Electronic Supplementary Material (ESI) for Organic & Biomolecular Chemistry.

Preparation and virtual screening of combinatorial compounds libraries

Dependencies: Python 2.x; OpenBabel with python bindings and Pybel; MGLTools.

1. Prepare sdf files for fragments (for example from downloaded list in Aldrich). Attachment point should be marked as a pseudo-atom *. This can be accomplished by utility **prepare_fragments.py.** It replaces a desired functional group in potential fragments with "*"
Usage:

```
prepare_fragments.py in_folder out_folder smarts_template
```

Before using this utility, it may be recommended to strip off all hydrogen atoms from the structures using, for example, Open Babel. After processing the files with **prepare_fragments.py** the hydrogens should then be attached and the resulting files inspected in a molecular viewer of your choice. While this is not strictly necessary, it may facilitate construction of appropriate SMILES codes for functional group specification. However, it may result in erroneous prediction of certain atom types later in the process (see below).

Example:

```
mkdir tmp F1
babel -m acids.smi tmp/C.pdb --gen3D
for f in tmp/*.pdb;do babel -d $f tmp/`basename $f pdb`sdf; done
rm tmp/*.pdb
prepare_fragments.py tmp F1 "[O]C(=O)"
for f in F1/*.sdf;do babel -p 7 $f F1/`basename $f sdf`pdb; done
rm tmp/*.sdf
for f in F1/*.pdb;do babel -p 7 $f F1/`basename $f pdb`sdf; done
rm F1/*.pdb
pymol F1/*
```

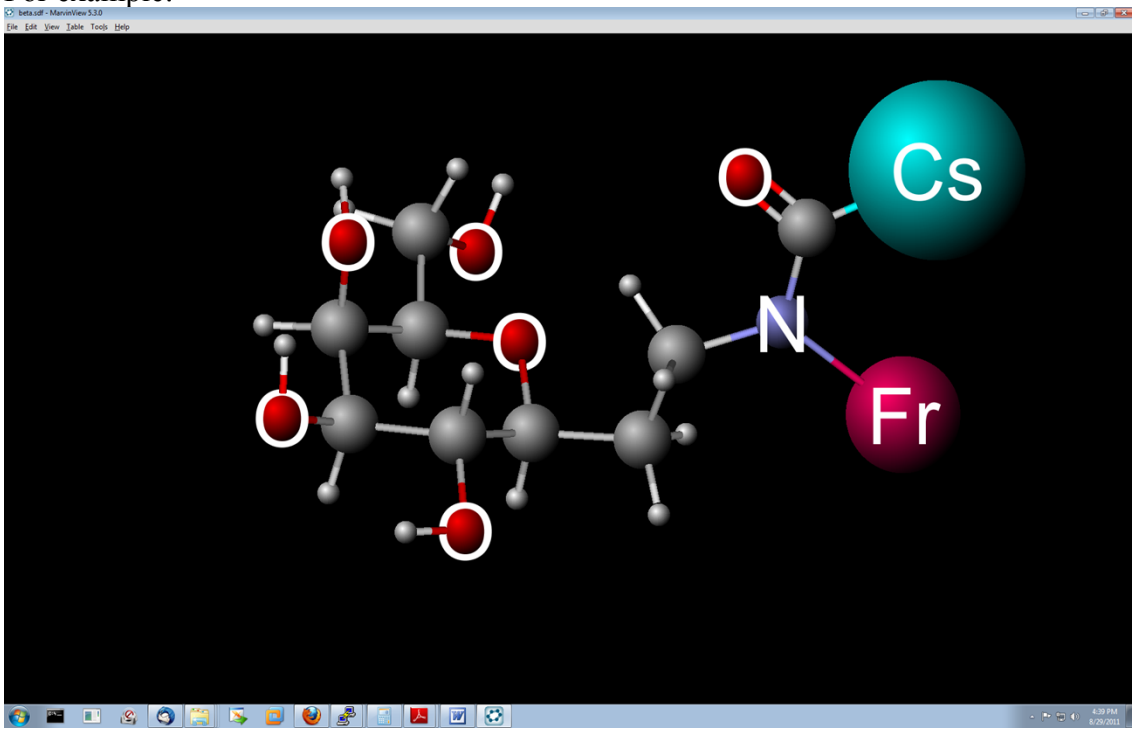
This script will create 3D representation of carboxylic acids structures containing in file `acids.smi` in SMILES format and place them into folder tmp. Then the hydrogens will be removed and the structures processed by reattachment of hydrogen to the fragments. Pymol can be used to repair possible mistakes of protonation. For example, babel added only one hydrogen atom to the methylene carbon in amine N3 automatically. This had to be corrected.

