssr)

2

>>> Chem.GetSSSR(m)

2

Atom indexes:

>>> list(ssr[0])

[1, 2, 3]

>>> list(ssr[1])

[4, 5, 2, 3]

More ring information:

>>> ri = m.GetRingInfo()

Number of rings the atom is involved in:

>>> ri.NumAtomRings(2)

2

Boolean; is atom in ring of size?

>>> ri.IsAtomInRingOfSize(1,4) # 1 = atom index, 4 = atoms in ring

False

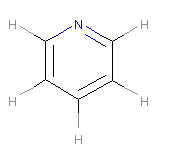
Boolean; is bond in ring of size?

>>> ri.IsBondInRingOfSize(3,4)

True

## Draw structures

>>> from rdkit.Chem import Draw

>>> m1 = Chem.MolFromSmiles('C1=CC=CN=C1')

>>> m2 = Chem.AddHs(m1)

>>> m3 = Chem.MolFromSmiles('c1ccccc1O')

>>> m\_list = m1, m2, m3

2D; create a png file with molecule in it:

>>> Draw.MolToFile(m1,'/path/to/molecule.png') # fig. 10

Figure : molecule.png

2D; create a png file with multiple molecules in it:

>>> multiple\_molecules = Draw.MolsToGridImage(m\_list, molsPerRow=2, subImgSize=(100,100))

>>> multiple\_molecules.save('/path/to/multiple\_molecules.png') # result not shown

When making an image with multiple similar molecules, you can align them by structure:

>>> p = Chem.MolFromSmiles('the overlapping substructure’)

>>> subms = [x for x in ms if x.HasSubstructMatch(p)] # ms = list with molecules

>>> for m in subms: AllChem.GenerateDepictionMatching2DStructure(m,p)

>>> img=Draw.MolsToGridImage(subms,molsPerRow=4,subImgSize=(200,200),legends=[x.GetProp

("\_Name") for x in subms])

>>> img.save('/home/stokm006/thesis/images/molecule\_example.png') # result not shown

## Fingerprinting and Molecular Similarity

Fingerprinting similarity scoring: identifies and hashes topological paths (e.g. along bonds) in the molecule and then uses them to set bits in a fingerprint of user-specified lengths

Default; Tanimoto similarity

>>> from rdkit import DataStructs

>>> from rdkit.Chem.Fingerprints import FingerprintMols

>>> from rdkit.Chem import Descriptors

>>> from rdkit.Chem import MACCSkeys

>>> m\_list2 = [Chem.MolFromSmiles('CCOC'), Chem.MolFromSmiles('CCO'), Chem.MolFromSmiles('COC')]

>>> fps = [FingerprintMols.FingerprintMol(x) for x in m\_list2]

>>> DataStructs.FingerprintSimilarity(fps[0],fps[1]))

0.6

>>> DataStructs.FingerprintSimilarity(fps[0],fps[0]))

1.0

>>> DataStructs.FingerprintSimilarity(fps[2],fps[1]))

0.25

MACCS keys

>>> fps = [MACCSkeys.GenMACCSKeys(x) for x in m\_list2]

>>> DataStructs.FingerprintSimilarity(fps[0],fps[1])

0.5

>>> DataStructs.FingerprintSimilarity(fps[0],fps[0])

1.0

>>> DataStructs.FingerprintSimilarity(fps[2],fps[1])

0.21428571428571427

Atom pair fingerprints, atom descriptor

>>> ms = [Chem.MolFromSmiles('C1CCC1OCC'),Chem.MolFromSmiles('CC(C)OCC'),

Chem.MolFromSmiles('CCOCC')]

>>> pairFps = [Pairs.GetAtomPairFingerprint(x) for x in ms]

Get the list of bits and their counts for each fingerprint as a dictionary:

>>> d = pairFps[-1].GetNonzeroElements()

>>> d

{558113: 2, 558146: 1, 558115: 2, 541732: 1, 1606721: 2, 1606690: 2}

Explanation of the bitscore.

