Python and Chemical Informatics

The Daylight and OpenEye toolkits, part II

Presented by Andrew Dalke, Dalke Scientific Software
for David Wild's I590 course at Indiana University
Mar. 1, 2005

Daylight's domain

Daylight provides chemical informatics database servers.
Originally Thor/Merlin, now an Oracle data cartridge.

The servers need to be chemistry aware.
Structures, substructures, reactions, fingerprints.
Developed as a set of libraries; sell the libraries too.

Their audience is chemist/programmers who will use their tools to do reseach and build user applications.

OpenEye

Another chemical informatics company located in Santa Fe.
(There are 6 of us here. I'm tied for smallest.)

Focus on chemistry for molecular modeling NOT databases.
Still need to be chemistry aware

Developed the OEChem library
Highly influenced by the Daylight model of building toolkits.
Used for their products and by chemist/programmers
C++ instead of C
Distributed with Python and (soon) Java interfaces

“Chemistry agnostic”

A lot of chemistry software uses the valance bond model.
But molecules aren’t simply graphs of atoms and bonds.
Consider aromaticity and chirality.

Daylight, MDL and Tripos have different chemical models
Can even be different that what a chemist expects
(eg, aromatic nitrogens in Daylight)

OEChem provides a graph model which can support all of the other chemistry models, but does not force one on you.

It also provides functions to help convert between styles.

OpenEye's domain

(Currently; they keep adding more)

Chemical graph model

read and write many different file formats:

line notations, nomenclature, 2D and 3D

convert between different chemistry models

substructure searching, reactions, MCS

3D structure

conformation enumeration, docking, shapes

electrostatics

force-field evaluation

... many of the tools you need for modeling

Parsing a SMILES string

"oechem" is a submodule of "openeye"
This loads all of the openeye variable and function names into the current module.

Create an empty molecule

```
>>> from openeye.oechem import *
>>> mol = OEMol()
>>> OEParseSmiles(mol, "c1ccccc1O")
1
>>>
```

Parse the SMILES string and put the results into the OEMol.
This is different from the Daylight model.

The Molecule class

A Molecule instance has atoms, bonds, and coordinates.
(but no cycles!)

Need to call a method to get the atoms

Atoms returned as a "generator"

Convert it to a list

```
>>> mol.GetAtoms()
<generator object at 0x46be40>
>>> list(mol.GetAtoms())
[<C OEAtomBase instance at _01857dc0_p_OEChem__OEAtomBase>, <C OEAtomBase instance at
_01857d80_p_OEChem__OEAtomBase>, <C OEAtomBase instance at _01857d40_p_OEChem__OEAtomBase>,
<C OEAtomBase instance at _01857d00_p_OEChem__OEAtomBase>, <C OEAtomBase instance at
_01857cc0_p_OEChem__OEAtomBase>, <C OEAtomBase instance at _01857c80_p_OEChem__OEAtomBase>,
<C OEAtomBase instance at _01857c40_p_OEChem__OEAtomBase>]
>>> for atom in mol.GetAtoms():
...   print atom.GetAtomicNum(),
...
6 6 6 6 6 6 8
>>>
```

A 'for' loop can iterate through the generator's contents

Need a method call here too

Generators? Methods?

Many factors go into developing an API -- performance, usability, readability, cross-platform support, cross-language support, similarity to other libraries, ...

PyDaylight is “pythonic” - designed to feel like a native Python library - and be easy to use

OEChem optimizes for performance and a consistent API across C++, Python and Java.