Analogously, the following script will prepare fragments from `amines.smi`.

```
rm -r tmp
mkdir tmp F2
babel -m amines.smi tmp/A.pdb --gen3D
for f in tmp/*.pdb;do babel -d $f tmp/`basename $f pdb`sdf; done
```

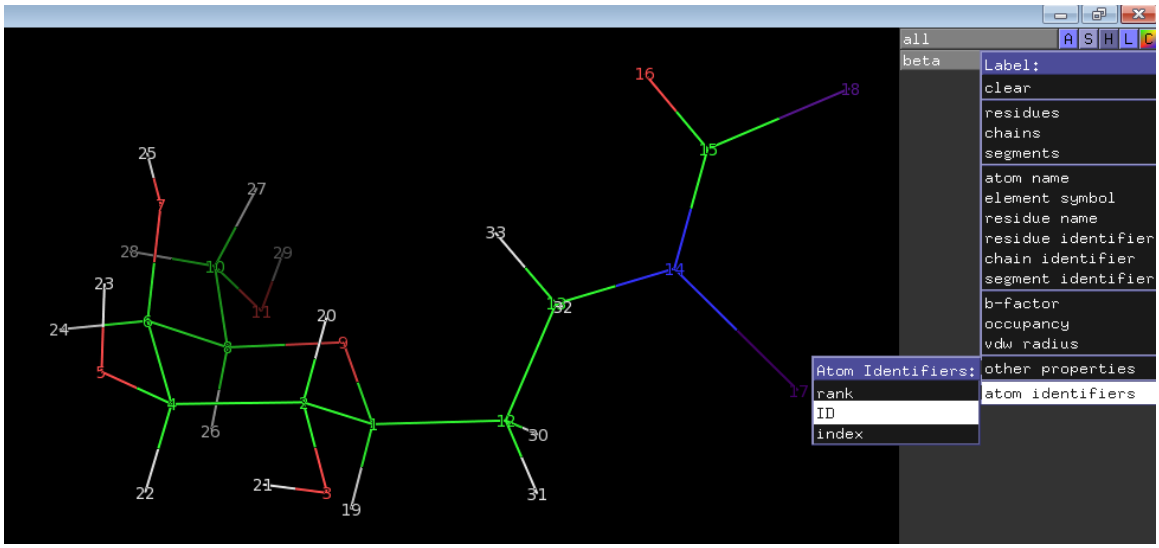
```
rm tmp/*.pdb
prepare_fragments.py tmp F2 "N"
for f in F2/*.sdf;do babel -p 7 $f F2/`basename $f sdf`pdb; done
rm tmp/*.sdf
for f in F2/*.pdb;do babel -p 7 $f F2/`basename $f pdb`sdf; done
rm F2/*.pdb
pymol F2/*
```

2. Prepare sdf file for scaffold. Attachment points should preferably be marked as univalent elements like Rb, Cs, Fr etc..
For example:



3. Restraint of ring atoms in carbohydrates is necessary to avoid large conformational distortions during geometry optimization of the virtual ligands.

Identify the atoms you would like to restrain during structure preparation (this improves chances of generating valid conformations). To do this, open scaffold file in PyMOL and label by atom ID:



Make a list of atoms you would like to restrain separated by comma.

4. Enumerate the library using utility **enumX.py**. For the first time, run this script in interactive mode (`enumX.py -I`). Alternatively, you may create a configuration file and run it with the flag "-C" (`enumX.py -C enum.conf`)

Example of configuration file for **enumX.py**:

```
fragment F1 37 acid-
# this is Rb pointer
fragment F2 55 amine-
# this is Cs pointer
scaffold trisaccharide.sdf
constrain atoms
2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,25,26,27,28,29,30 #
enumeration by atom ID in Pymol
representations 6 # the number of times each element will be
represented in the library. To make a complete combinatorial
library set this value higher or equal to the size of smallest
fragment set.
output list.txt
```

Now you can run (`enumX.py -C enum.conf`) and obtain the library enumeration file **list.txt**

5. Run utility **genX.py** with the file **list.txt** as a configuration file using the following bash commands.

```
mkdir products
genX.py products/ pdb list.txt 8 &
# 8 here is the number of CPUs to use
```

Wait for completion then find the pdb files for your library in products/ folder.

6. Convert pdb files for the ligands into pdbqt using utility **prepare_ligand4.py** from MGLTools. Torsional constrains can be introduced at this stage.

7. Conduct docking using Autodock Vina.

8. Select the poses with low RMSD with respect to the template structure (coordinates of a fragment of the ligand in the binding site). Apply utility **GoodPoses.py**. Usage:

```
GoodPoses.py test_file.pdbqt template_file.pdb
```

Note: the template_file.pdb should contain minimum features defining the location of the ligand in the binding site. It is recommended to remove all hydrogen atoms.

Files and scripts used in this example

These scripts have been tested to work after copying and pasting from this document. If these scripts are intended for use with a Linux computer it may be necessary to run the 'dos2unix' on the resultant file after copying and pasting the script contents.
prepare_fragments.py

```
#!/usr/bin/env python
from openbabel import *
from pybel import *
import os
import sys

if __name__ == '__main__':
      if len(sys.argv) != 4:
            print( "usage: ./prepare_fragments.py input_directory
output_directory functional_group_smiles ")
            sys.exit(0)

      if os.path.isdir(sys.argv[1]) == 0:
            print( sys.argv[1] +" is not a directory")
            sys.exit(-1)

      if os.path.isdir(sys.argv[2]) == 0:
            print( sys.argv[2] +" is not a directory")
            sys.exit(-1)

      templateSmarts = Smarts( sys.argv[3] )

      filesList = os.listdir(sys.argv[1])

      for i in range (0, len(filesList)):
            inputMol = OBMol()
            fileConversion = OBConversion()
```

```
          fileConversion.SetInAndOutFormats(
fileConversion.FormatFromExt( filesList[i] ),
fileConversion.FormatFromExt( filesList[i] ) )
          infile = os.path.join( sys.argv[1], filesList[i] )
          outfile = os.path.join( sys.argv[2], filesList[i] )
          fileConversion.ReadFile( inputMol, infile )
          pybelMol = Molecule( inputMol )
          matches = templateSmarts.findall( pybelMol )
          if len(matches) == 1:
               origMol = OBMol( inputMol )
               for y in sorted( matches[0], reverse = True ):
                    for neighborAtom in OBAtomAtomIter(
origMol.GetAtom( y ) ):
                         if neighborAtom.GetIdx() not in
matches[0]:
                              inputMol.GetAtom( y
).SetAtomicNum( 0 )
                              break
                    if inputMol.GetAtom( y ).GetAtomicNum() !=
0:
                         inputMol.DeleteAtom( inputMol.GetAtom(
y ) )

               fileConversion.WriteFile( inputMol, outfile )
          else:
               print( "error encountered.  file ID:" )
               print( filesList[i] )
               continue
```