>>> Pairs.ExplainPairScore(558115)

(('C', 1, 0), 3, ('C', 2, 0))

Meaning:C with 1 neighbor and 0 pi electrons which is 3 bonds from a C with 2 neighbors and 0 pi electrons

Dice similarity; The usual metric for similarity between atom-pair fingerprints

>>> DataStructs.DiceSimilarity(pairFps[0],pairFps[1])

0.3333333333333333

Atom decriptor without count

>>> pairFps = [Pairs.GetAtomPairFingerprintAsBitVect(x) for x in ms]

>>> DataStructs.DiceSimilarity(pairFps[0],pairFps[1])

0.48

Morgan/ circular fingerprints

The default atom invariants use connectivity information similar to those used for the well known ECFP family of fingerprints

>>> m1 = Chem.MolFromSmiles('CCOC')

>>> fp1 = AllChem.GetMorganFingerprint(m1, 2)

>>> m2 = Chem.MolFromSmiles('CCO')

>>> fp2 = AllChem.GetMorganFingerprint(m2, 2)

>>> DataStructs.DiceSimilarity(fp1,fp2))

0.4

To calculate them as bit vector (without counts)

>>> fp1 = AllChem.GetMorganFingerprintAsBitVect(m1,2,nBits=1024)

>>> fp2 = AllChem.GetMorganFingerprintAsBitVect(m2,2,nBits=1024)

>>> DataStructs.DiceSimilarity(fp1,fp2)

0.42857142857142855

Feature-based invariants, similar to those used for the FCFP fingerprints, can also be used:

>>> ffp1 = AllChem.GetMorganFingerprint(m1,2,useFeatures=True)

>>> ffp2 = AllChem.GetMorganFingerprint(m2,2,useFeatures=True)

>>> DataStructs.DiceSimilarity(ffp1,ffp2)

0.4

Bits from Morgan Fingerprints

>>> m9 = Chem.MolFromSmiles('c1cccnc1C')

>>> info={}

>>> AllChem.GetMorganFingerprint(m9,2,bitInfo=info)

>>> info

{98513984: ((1, 1), (2, 1)), 2246728737: ((6, 0),), 1207774339: ((2, 2),), 3217380708: ((5, 0),), 4048591891: ((5, 2),), 3776905034: ((3, 1),), 1100037548: ((4, 1),), 3999906991: ((1, 2),), 3218693969: ((0, 0), (1, 0), (2, 0), (3, 0)), 1235524787: ((0, 2),), 2614860224: ((3, 2),), 951226070: ((0, 1),), 1751362425: ((4, 2),), 422715066: ((6, 1),), 4036277955: ((5, 1),), 2041434490: ((4, 0),)}

>>> info[98513984]

((1, 1), (2, 1))

Meaning: bit 98513984 is set twice: once by atom 1 and once by atom 2, each at radius 1.

>>> info[4048591891]

((5, 2),)

Meaning: Bit 4048591891 is set once by atom 5 at radius 2.

Extract a substructure with a x radius and x atom from a bit.

Example; bit 4048591891, with radius 2 and atom 5.

>>> env = Chem.FindAtomEnvironmentOfRadiusN(m,2,5)

>>> amap={}

>>> submol=Chem.PathToSubmol(m,env,atomMap=amap)

>>> submol.GetNumAtoms()

6

>>> amap

{0: 3, 1: 5, 3: 4, 4: 0, 5: 1, 6: 2}

Convert bit to SMILE:

>>> Chem.MolToSmiles(submol)

ccc(C)nc

Root the SMILE at the central atom:

>>> Chem.MolToSmiles(submol,rootedAtAtom=amap[5],canonical=False)

c(nc)(C)cc

Descriptor Calculation; used in papers or coding languages

See also: <http://www.rdkit.org/docs/GettingStartedInPython.html#list-of-available-descriptors>

>>> m = Chem.MolFromSmiles('c1ccccc1O')

Descriptor TPSA

>>> Descriptors.TPSA(m)

20.23

Descriptor MolLogP

>>> Descriptors.MolLogP(m)

1.03922

## Chemical reactions

Amide bond formation; first create ‘template’ rxn

>>> rxn = AllChem.ReactionFromSmarts('[C:1](=[O:2])-[OD1].[N!H0:3]>>[C:1](=[O:2])[N:3]')

>>> ps = rxn.RunReactants((Chem.MolFromSmiles('CC(=O)O'),Chem.MolFromSmiles('NC')))

>>> Chem.MolToSmiles(ps[0][0]))

CNC(C)=O

For more complex reactions:

>>> rxn = AllChem.ReactionFromSmarts('[C:1]=[C:2].[C:3]=[\*:4][\*:5]=[C:6]>>[C:1]1[C:2][C:3]

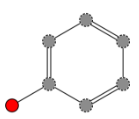
[\*:4]=[\*:5][C:6]1')

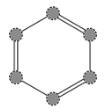
>>> ps = rxn.RunReactants((Chem.MolFromSmiles('OC=C'), Chem.MolFromSmiles('C=CC(N)=C')))

>>> Chem.MolToSmiles(ps[0][0])

NC1=CCCC(O)C1

## Maximum common substructure

The FindMCS function finds a maximum common substructure (MCS) of two or more molecules.