Working with bonds

GetBonds() returns a generator over the bonds

bond order

```
>>> mol.GetBonds()
<generator object at 0x47f878>
>>> for bond in mol.GetBonds():
...   print bond.GetBgn().GetAtomicNum(), bond.GetOrder(),
...   print bond.GetEnd().GetAtomicNum()
...
6 2 6
6 1 6
6 2 6
6 1 6
6 2 6
6 1 6
6 1 8
>>> for atom in mol.GetAtoms():
...   print len(list(atom.GetBonds())),
...
2 2 2 2 2 3 1
>>>
```

Get the atoms at the end of the bond using GetBgn() and GetEnd()

Can also get the bonds for a given atom

More atomic properties

```
>>> for atom in mol.GetAtoms():
...   print OEGetAtomicSymbol(atom.GetAtomicNum()),
...   print len(list(atom.GetBonds())),
...   print atom.GetImplicitHCount(), atom.IsAromatic()
...
C 2 1 1
C 2 1 1
C 2 1 1
C 2 1 1
C 2 1 1
C 3 0 1
O 1 1 0
>>>
```

Compare to the PyDaylight version

```
>>> for atom in mol.atoms:
...   print atom.symbol, len(atom.bonds), atom.imp_hcount,
...   print atom.aromatic
```

Cycles

How many cycles does cubane have?

While there are cycles:

find a cycle

remove a bond from the cycle

You'll remove 5 bonds -> 5 cycles

Which bonds are in a cycle? No unique solution!

The answer depends on your model of chemistry.

OEChem doesn't attempt to solve it.

Read "**Smallest Set of Smallest Rings (SSSR) considered Harmful**"

http://www.eyesopen.com/docs/html/cplusprog/node127.html

Generating a SMILES

Because the chemistry model is not tied to the molecule, SMILES generation is not a method - it's a function

```
>>> mol = OEMol()
>>> OEParseSmiles(mol, "c1ccccc1O")
1
>>> OECreateCanSmiString(mol)
'c1ccc(cc1)O'
>>> OEParseSmiles(mol, "[238U+]")
1
>>> OECreateCanSmiString(mol)
'c1ccc(cc1)O.[U+]'
>>> OECreateIsoSmiString(mol)
'c1c(cccc1)O.[238U+]'
>>>
```

OEParseSmiles adds to an existing OEMol

Use a different function to make the isomeric SMILES

cansmiles version 1

Convert all SMILES from a file into canonical form

```
from openeye.oechem import *
for line in open("/usr/local/daylight/v481/data/drugs.smi"):
  smiles = line.split()[0]

  mol = OEMol()

  if not OEParseSmiles(mol, smiles):
      raise Exception("Cannot parse %s" % (smiles,))

  print OECreateCanSmiString(mol)
```

Creates a new OEMol for each SMILES

Raise an exception for invalid SMILES
(returns 1 for valid, 0 for invalid)

Print the canonical SMILES

cansmiles version 2

Reuse the same OEMol

```
from openeye.oechem import *

mol = OEMol()

for line in open("/usr/local/daylight/v481/data/drugs.smi"):
  smiles = line.split()[0]
  if not OEParseSmiles(mol, smiles):
      raise Exception("Cannot parse %s" % (smiles,))

  print OECreateCanSmiString(mol)
  mol.Clear()
```

Create only one OEMol

Remove all the atom and bond data from the molecule

File I/O

OEChem supports many different chemical formats

```
>>> ifs = oemolistream()
>>> ifs.open("drugs.smi")
1
>>> ifs.GetFormat()
1
>>> OEFormat_SMI, OEFormat_SDF, OEFormat_MOL2
(1, 9, 4)
>>> for mol in ifs.GetOEMols():
...   print OECreateCanSmiString(mol)
...
c1ccc2c(c1)C34CCN5C3CC6C7C4N2C(=O)CC7OCC=C6C5
CN1C2CCC1C(C(C2)OC(=O)c3ccccc3)C(=O)OC
COc1ccc2c(c1)c(ccn2)C(C3CC4CCN3CC4C=C)O
CN1CC(C=C2C1CC3=CCNc4c3c2ccc4)C(=O)O
CCN(CC)C(=O)C1CN(C2Cc3c[nH]c4c3c(ccc4)C2=C1)C
CN1CCC23c4c5ccc(c4OC2C(C=CC3C1C5)O)O
CC(=O)Oc1ccc2c3c1OC4C35CCN(C(C2)C5C=CC4OC(=O)C)C
CN1CCCC1c2cccnc2
Cn1cnc2c1c(=O)n(c(=O)n2C)C
CC1=C(C(CC1)(C)C)C=CC(=CC=CC(=CCO)C)C
```