enumX.py

```
#!/usr/bin/env python

import os
import sys
import string
import random
import math

if __name__ == '__main__':

  if( ( len( sys.argv ) < 2  or "-I" not in sys.argv[1] )
and ( ( len( sys.argv ) < 3 ) or ( "-C" not in sys.argv[1]
) ) ):
```

```
    print( "Usage:\n\t./enumX.py -I\t\t\tInteractive
mode\n\t./enumX -C config_file\t\tPreconfigured
mode\n\nExiting.\n" )
    sys.exit(1)

  fragments=[]
  heavyNumbers=[]
  codeWords=[]
  scaffoldList = []
  fileList = []
  outputpath = ""
  if "-I" in sys.argv[1]:
    scaffoldPath = raw_input("What is the name of the
directory containing the scaffold(s)?  Type the path and
press enter:  ")
    if os.path.isdir(scaffoldPath) != 1:
      print("Error: path not entered.")

    thisFragment = raw_input("What is the name of the
directory containing the fragments?\nEnter a blank line
when done:  ")
    while os.path.isdir(thisFragment):
      fragments.append(thisFragment)
      thisNumber = raw_input("What is the (integer) atomic
number of the placeholder atom for this fragment?  ")
      heavyNumbers.append(int(thisNumber))
      thisCode = raw_input("What is the code for this
fragment? (M for aMines, L for aLdehyde, S for Sugar):  ")
      codeWords.append(thisCode)
      thisFragment = raw_input("What is the name of the
directory containing the fragments?\nEnter a blank line
when done:  ")

    reprNum = raw_input("What is the minimum number of
combinations to be formed from each building block set?
\nEnter your choice:  ")
    constraints = raw_input( "If you wish to constrain any
atoms in your scaffold, enter the atom numbers separated by
commas.\nFor example: 1,3,4,5 would constrain atoms 1, 3,
4, and 5.  \nEnter your constraints:  " )
    if constraints == "":
      constraints = ','
    if os.path.isdir( scaffoldPath ):
      for scaffolds in os.listdir(scaffoldPath):
        scaffoldList.append( os.path.join( scaffoldPath ,
scaffolds ) )
    elif os.path.exists( scaffoldPath ):
```

```
      scaffoldList.append( scaffoldPath )

  if "-C" in sys.argv[1] and os.path.exists( sys.argv[2] ):
    configFile = open( sys.argv[2], "r" )
    lineList = configFile.readlines()
    reprNum = 1
    constraints = ","
    for lines in lineList:
      words = lines.split()
      if len( words ) == 0:
        continue

      if( words[0][0] == "#" ):
        continue

      elif words[0] == "fragment" and os.path.isdir(
words[1] ) and isinstance( int( words[2] ), int ):
        fragments.append( lines.split()[1] )
        heavyNumbers.append( int( words[2] ) )
        if len( words ) > 3:
          codeWords.append( words[3] )
        else:
          codeWords.append( "" )
        continue

      elif ( ( words[0] == "scaffold" ) and os.path.exists(
words[1] ) ):
        if os.path.isdir( words[1] ):
          for scaffold in os.listdir( words[1] ):
            scaffoldList.append( os.path.join( words[1],
scaffold ) )
        else:
          scaffoldList.append( words[1] )
        continue

      elif ( ( len( words ) == 3 ) and ( words[1]  ==
"atoms" ) and ( "," in words[2]) ):
        constraints = words[2]
        continue

      elif words[0] == "representations" :
        reprNum = int( words[1] )
        if isinstance( reprNum, int ):
          continue
      elif ( ( words[0] == "output" ) and ( len( words ) ==
2 ) ):
        outputpath = words[1]
```

```
        continue

      print("Invalid line: \n"+lines+"\nin configuration
file: "+sys.argv[2]+"\n" )
      sys.exit(1)

  products = []
  maxFiles = 0
  fileLists = []
  fileTable = [[]]

  delim = os.path.join( "a", "b" )[1]

  fileLists.append( scaffoldList )

  for x in range ( len(heavyNumbers) ):
    fileList = []
    for fileName in os.listdir( fragments[x] ):
      fileList.append( os.path.join( fragments[x], fileName
)+" "+`heavyNumbers[x]` )

    fileLists.append( fileList )

    if len( fileList ) > maxFiles:
      maxFiles = len( fileList )
    fileTable.append([])

  numSamples = int( reprNum ) * maxFiles

  for y in range( int( math.ceil( numSamples/ float( len(
fileLists[0] ) ) ) ) ):
    for scaffold in scaffoldList:
      fileTable[0].append( scaffold )

  for x in range( 1, len( fileLists ) ):
    for y in range( int(math.ceil( numSamples / float( len(
fileLists[x] ) ) ) ) ):
      random.shuffle( fileLists[x] )
      for file in fileLists[x]:
        for i in range( len( scaffoldList ) ):
          fileTable[x].append(file)

  for y in range( numSamples ):
    line = fileTable[0][y] + " " + constraints
    code = fileTable[0][y].split(delim)[1].rsplit('.')[0]
    for x in range( 1, len( fileTable ) ):
      if( codeWords[ (x - 1) ] != "" ):
```

```
        code += "_"+codeWords[( x - 1
)]+fileTable[x][y].split(delim)[1].rsplit('.')[0]
      line += " "+fileTable[x][y]
    products.append( code+" "+line )

  while outputpath == "":
    outputpath = raw_input( "Please enter the output
filename:  " )

  outFile = open(outputpath, "w")
  for line in products:
    outFile.write(line+"\n")
```