>>> from rdkit.Chem import rdFMCS

>>> m1 = Chem.MolFromSmiles("c1ccccc1O") # fig. 11

>>> m2 = Chem.MolFromSmiles("c1ccccc1") # fig. 12

>>> m3 = Chem.MolFromSmiles("c1ccccc1N") # fig. 13

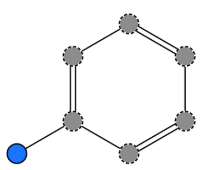
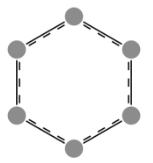
>>> m\_list = [mol1,mol2,mol3]

>>> res=rdFMCS.FindMCS(m\_list)

>>> res.numAtoms

Figure : m2

Figure : m1

6

>>> res.numBonds

6

>>> res.smartsString # fig. 14

Figure : [#6]1:[#6]:[#6]:[#6]:[#6]:[#6]:1

[#6]1:[#6]:[#6]:[#6]:[#6]:[#6]:1

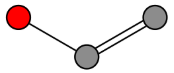
Figure : m3

## Compare atom and bond difference between two molecules.

See table 2 for the rdFMCS.findMCS argument options

Two new molecules:

>>> m\_list2 = (Chem.MolFromSmiles('NCC'),Chem.MolFromSmiles('OC=C')) # fig. 15 and 16



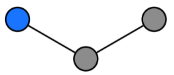


Figure : NCC

Figure : OC=C

Compare the atom difference:

>>> rdFMCS.FindMCS(m\_ list2, atomCompare=rdFMCS.AtomCompare.CompareAny).smartsString

[#7,#8]-[#6] # fig. 17

Compare the bond difference:

>>> rdFMCS.FindMCS(m\_ list2, bondCompare=rdFMCS.BondCompare.CompareAny).smartsString

[#6]-,=[#6] # fig. 18



Figure : Figure 18: [#7,#8]-[#6]; N or O

Figure : [#6]-,=[#6] single or double bond

Table 2: Argument options

|  |  |
| --- | --- |
| Argument | Function |
| AtomCompare.CompareAny | matches any atom to any other atom |
| CompareElements | compares by element type |
| CompareIsotopes | matches based on the isotope label |
| BondCompare.CompareAny | matches any bond to any other bond |
| BondCompare.CompareOrderExact | bonds are equivalent if and only if they have the same bond type |
| BondCompare.CompareOrder | Match single and aromatic bonds to other, but requires an exact order match otherwise |
| ringMatchesRingOnly=True | Ring bond will only match ring bonds. (the default setting allows ring to linear carbon chain matching) |
| completeRingsOnly=True | Partial rings are not allowed. (the default setting does allow partial rings) |
| timeout | Provides a time limit for big searches |

## Generating Similarity Maps Using Fingerprints

The SimilarityMaps module supports three kind of fingerprints: atom pairs, topological torsions and Morgan fingerprints.

>>> from rdkit.Chem.Draw import SimilarityMaps

>>> m1 = Chem.MolFromSmiles("c1ccccc1O")

>>> m2 = Chem.MolFromSmiles("c1ccccc1N")

The function GetSimilarityMapForFingerprint involves the normalisation of the atomic weights such that the maximum absolute weight is 1

>>> fig, maxweight = SimilarityMaps.GetSimilarityMapForFingerprint(m1, m2, SimilarityMaps.GetMorganFingerprint) # or GetAPFingerprint or GetTTFingerprint

>>> fig.savefig('/path/to/mol\_similaritymap.png', bbox\_inches='tight') # fig. 19

