NAME

RDKitFilterTorsionLibraryAlerts.py - Filter torsion library alerts

SYNOPSIS

RDKitFilterTorsionLibraryAlerts.py [--alertsMode <Red, RedAndOrange>] [--alertsMinCount <Number>] [--infileParams <Name,Value,...>] [--mode <filter or count>] [--mp <yes or no>] [--mpParams <Name,Value,...>] [--nitrogenLonePairParams <Name,Value,...>] [--outfileAlerts <yes or no>] [--outfileAlertsMode <All or AlertsOnly>] [--outfileFiltered <yes or no>] [--outfilesFilteredByRules <yes or no>] [--outfilesFilteredByRulesMaxCount <All or number>] [--outfileSummary <yes or no>] [--outfileSDFieldLabels <Type,Label,...>] [--outfileParams <Name,Value,...>] [--overwrite] [--rotBondsSMARTSMode <NonStrict, SemiStrict,...>] [--rotBondsSMARTSPattern <SMARTS>] [--torsionLibraryFile <FileName or auto>] [-w <dir>] -i <infile> -o <outfile>

RDKitFilterTorsionLibraryAlerts.py [--torsionLibraryFile <FileName or auto>] -l | --list

RDKitFilterTorsionLibraryAlerts.py -h | --help | -e | --examples

DESCRIPTION

Filter strained molecules from an input file for torsion library [Ref 146, 152, 159] alerts by matching rotatable bonds against SMARTS patterns specified for torsion rules in a torsion library file and write out appropriate molecules to output files. The molecules must have 3D coordinates in input file. The default torsion library file, TorsionLibrary.xml, is available under MAYACHEMTOOLS/lib/data directory.

The data in torsion library file is organized in a hierarchical manner. It consists of one generic class and six specific classes at the highest level. Each class contains multiple subclasses corresponding to named functional groups or substructure patterns. The subclasses consist of torsion rules sorted from specific to generic torsion patterns. The torsion rule, in turn, contains a list of peak values for torsion angles and two tolerance values. A pair of tolerance values define torsion bins around a torsion peak value. For example:

```
<library>
    <hierarchyClass name="GG" id1="G" id2="G">
    ...
    </hierarchyClass>
    <hierarchyClass name="CO" id1="C" id2="O">
        <hierarchySubClass name="Ester bond I" smarts="O=[C:2][O:3]">
            <torsionRule smarts="[O:1]=[C:2]!@[O:3]~[CH0:4]">
                <angleList>
                    <angle value="0.0" tolerance1="20.00"
                     tolerance2="25.00" score="56.52"/>
                </angleList>
            </torsionRule>
            ...
        ...
     ...
    </hierarchyClass>
    <hierarchyClass name="NC" id1="N" id2="C">
     ...
    </hierarchyClass>
    <hierarchyClass name="SN" id1="S" id2="N">
    ...
    </hierarchyClass>
    <hierarchyClass name="CS" id1="C" id2="S">
    ...
    </hierarchyClass>
    <hierarchyClass name="CC" id1="C" id2="C">
    ...
    </hierarchyClass>
    <hierarchyClass name="SS" id1="S" id2="S">
     ...
    </hierarchyClass>
</library>
```

The rotatable bonds in a 3D molecule are identified using a default SMARTS pattern. A custom SMARTS pattern may be optionally specified to detect rotatable bonds. Each rotatable bond is matched to a torsion rule in the torsion library and assigned one of the following three alert categories: Green, Orange or Red. The rotatable bond is marked Green or Orange for the measured angle of the torsion pattern within the first or second tolerance bins around a torsion peak. Otherwise, it's marked Red implying that the measured angle is not observed in the structure databases employed to generate the torsion library.

The following output files are generated after the filtering:

```
    <OutfileRoot>.sdf
    <OutfileRoot>_Filtered.sdf
    <OutfileRoot>_AlertsSummary.csv
    <OutfileRoot>_Filtered_TopRule*.sdf
```

The supported input file formats are: Mol (.mol), SD (.sdf, .sd)

The supported output file formats are: SD (.sdf, .sd)

OPTIONS

-a, --alertsMode <Red, RedAndOrange> [default: Red]

Torsion library alert types to use for filtering molecules containing rotatable bonds marked with Green, Orange, or Red alerts. Possible values: Red or RedAndOrange.

--alertsMinCount <Number> [default: 1]

Minimum number of rotatable bond alerts in a molecule for filtering the molecule.