genX.py

```
#!/usr/bin/env python
from openbabel import *
import os
import openbabel
import sys
import math
import string

def merge( scaffoldmol, fragmentmol, atomNum, outname, atomCons
):
      outputConversion = openbabel.OBConversion()
      outputConversion.SetOutFormat(outputConversion.FormatFromEx
t(".mol2"))
      product = openbabel.OBMol()
      product.BeginModify()

      for anyAtom in openbabel.OBMolAtomIter(scaffoldmol):
            if anyAtom.GetAtomicNum() == atomNum:
                  for scaffoldAnchor in
openbabel.OBAtomAtomIter(anyAtom):
                        scaffoldAIndex = scaffoldAnchor.GetIdx()
                        scaffoldBIndex = anyAtom.GetIdx()

      for anyAtom in openbabel.OBMolAtomIter(fragmentmol):
            if anyAtom.GetAtomicNum() == 0:
                  for fragmentAnchor in
openbabel.OBAtomAtomIter(anyAtom):
                        fragmentBIndex = fragmentAnchor.GetIdx() +
scaffoldmol.NumAtoms()
                        fragmentAIndex = anyAtom.GetIdx() +
scaffoldmol.NumAtoms()

      myElements = openbabel.OBElementTable()
      resNum = 1
```

```
      for anyResidue in openbabel.OBResidueIter(scaffoldmol):
            product.AddResidue(anyResidue)
            productResidue =
product.GetResidue(anyResidue.GetIdx())
            productResidue.SetName( "LIG"+`resNum` )
            resNum += 1

      for anyAtom in openbabel.OBMolAtomIter(scaffoldmol):
            product.AddAtom(anyAtom)
            productAtom = product.GetAtom(anyAtom.GetIdx())
            atomsResidue =
product.GetResidue(anyAtom.GetResidue().GetIdx())
            atomsResidue.AddAtom(productAtom)
            atomsResidue.SetAtomID( productAtom,
myElements.GetSymbol( productAtom.GetAtomicNum() ) + str(
productAtom.GetIdx() ) )

      for anyBond in openbabel.OBMolBondIter(scaffoldmol):
            product.AddBond(anyBond.GetBeginAtom().GetIdx(),
anyBond.GetEndAtom().GetIdx(), anyBond.GetBondOrder())

      for anyAtom in openbabel.OBMolAtomIter(fragmentmol):
            product.AddAtom(anyAtom)

      for anyBond in openbabel.OBMolBondIter(fragmentmol):
            product.AddBond(anyBond.GetBeginAtom().GetIdx() +
scaffoldmol.NumAtoms(), anyBond.GetEndAtom().GetIdx() +
scaffoldmol.NumAtoms(), anyBond.GetBondOrder())

      myField = openbabel.OBForceField.FindForceField("UFF")
      constraints = OBFFConstraints()
      fragmentRes = openbabel.OBResidue()
      fragmentRes.SetName("LIG"+`resNum`)
      fragmentRes.SetNum(resNum)
      product.AddResidue( fragmentRes )

      for atomID in range ( scaffoldmol.NumAtoms() + 1 ,
scaffoldmol.NumAtoms() + fragmentmol.NumAtoms() + 1 ):
            atom = product.GetAtom(atomID)
            fragmentRes.AddAtom(atom)
            fragmentRes.SetAtomID(atom,
myElements.GetSymbol(atom.GetAtomicNum()) + str(atom.GetIdx()))
      for atomIDs in atomCons:
            constraints.AddAtomConstraint( atomIDs )

      myBuilder = openbabel.OBBuilder()
      attempts = 0

      while ( myBuilder.Swap( product, scaffoldAIndex,
scaffoldBIndex, fragmentAIndex, fragmentBIndex ) == 0 ):
            if( attempts ) == 20:
```

```
                sys.exit(-1)

            attempts += 1

            if myField.Setup(product, constraints):
                myField.WeightedRotorSearch( 5,5 )
                myField.GetCoordinates(product)

      product.DeleteAtom(product.GetAtom(fragmentAIndex))
      product.DeleteAtom(product.GetAtom(scaffoldBIndex))
      outputConversion.WriteFile(product, outname+".mol2")
      outputConversion.CloseOutFile()


def repeatLoop( fragmentPath, heavyNumbers, scaffoldPath,
outPath, atomCons ):
      obconversionSc = OBConversion()
      obconversionFr = OBConversion()