>>> maxweight

0.4121212121212121

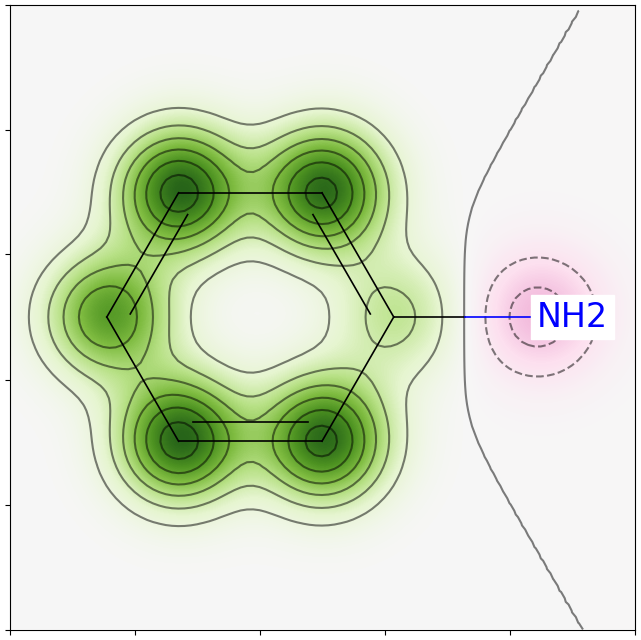


Figure : GetSimilarityMapForFingerprint with normalisation

You can also create a similarity map without normalisation step:

>>> weights = SimilarityMaps.GetAtomicWeightsForFingerprint(m1, m2, SimilarityMaps.GetMorganFingerprint)

>>> fig = SimilarityMaps.GetSimilarityMapFromWeights(m2, weights, size=(150, 150))

>>> fig.savefig('/path/to/mol\_similaritymap.png', bbox\_inches='tight') # fig. 20

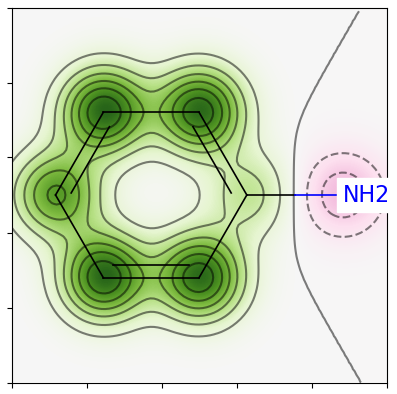


Figure : GetSimilarityMapForFingerprint without normalisation

You can also visualize the degree of partial charge by using atomic charge:

>>> AllChem.ComputeGasteigerCharges(m1)

>>> charges = [float(atom.GetProp('\_GasteigerCharge')) for atom in m1.GetAtoms()]

>>> fig = SimilarityMaps.GetSimilarityMapFromWeights(m2,charges, size=(150, 150),scale=10)

>>> fig.savefig('/path/to/molcharge\_similaritymap.png', bbox\_inches='tight') # fig. 21

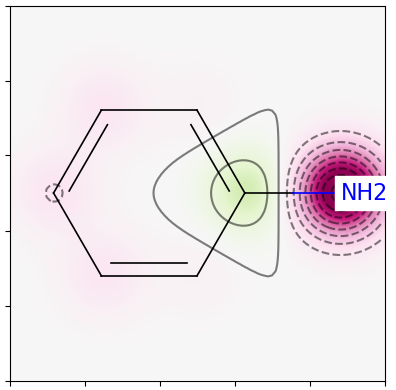


Figure degree of partial charge:

## Chemical features

Identify chemical features in molecules using a SMARTS-based feature definition language.

>>> import os

>>> from rdkit.Chem import ChemicalFeatures

>>> fdefName = os.path.join(RDConfig.RDDataDir,'BaseFeatures.fdef')

>>> factory = ChemicalFeatures.BuildFeatureFactory(fdefName)

>>> m = Chem.MolFromSmiles('OCc1ccccc1CN')

>>> feats = factory.GetFeaturesForMol(m)

>>> len(feats)

8

Individual feature information

>>> feats[0].GetFamily()

'Donor'

>>> feats[0].GetType()

'SingleAtomDonor'

>>> feats[0].GetAtomIds()

(0,)

If the molecules has coordinates

>>> from rdkit.Chem import AllChem

>>> AllChem.Compute2DCoords(m)

>>> list(feats[0].GetPos())

[2.0705367611607857, -2.3356749604090465, 0.0]

## Molecular Fragments

Fragmenting molecules and working with those fragments. Fragments are defined to be made up of a set of connected atoms that may have associated functional groups.