Create an input stream

Open the named file. Use the extension to guess the format

Iterate over the OEMols in the input stream

cansmiles version 3

```
from openeye.oechem import *

ifs = oemolistream()
ifs.open("/usr/local/daylight/v481/data/drugs.smi")

for mol in ifs.GetOEMols():
  print OECreateCanSmiString(mol)
```

File conversion

```
from openeye.oechem import *

ifs = oemolistream()
ifs.open("/usr/local/daylight/v481/data/drugs.smi")

ofs = oemolostream()
ofs.open("drugs.sdf")

for mol in ifs.GetOEMols():
  OEWriteMolecule(ofs, mol)

ofs.close()
ifs.close()
```

Open the input stream

Open the output stream

By default the ".sdf" extension selects SDF output

Write the molecule to the given stream in the appropriate format

Optional but a good idea

SD Files

SD files (a.k.a. “sdf”, “MDL” or “CT” files) are often used to exchange chemical data.

Well-defined file format (available from mdli.com)
Stores coordinate data (either 2D or 3D, not both)
Format started in the 1970s (I think)
One section allows arbitrary key/value data

Example SD file

```
OXAZOLE
  MOE1998

  8  8  0  0  0  0  0  0  0  0  1 V2000
   -0.1230   -1.0520    0.2790 C   0  0
   -0.2220   -2.1180    0.4340 H   0  0
    0.8190   -0.3850   -0.4660 C   0  0
    1.6680   -0.6730   -1.0700 H   0  0
    0.5590    0.9450   -0.3780 O   0  0
   -0.5390    1.0060    0.4270 C   0  0
   -0.9280    1.9930    0.6380 H   0  0
   -0.9920   -0.1560    0.8500 N   0  0
  1  2  1
  1  3  2
  1  8  1
  3  4  1
  3  5  1
  5  6  1
  6  7  1
  6  8  2
M  END
>  <P1>
0.12

>  <$SMI>
c1cocn1

$$$$
```

"CT" (connection table) section

Tag named "P1" with value "0.12"

Tag named "$SMI" with value "c1cocn1"

OEMol vs. OEGraphMol

OEChem has several different types of molecule classes. They implement the same basic interface and can often be used interchangeably.

OpenEye distinguishes between a multiple conformer molecule type (like OEMol) and a single conformer type (including OEGraphMol).

Details at http://www.eyesopen.com/docs/html/cplusprog/node104.html

Only OEGraphMol can contain SD tag data - why?

Accessing Tags/Values

```
>>> mol = OEGraphMol()
>>> ifs = oemolistream()
>>> ifs.open("oxazole.sdf")
1
>>> OEReadMolecule(ifs, mol)
1
>>> for pair in OEGetSDDataPairs(mol):
...   print repr(pair.GetTag()), "=",
...   print repr(pair.GetValue())
...
'P1' = '0.12'
'$SMI' = 'c1cocn1'
>>> OEGetSDData(mol, "$SMI")
'c1cocn1'
>>> OESetSDData(mol, "P1",  "xyzzy")
1
>>> OEGetSDData(mol, "P1")
'xyzzy'
>>>
```

Add a "$SMI" tag

Process an SD file and add the "$SMI" tag to each record where the value is the canonical SMILES string

```
>>> from openeye.oechem import *
>>> ifs = oemolistream()
>>> ifs.open("drugs.sdf")
1
>>> ofs = oemolostream()
>>> ofs.open("drugs2.sdf")
1
>>> for mol in ifs.GetOEGraphMols():
...   OESetSDData(mol, "$SMI", OECreateCanSmiString(mol))
...   OEWriteMolecule(ofs, mol)
...
1
1
1
1
1
1
1
1
1
1
>>> ofs.close()
```