      fragmol = OBMol()
      scaffoldmol = OBMol()
      obconversionSc.SetInAndOutFormats(
obconversionSc.FormatFromExt(scaffoldPath),
obconversionSc.FindFormat("pdb") )
      obconversionFr.SetInFormat( obconversionSc.FormatFromExt(
fragmentPath[0] ) )
      obconversionSc.ReadFile( scaffoldmol, scaffoldPath )
      obconversionFr.ReadFile( fragmol, fragmentPath[0] )
      product = OBMol()
      product.BeginModify()
      myElements = openbabel.OBElementTable()

      for anyAtom in openbabel.OBMolAtomIter(scaffoldmol):
            product.AddAtom(anyAtom)
            productAtom = product.GetAtom(anyAtom.GetIdx())

      for anyBond in openbabel.OBMolBondIter(scaffoldmol):
            product.AddBond(anyBond.GetBeginAtom().GetIdx(),
anyBond.GetEndAtom().GetIdx(), anyBond.GetBondOrder())

      obconversionSc.WriteFile( product, outPath+".mol2" )
      obconversionSc.CloseOutFile()
      obconversionSc.SetInAndOutFormats(
obconversionSc.FindFormat("pdb"),
obconversionSc.FindFormat("mol2") )
      obconversionSc.ReadFile( scaffoldmol, outPath+".mol2" )
      obconversionSc.SetInFormat(
obconversionSc.FindFormat("mol2"))
      merge( scaffoldmol, fragmol, int(heavyNumbers[0]), outPath,
atomCons )
      for x in range ( 1 , len(heavyNumbers) ):
            obconversionSc.ReadFile(scaffoldmol, outPath+".mol2")
```

```
	obconversionFr.SetInFormat(obconversionFr.FormatFromExt(fra
gmentPath[x]))
		obconversionFr.ReadFile(fragmol, fragmentPath[x])
		merge( scaffoldmol, fragmol, int(heavyNumbers[x]),
outPath, atomCons )

	obconversionSc.ReadFile( product, outPath+".mol2" )
	obconversionSc = OBConversion()

	for residue in OBResidueIter(product):
		residue.SetName(residue.GetName()[0:3])

	constraints = OBFFConstraints()
	for atomID in atomCons:
		constraints.AddAtomConstraint( atomID )

	obconversionSc.SetOutFormat( obconversionSc.FormatFromExt(
outPath ) )
	obFF = OBForceField.FindForceField( "MMFF94s" )
	if obFF.Setup( product, constraints ):
		obFF.WeightedRotorSearch( 200, 100 )
#polish the final rotamer
		obFF.SteepestDescent( 1000 )
		obFF.ConjugateGradients( 5000, 0.0000001 )
		obFF.GetCoordinates( product )

		product.SetTitle( outPath )
		obconversionSc.WriteFile( product, outPath )
		obconversionSc.CloseOutFile()
		os.remove( outPath+".mol2" )
	else:
		print( "Failed to set up MMFF94 for "+outPath )

if __name__ == '__main__':

	if len(sys.argv) < 4:
		print("Usage: gen6.py <output directory> <format>
<library list> <num jobs>.")
		sys.exit(-1)

	f = open(sys.argv[3], 'r')
	line=f.readline().rsplit()
	forks = 0
	while len(line) > 3:
		heavynumbers = []
		fragmentPath = []
		scaffoldPath = line[1]
		outPath =
os.path.join(sys.argv[1],line[0]+"."+sys.argv[2])

		atomCons = []
```

```
            atomIDs = line[2].split( ',' )
            for atomID in atomIDs:
                  try:
                        atomCons.append( int( atomID ) )
                  except:
                        continue

            fragmentPath.append(line[3])
            heavynumbers.append(line[4])
            for x in range( 2, ( len(line) ) / 2 ):
                  fragmentPath.append( line[ 2 * x + 1 ] )
                  heavynumbers.append( line[ 2 * x + 2 ] )

            if os.path.exists( outPath ) == 0:
                  pid = os.fork()
                  if pid:
                        forks += 1
                        while forks >= int(sys.argv[4]):
                              os.wait()
                              forks -= 1

                  else:
                        repeatLoop( fragmentPath, heavynumbers,
scaffoldPath, outPath, atomCons )
                        sys.exit(0)

            line=f.readline().rsplit()
```