-e, --examples

Print examples.

-h, --help

Print this help message.

-i, --infile <infile>

Input file name.

--infileParams <Name,Value,...> [default: auto]

A comma delimited list of parameter name and value pairs for reading molecules from files. The supported parameter names for different file formats, along with their default values, are shown below:

```
SD, MOL: removeHydrogens,no,sanitize,yes,strictParsing,yes
```

-l, --list

List torsion library information without performing any filtering.

-m, --mode <filter or count> [default: filter]

Specify whether to filter molecules for torsion library [Ref 146, 152, 159] alerts by matching rotatable bonds against SMARTS patterns specified for torsion rules and write out the rest of the molecules to an outfile or simply count the number of matched molecules marked for filtering.

--mp <yes or no> [default: no]

Use multiprocessing.

By default, input data is retrieved in a lazy manner via mp.Pool.imap() function employing lazy RDKit data iterable. This allows processing of arbitrary large data sets without any additional requirements memory.

All input data may be optionally loaded into memory by mp.Pool.map() before starting worker processes in a process pool by setting the value of 'inputDataMode' to 'InMemory' in '--mpParams' option.

A word to the wise: The default 'chunkSize' value of 1 during 'Lazy' input data mode may adversely impact the performance. The '--mpParams' section provides additional information to tune the value of 'chunkSize'.

--mpParams <Name,Value,...> [default: auto]

A comma delimited list of parameter name and value pairs to configure multiprocessing.

The supported parameter names along with their default and possible values are shown below:

```
chunkSize, auto
inputDataMode, Lazy   [ Possible values: InMemory or Lazy ]
numProcesses, auto   [ Default: mp.cpu_count() ]
```

These parameters are used by the following functions to configure and control the behavior of multiprocessing: mp.Pool(), mp.Pool.map(), and mp.Pool.imap().

The chunkSize determines chunks of input data passed to each worker process in a process pool by mp.Pool.map() and mp.Pool.imap() functions. The default value of chunkSize is dependent on the value of 'inputDataMode'.

The mp.Pool.map() function, invoked during 'InMemory' input data mode, automatically converts RDKit data iterable into a list, loads all data into memory, and calculates the default chunkSize using the following method as shown in its code:

```
chunkSize, extra = divmod(len(dataIterable), len(numProcesses) * 4)
if extra: chunkSize += 1
```

For example, the default chunkSize will be 7 for a pool of 4 worker processes and 100 data items.

The mp.Pool.imap() function, invoked during 'Lazy' input data mode, employs 'lazy' RDKit data iterable to retrieve data as needed, without loading all the data into memory. Consequently, the size of input data is not known a priori. It's not possible to estimate an optimal value for the chunkSize. The default chunkSize is set to 1.

The default value for the chunkSize during 'Lazy' data mode may adversely impact the performance due to the overhead associated with exchanging small chunks of data. It is generally a good idea to explicitly set chunkSize to a larger value during 'Lazy' input data mode, based on the size of your input data and number of processes in the process pool.

The mp.Pool.map() function waits for all worker processes to process all the data and return the results. The mp.Pool.imap() function, however, returns the the results obtained from worker processes as soon as the results become available for specified chunks of data.

The order of data in the results returned by both mp.Pool.map() and mp.Pool.imap() functions always corresponds to the input data.

-n, --nitrogenLonePairParams <Name,Value,...> [default: auto]

A comma delimited list of parameter name and value pairs to match torsion SMARTS patterns containing non-standard construct 'N_lp' corresponding to nitrogen lone pair.

The supported parameter names along with their default and possible values are shown below:

```
allowHydrogenNbrs, yes   [ Possible values: yes or no ]
planarityTolerance, 1  [Possible values: >=0]
```

These parameters are used during the matching of torsion rules containing 'N_lp' in their SMARTS patterns. The 'allowHydrogensNbrs' allows the use hydrogen neighbors attached to nitrogen during the determination of its planarity. The 'planarityTolerance' in degrees represents the tolerance allowed for nitrogen to be considered coplanar with its three neighbors.