Store the fragments as entries in a FragCatalog:

>>> from rdkit.Chem import FragmentCatalog

>>> fName=os.path.join(RDConfig.RDDataDir,'FunctionalGroups.txt')

>>> fparams = FragmentCatalog.FragCatParams(1,6,fName)

>>> fparams.GetNumFuncGroups()

39

>>> fcat=FragmentCatalog.FragCatalog(fparams)

>>> fcgen=FragmentCatalog.FragCatGenerator()

Find the fragments for a given molecule:

>>> m = Chem.MolFromSmiles('OCC=CC(=O)O')

>>> fcgen.AddFragsFromMol(m,fcat)

3

>>> fcat.GetEntryDescription(0)

C<-O>C

>>> fcat.GetEntryDescription(1)

C=C<-C(=O)O>

>>> fcat.GetEntryDescription(2)

C<-C(=O)O>=CC<-O>

>>> list(fcat.GetEntryDownIds(0))

[2]

The pieces between < and > describe functional groups attached to the fragment. For example, in the above example, the catalog entry 0 corresponds to an ethyl fragment with an alcohol attached to one of the carbons and entry 1 is an ethylene with a carboxylic acid on one carbon.

Detailed information about the functional groups can be obtained:

>>> list(fcat.GetEntryFuncGroupIds(2)) # can be 0, 1 or 2

[34, 1]

>>> fparams.GetFuncGroup(1)

>>> Chem.MolToSmarts(fparams.GetFuncGroup(1))

\*-C(=O)-,:[O&D1]

>>> Chem.MolToSmarts(fparams.GetFuncGroup(34))

\*-[O&D1]

>>> fparams.GetFuncGroup(1).GetProp('\_Name')

-C(=O)O

>>> fparams.GetFuncGroup(34).GetProp('\_Name')

-O

The fragments from multiple molecules can be added to a catalog

>>> ms = ['COC1=C(C=CC(=C1)C=CC(=O)O)O', 'C1=CC2=C(C=C1O)C(=CN2)CCN', 'CC1=CC(=O)OC2=C1C=CC(=C2)O']

>>> fcat=FragmentCatalog.FragCatalog(fparams)

>>> m\_list= []

>>> for m in ms:

m = Chem.MolFromSmiles(m)

m\_list += [m]

>>> for mol in m\_list:

nAdded=fcgen.AddFragsFromMol(mol,fcat)

Number of entries in catalog

>>> fcat.GetNumEntries()

331

>>> fcat.GetEntryDescription(0)

c<-OMe>c<-O>

Now the catalog can be used to fingerprint molecules

>>> fpgen = FragmentCatalog.FragFPGenerator()

>>> fp = fpgen.GetFPForMol(m\_list[0],fcat)

>>> fp.GetNumOnBits()

63

To find the fragments that two molecules have in common by taking the intersection of their fingerprints:

>>> fp2 = fpgen.GetFPForMol(m\_list[1],fcat)

>>> fpandfp2 = fp&fp2

>>> obl = list(andfp.GetOnBits()

>>> obl

[1, 2, 3, 8, 9, 10, 16, 17, 18, 20, 28, 29, 31, 46]

>>> fcat.GetEntryDescription(obl[-1])

c<-O>ccc(c)C

>>> fcat.GetEntryDescription(obl[-5])

cc(c)C

To find the fragments that distinguish one molecule from another:

>>> combinedFp=fp&(fp^fp2) # can be more efficent than fp&(!fp2)

>>> obl = list(combinedFp.GetOnBits())

>>> fcat.GetEntryDescription(obl[-1])

C<-C(=O)O>=Cc(cc)cc<-OMe>

## Notes

* The RDKit pickle format is fairly compact and it is much faster to build a molecule from a pickle than from a Mol file or SMILES string, so storing molecules you will be working with repeatedly as pickles can be a good idea.
* For multiple smiles, just add them in a list: smiles\_list = ['COC1=C(C=CC(=C1)C=CC(=O)O)O', 'C1=CC2=C(C=C1O)C(=CN2)CCN', 'CC1=CC(=O)OC2=C1C=CC(=C2)O']
* Other useful RDkit functions:
  + 2D Pharmacophore Fingerprints
  + Remove or replace cores
  + Attach indices to the atoms in the SMARTS pattern
  + Calculating Topological torsion descriptors
  + Protecting Atoms
  + Picking Diverse Molecules Using Fingerprints
  + Murcko Decomposition