---

acids.smi

```
FC1=CC(CC(O)=O)=CC=C1 C1
OC(CC1CCCC1)=O   C2
OC(CC1=CC=C(C)C=C1)=O C3
OC(/C(C)=C/CC)=O C4
OC(C1=CC=CC([N+]([O-])=O)=C1)=O  C5
OC(C1=CC=C(N(N=C(C)C2)C2=O)C=C1)=O    C6
```

---

amines.smi

```
NC1=CC(OCCO2)=C2C=C1  N1
N[C@H]1CCCC2=CC=CC=C21      N2
NC1=CC(OCO2)=C2C=C1   N3
NC1=CC(CCCC2)=C2C=C1  N4
NCCC1=NC=CC=C1   N5
NC1=CC=C(OC)C=C1 N6
```

---

trisaccharide.sdf

```
scaffold.pdb
 OpenBabel10131422163D

 83 85  0  0  1  0  0  0  0  0999 V2000
    4.3360   -5.7220   -0.7990 N   0  0  0  0  0  0  0  0  0  0  0
0  0
   -5.0190    7.0720    3.2340 C   0  0  1  0  0  0  0  0  0  0  0
0  0
   -6.1110    6.1580    3.4360 O   0  0  0  0  0  0  0  0  0  0  0
0  0
   -5.2430    8.4230    3.9320 C   0  0  1  0  0  0  0  0  0  0  0
0  0
   -3.8000    6.4570    3.6690 O   0  0  0  0  0  0  0  0  0  0  0
0  0
   -5.8360    8.2450    5.3520 C   0  0  2  0  0  0  0  0  0  0  0
0  0
   -1.2180    3.9780    1.5280 O   0  0  0  0  0  0  0  0  0  0  0
0  0
   -6.9780    7.2120    5.4320 C   0  0  2  0  0  0  0  0  0  0  0
0  0
   -0.5510    2.8740    3.4940 O   0  0  0  0  0  0  0  0  0  0  0
0  0
   -6.4710    5.9010    4.8130 C   0  0  1  0  0  0  0  0  0  0  0
0  0
    1.1710   -0.3180    3.2540 O   0  0  0  0  0  0  0  0  0  0  0
0  0
   -1.6710    3.4190    2.7730 C   0  0  1  0  0  0  0  0  0  0  0
0  0
    3.3500   -0.9100    2.6960 O   0  0  0  0  0  0  0  0  0  0  0
0  0
   -2.2970    4.5220    3.6440 C   0  0  1  0  0  0  0  0  0  0  0
0  0
    4.5450   -3.4590    1.9960 O   0  0  0  0  0  0  0  0  0  0  0
0  0
   -3.3780    5.3020    2.8790 C   0  0  2  0  0  0  0  0  0  0  0
0  0
    2.4730   -5.7810   -2.0290 O   0  0  0  0  0  0  0  0  0  0  0
0  0
   -2.8970    5.7490    1.4730 C   0  0  2  0  0  0  0  0  0  0  0
0  0
   -2.2540    4.5670    0.7120 C   0  0  1  0  0  0  0  0  0  0  0
0  0
    2.5560    0.1670    3.1870 C   0  0  1  0  0  0  0  0  0  0  0
0  0
    2.6350    1.3650    2.2130 C   0  0  2  0  0  0  0  0  0  0  0
0  0
    3.9340    1.9440    2.2630 O   0  0  0  0  0  0  0  0  0  0  0
0  0
```

```
    1.6220    2.4400    2.6160 C   0  0  2  0  0  0  0  0  0  0
0  0
   -1.5020   -0.5850    2.5660 O   0  0  0  0  0  0  0  0  0  0
0  0
    0.2120    1.8680    2.8180 C   0  0  1  0  0  0  0  0  0  0
0  0
    1.5710    3.4340    1.5910 O   0  0  0  0  0  0  0  0  0  0
0  0
    0.2160    0.6500    3.7380 C   0  0  1  0  0  0  0  0  0  0
0  0
   -2.9070    3.9140    4.7840 O   0  0  0  0  0  0  0  0  0  0
0  0
    3.3960   -2.0520    3.5850 C   0  0  0  0  0  0  0  0  0  0
0  0
   -1.0720    3.9130   -1.3300 O   0  0  0  0  0  0  0  0  0  0
0  0
    3.3810   -3.3530    2.7700 C   0  0  0  0  0  0  0  0  0  0
0  0
   -2.0120    6.8770    1.5570 O   0  0  0  0  0  0  0  0  0  0
0  0
    4.8310   -4.7110    1.3740 C   0  0  0  0  0  0  0  0  0  0
0  0
   -4.0370    9.2150    3.9680 O   0  0  0  0  0  0  0  0  0  0
0  0
    3.7700   -5.0520    0.3480 C   0  0  0  0  0  0  0  0  0  0
0  0
   -8.5240    4.9350    3.9530 O   0  0  0  0  0  0  0  0  0  0
0  0
    3.6400   -6.0490   -1.8960 C   0  0  0  0  0  0  0  0  0  0
0  0
   -6.2750    9.5130    5.8340 O   0  0  0  0  0  0  0  0  0  0
0  0
   -8.1400    7.6290    4.7380 O   0  0  0  0  0  0  0  0  0  0
0  0
   -1.1490   -0.0180    3.8180 C   0  0  0  0  0  0  0  0  0  0
0  0
   -1.6930    4.9960   -0.6500 C   0  0  0  0  0  0  0  0  0  0
0  0
   -7.4930    4.7650    4.8990 C   0  0  0  0  0  0  0  0  0  0
0  0
    5.7450   -6.0620   -0.7610 Cs  0  0  0  0  0  0  0  0  0  0
0  0
    4.8340   -7.0350   -3.3730 Rb  0  0  0  0  0  0  0  0  0  0
0  0
   -4.9560    7.2880    2.1880 H   0  0  0  0  0  0  0  0  0  0
0  0
   -5.9620    8.9570    3.3470 H   0  0  0  0  0  0  0  0  0  0
0  0
   -5.0550    7.8530    5.9700 H   0  0  0  0  0  0  0  0  0  0
0  0
   -7.2500    7.0920    6.4600 H   0  0  0  0  0  0  0  0  0  0
0  0
```

```
   -5.6200    5.5730    5.3730 H   0  0  0  0  0  0  0  0  0  0
0  0
   -2.3910    2.6560    2.5620 H   0  0  0  0  0  0  0  0  0  0
0  0
   -1.5220    5.2020    3.9290 H   0  0  0  0  0  0  0  0  0  0
0  0