Example output

```
nicotine
  -OEChem-03010303112D

 12 13  0     0  0  0  0  0  0999 V2000
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 N   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 N   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
  1  6  2  0  0  0  0
  1  2  1  0  0  0  0
  2  3  2  0  0  0  0
  3  4  1  0  0  0  0
  4  5  2  0  0  0  0
  5  6  1  0  0  0  0
  6  7  1  0  0  0  0
  7 11  1  0  0  0  0
  7  8  1  0  0  0  0
  8  9  1  0  0  0  0
  9 10  1  0  0  0  0
 10 11  1  0  0  0  0
 11 12  1  0  0  0  0
M  END
> <$SMI>
CN1CCCC1c2cccnc2

$$$$
```

The new tag field

SMARTS searches

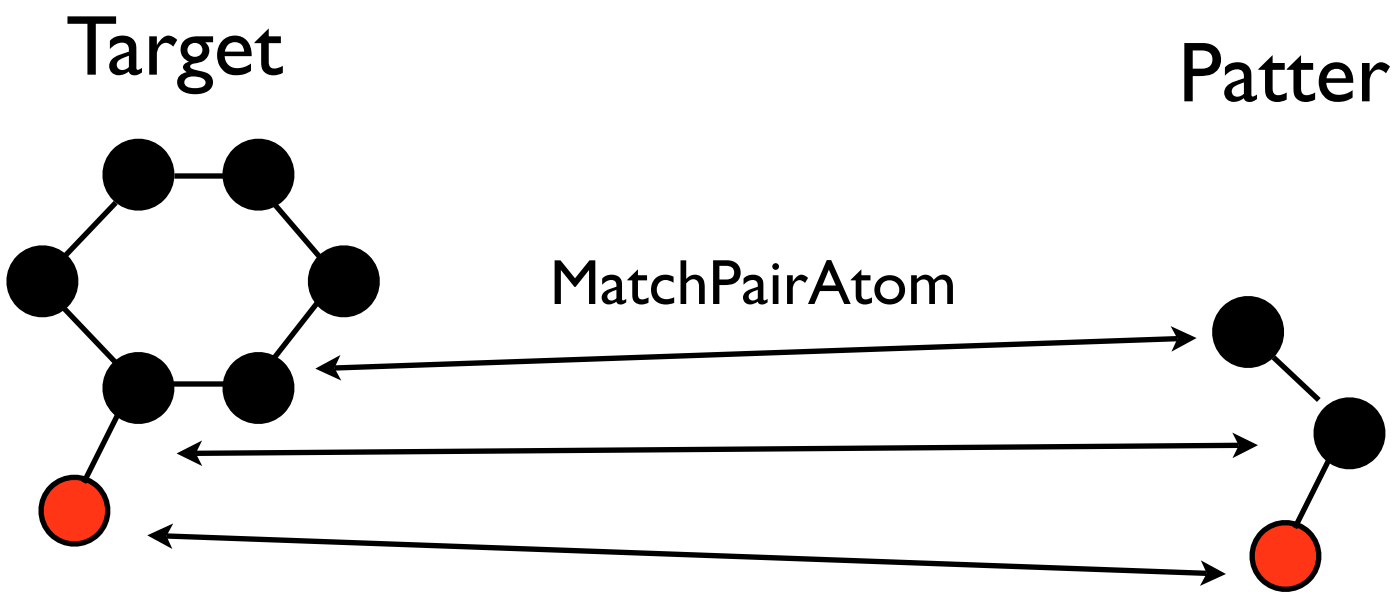
```
>>> from openeye.oechem import *
>>> pat = OESubSearch()
>>> pat.Init("C(=O)O")
1
>>> heroin = OEGraphMol()
>>> OEParseSmiles(heroin, "C123C5C(OC(=O)C)C=CC2C(N(C)CC1)Cc(ccc4OC(=O)C)c3c4O5")
1
>>> pat.Match(heroin)
<generator object at 0x17410d0>
>>> len(list(pat.Match(heroin)))
2
>>>
```