   -4.2100    4.6470    2.7270 H   0  0  0  0  0  0  0  0  0  0
0  0
   -3.7520    6.0660    0.9140 H   0  0  0  0  0  0  0  0  0  0
0  0
   -3.0150    3.8410    0.5190 H   0  0  0  0  0  0  0  0  0  0
0  0
    2.9020    0.4880    4.1470 H   0  0  0  0  0  0  0  0  0  0
0  0
    2.4220    1.0100    1.2260 H   0  0  0  0  0  0  0  0  0  0
0  0
    4.1120    2.2610    3.1620 H   0  0  0  0  0  0  0  0  0  0
0  0
    1.9430    2.8560    3.5480 H   0  0  0  0  0  0  0  0  0  0
0  0
   -2.3730   -1.0060    2.6360 H   0  0  0  0  0  0  0  0  0  0
0  0
   -0.1810    1.5870    1.8630 H   0  0  0  0  0  0  0  0  0  0
0  0
    1.2770    3.0270    0.7610 H   0  0  0  0  0  0  0  0  0  0
0  0
    0.4790    0.9940    4.7160 H   0  0  0  0  0  0  0  0  0  0
0  0
   -3.6030    3.3050    4.4920 H   0  0  0  0  0  0  0  0  0  0
0  0
    2.5450   -2.0300    4.2340 H   0  0  0  0  0  0  0  0  0  0
0  0
    4.2950   -2.0110    4.1640 H   0  0  0  0  0  0  0  0  0  0
0  0
   -0.7300    4.2210   -2.1840 H   0  0  0  0  0  0  0  0  0  0
0  0
    3.3250   -4.1870    3.4380 H   0  0  0  0  0  0  0  0  0  0
0  0
    2.5320   -3.3450    2.1190 H   0  0  0  0  0  0  0  0  0  0
0  0
   -1.2200    6.6300    2.0590 H   0  0  0  0  0  0  0  0  0  0
0  0
    4.8570   -5.4770    2.1210 H   0  0  0  0  0  0  0  0  0  0
0  0
    5.7780   -4.6420    0.8810 H   0  0  0  0  0  0  0  0  0  0
0  0
   -3.3640    8.7550    4.4930 H   0  0  0  0  0  0  0  0  0  0
0  0
    3.0430   -5.6930    0.8010 H   0  0  0  0  0  0  0  0  0  0
0  0
    3.3210   -4.1400    0.0150 H   0  0  0  0  0  0  0  0  0  0
0  0
```

```
   -9.1560    4.2030    4.0290 H   0  0  0  0  0  0  0  0  0  0
0  0
   -6.9810    9.8500    5.2610 H   0  0  0  0  0  0  0  0  0  0
0  0
   -7.9300    7.7490    3.7990 H   0  0  0  0  0  0  0  0  0  0
0  0
   -1.1200   -0.7900    4.5580 H   0  0  0  0  0  0  0  0  0  0
0  0
   -1.8790    0.7190    4.0810 H   0  0  0  0  0  0  0  0  0  0
0  0
   -0.9700    5.7700   -0.4990 H   0  0  0  0  0  0  0  0  0  0
0  0
   -2.5080    5.3430   -1.2510 H   0  0  0  0  0  0  0  0  0  0
0  0
   -7.9190    4.7520    5.8810 H   0  0  0  0  0  0  0  0  0  0
0  0
   -6.9920    3.8420    4.6930 H   0  0  0  0  0  0  0  0  0  0
0  0
  1 35  1  0  0  0  0
  1 37  1  0  0  0  0
  1 43  1  0  0  0  0
  2  3  1  0  0  0  0
  2  4  1  0  0  0  0
  2  5  1  0  0  0  0
  2 45  1  6  0  0  0
  3 10  1  0  0  0  0
  4  6  1  0  0  0  0
  4 34  1  0  0  0  0
  4 46  1  6  0  0  0
  5 16  1  0  0  0  0
  6  8  1  0  0  0  0
  6 38  1  0  0  0  0
  6 47  1  1  0  0  0
  7 12  1  0  0  0  0
  7 19  1  0  0  0  0
  8 10  1  0  0  0  0
  8 39  1  0  0  0  0
  8 48  1  1  0  0  0
  9 12  1  0  0  0  0
  9 25  1  0  0  0  0
 10 42  1  0  0  0  0
 10 49  1  1  0  0  0
 11 20  1  0  0  0  0
 11 27  1  0  0  0  0
 12 14  1  0  0  0  0
 12 50  1  6  0  0  0
 13 20  1  0  0  0  0
 13 29  1  0  0  0  0
 14 16  1  0  0  0  0
 14 28  1  0  0  0  0
 14 51  1  1  0  0  0
 15 31  1  0  0  0  0
```

```
 15 33  1  0  0  0  0
 16 18  1  0  0  0  0
 16 52  1  6  0  0  0
 17 37  2  0  0  0  0
 18 19  1  0  0  0  0
 18 32  1  0  0  0  0
 18 53  1  6  0  0  0
 19 41  1  0  0  0  0
 19 54  1  6  0  0  0
 20 21  1  0  0  0  0
 20 55  1  1  0  0  0
 21 22  1  0  0  0  0
 21 23  1  0  0  0  0
 21 56  1  6  0  0  0
 22 57  1  0  0  0  0
 23 25  1  0  0  0  0
 23 26  1  0  0  0  0
 23 58  1  1  0  0  0
 24 40  1  0  0  0  0
 24 59  1  0  0  0  0
 25 27  1  0  0  0  0
 25 60  1  6  0  0  0
 26 61  1  0  0  0  0
 27 40  1  0  0  0  0
 27 62  1  1  0  0  0
 28 63  1  0  0  0  0
 29 31  1  0  0  0  0
 29 64  1  0  0  0  0
 29 65  1  0  0  0  0
 30 41  1  0  0  0  0
 30 66  1  0  0  0  0
 31 67  1  0  0  0  0
 31 68  1  0  0  0  0
 32 69  1  0  0  0  0
 33 35  1  0  0  0  0
 33 70  1  0  0  0  0
 33 71  1  0  0  0  0
 34 72  1  0  0  0  0
 35 73  1  0  0  0  0
 35 74  1  0  0  0  0
 36 42  1  0  0  0  0
 36 75  1  0  0  0  0
 37 44  1  0  0  0  0
 38 76  1  0  0  0  0
 39 77  1  0  0  0  0
 40 78  1  0  0  0  0
 40 79  1  0  0  0  0
 41 80  1  0  0  0  0
 41 81  1  0  0  0  0
 42 82  1  0  0  0  0
 42 83  1  0  0  0  0
M  END
```

$$$$

GoodPoses.py

```
#!/usr/bin/env python
import os
import sys
from openbabel import *
import pybel
from math import *

#if len( sys.argv ) < 3:
#    print "Usage:\n\tctGoodPoses.py test_file.pdbqt
template_file.pdb\n"
#    sys.exit(0)

#file to be tested
testFile = sys.argv[1]

#template file for filtering invalid results
templateFile = sys.argv[2]

#list of atom IDs from template to exclude
#excluded = [0,1,2,3]
excluded = []

#or if you want to specify them manually, append 'em to
your arguments:
for items in range( 3, len( sys.argv ) ):
     excluded.append( sys.argv[items] )

#cutoff (RMSD angstroms) for filtering
filterCutoff = 1.5
if len( sys.argv ) == 4:
  filterCutoff = float(sys.argv[3])

def obtainMatches( template, test ):
     matchMaker = pybel.Smarts( pybel.Molecule( template
).write() )
     matches = matchMaker.findall( pybel.Molecule( test ) )
     return matches

def getTemplateCoords( template, tmplMask ):
     returnList = []
     matches = obtainMatches( template, template )
```

```
     if not len( matches ) == 1:
          return returnList

     templateMatch = matches[0]

     for atomID in templateMatch:
          theAtom = template.GetAtom(atomID)

          if (atomID in tmplMask ) or (
theAtom.GetAtomicNum() == 1 ):
               returnList.append( [] )

          else:
               returnList.append( [theAtom.x(),
theAtom.y(), theAtom.z()] )

     return returnList


def compare( template, tstStrct, tmplMask ):
     totalPoseDev = 0
     atomsCompared = 0

     templateCoords = getTemplateCoords( template, tmplMask
)
     matches = obtainMatches( template, tstStrct )

     if len( templateCoords ) == 0:
          return -1

     if not len( matches ) == 1:
          return -1

     for listID in range( len( templateCoords ) ):
          tmpCoord = templateCoords[ listID ]

          if len( tmpCoord ) == 3:
               testAtom = tstStrct.GetAtom(
matches[0][listID] )
               thisDev = \
               (
                    pow( testAtom.x() - tmpCoord[0], 2 ) +\
                    pow( testAtom.y() - tmpCoord[1], 2 ) +\
                    pow( testAtom.z() - tmpCoord[2], 2 ) \
               )
               totalPoseDev += thisDev
               atomsCompared += 1
```

```
     return sqrt( totalPoseDev / atomsCompared )


if __name__ == '__main__':
     if not os.path.exists( testFile ):
          print( "0" )
          sys.exit(0)
     testFormat = os.path.splitext( testFile )[1][1:]
     templateFormat = os.path.splitext( templateFile
)[1][1:]
     pybMols = list( pybel.readfile( testFormat, testFile )
)
     template = pybel.readfile( templateFormat,
templateFile ).next().OBMol
     numGoodPoses = 0
     for molNum in range( len( pybMols ) ):
          pybMol = pybMols[molNum].OBMol
          pybMol.AddHydrogens()
          result = compare( template, pybMol, excluded )
          if result == -1:
               sys.stderr.write( "Invalid - no structural
match found for structure %s on pose number %d" % (
testFile, molNum ) )
          elif result < filterCutoff:
               numGoodPoses += 1
     print( "%d" % numGoodPoses )
     sys.exit(0)
```