Using "Init" this way to avoid C++ exceptions

OEChem uses a lot of generators

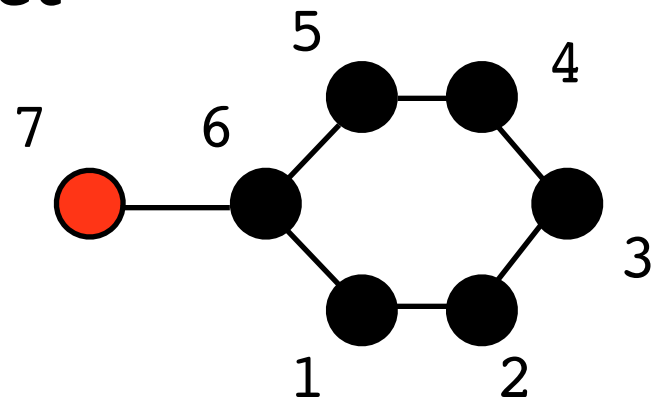
Match results

Each match result returns a mapping between the target (the molecule) and the pattern (the SMARTS)

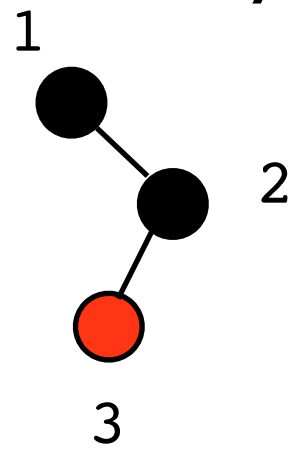


MatchBase is a “molecule”
Has GetAtoms(), GetBonds() which return MatchPairAtom and MatchPairBonds

Target



Query




```
>>> mol = OEGraphMol()
>>> OEParseSmiles(mol, "c1ccccc1O")
1
>>> for i, atom in enumerate(mol.GetAtoms()):
...   success = atom.SetName("T" + str(i+1))
...
>>> pat = OESubSearch()
>>> pat.Init("ccO")
1
>>> for i, atom in enumerate(pat.GetPattern().GetAtoms()):
...   success = atom.SetName("p" + str(i+1))
...
>>> for matchbase in pat.Match(mol):
...   print "Match",
...   for matchpair in matchbase.GetAtoms():
...     print "(%s, %s)" % (matchpair.target.GetName(), matchpair.pattern.GetName()),
...   print
...
Match (T1, p1) (T6, p2) (T7, p3)
Match (T5, p1) (T6, p2) (T7, p3)
>>>
```

All objects can be given a “Name”

Exercise 1 - smiles2sdf

Write a program that takes a SMILES file name on the command line and converts it to an SD file with two new tag fields. One field is named "SMILES" and contains the canonical SMILES string. The other is named "MW" and contains the molecular weight.

The SMILES file name will always end with ".smi" and the SD file name will be the SMILES file name + ".sdf".

Do not write your own molecular weight function.

Next page shows how your program should start.

Start of answer #1

```
# convert a SMILES file to an SD file
# The canonical SMILES will be added to the "SMILES" tag.
# The average molecular weight will be added to the "MW" tag.

import sys
from openeye.oechem import *

if len(sys.argv) != 2:
  sys.exit("wrong number of parameters")

smiles_filename = sys.argv[1]
if not smiles_filename.endswith(".smi"):
  sys.exit("SMILES filename must end with .smi")

sd_filename = smiles_filename + ".sdf"

  .... your code goes here ....
```

Exercise 2 - re-explore the NCI data set

Using the NCI SMILES data set as converted by CACTVS, and using OEChem this time, how many ...

1. ... SMILES are in the data set?
2. ... could not be processed by OEChem?
3. ... contain more than 30 atoms?
4. ... contain sulphers?
5. ... contain atoms other than N, C, O, S, and H?
6. ... contain more than one component in the SMILES?
7. ... have a linear chain of at least 15 atoms?

Are any of these different than the answers you got with Daylight?