The torsion rules containing 'N_lp' in their SMARTS patterns are categorized into the following two types of rules:

```
TypeOne:

[CX4:1][CX4H2:2]!@[NX3;"N_lp":3][CX4:4]
[C:1][CX4H2:2]!@[NX3;"N_lp":3][C:4]
... ... ...

TypeTwo:

[!#1:1][CX4:2]!@[NX3;"N_lp":3]
[C:1][$(S(=O)=O):2]!@["N_lp":3]
... ... ...
```

The torsions are matched to torsion rules containing 'N_lp' using specified SMARTS patterns without the 'N_lp' along with additional constraints using the following methodology:

```
TypeOne:
```

```
. SMARTS pattern must contain four mapped atoms and the third
   mapped atom must be a nitrogen matched with 'NX3:3'
. Nitrogen atom must have 3 neighbors. The 'allowHydrogens'
   parameter controls inclusion of hydrogens as its neighbors.
. Nitrogen atom and its 3 neighbors must be coplanar.
   'planarityTolerance' parameter provides tolerance in degrees
   for nitrogen to be considered coplanar with its 3 neighbors.

TypeTwo:

. SMARTS pattern must contain three mapped atoms and the third
   mapped atom must be a nitrogen matched with 'NX3:3'. The
   third mapped atom may contain only 'N_lp:3' The missing 'NX3'
   is automatically detected.
. Nitrogen atom must have 3 neighbors. 'allowHydrogens'
   parameter controls inclusion of hydrogens as neighbors.
. Nitrogen atom and its 3 neighbors must not be coplanar.
   'planarityTolerance' parameter provides tolerance in degrees
   for nitrogen to be considered coplanar with its 3 neighbors.
. Nitrogen lone pair position equivalent to VSEPR theory is
   determined based on the position of nitrogen and its neighbors.
   A vector normal to 3 nitrogen neighbors is calculated and added
   to the coordinates of nitrogen atom to determine the approximate
   position of the lone pair. It is used as the fourth position to
   calculate the torsion angle.
```

-o, --outfile <outfile>

Output file name.

--outfileAlerts <yes or no> [default: yes]

Write out alerts information to SD output files.

--outfileAlertsMode <All or AlertsOnly> [default: AlertsOnly]

Write alerts information to SD output files for all alerts or only for alerts specified by '--AlertsMode' option. Possible values: All or AlertsOnly This option is only valid for 'Yes' value of '--outfileAlerts' option.

The following alerts information is added to SD output files using 'TorsionAlerts' data field:

```
RotBondIndices TorsionAlert TorsionIndices TorsionAngle
TorsionAngleViolation HierarchyClass HierarchySubClass
TorsionRule TorsionPeaks Tolerances1 Tolerances2
```

The 'RotBondsCount' and 'TorsionAlertsCount' data fields are always added to SD output files containing both remaining and filtered molecules.

Format:

```
> <RotBondsCount>
Number

> <TorsionAlertsCount (Green Orange Red)>
Number Number Number

> <TorsionAlerts (RotBondIndices TorsionAlert TorsionIndices
   TorsionAngle TorsionAngleViolation HierarchyClass
   HierarchySubClass TorsionPeaks Tolerances1 Tolerances2
   TorsionRule)>
AtomIndex2,AtomIndex3  AlertType AtomIndex1,AtomIndex2,AtomIndex3,
AtomIndex4 Angle AngleViolation ClassName SubClassName
CommaDelimPeakValues CommaDelimTol1Values CommDelimTol2Values
SMARTS ... ... ...
 ... ... ...
```

A set of 11 values is written out as value of 'TorsionAlerts' data field for each torsion in a molecule. The space character is used as a delimiter to separate values with in a set and across set. The comma character is used to delimit multiple values for each value in a set.

The 'RotBondIndices' and 'TorsionIndices' contain 2 and 4 comma delimited values representing atom indices for a rotatable bond and matched torsion. The 'TorsionPeaks', 'Tolerances1', and 'Tolerances2' contain same number of comma delimited values corresponding to torsion angle peaks and tolerance intervals specified in torsion library. For example:

```
... ... ...
>  <RotBondsCount>  (1)
7

>  <TorsionAlertsCount (Green Orange Red)>  (1)
3 2 2

>  <TorsionAlerts (RotBondIndices TorsionAlert TorsionIndices
    TorsionAngle TorsionAngleViolation HierarchyClass
    HierarchySubClass TorsionPeaks Tolerances1 Tolerances2
    TorsionRule)>
1,2 Red 32,2,1,0 0.13 149.87 NC Anilines 180.0 10.0 30.0 [cH0:1][c:2]
([cH,nX2H0])!@[NX3H1:3][CX4:4] 8,9 Red 10,9,8,28 -0.85 GG
None -90.0,90.0 30.0,30.0 60.0,60.0 [cH1:1][a:2]([cH1])!@[a:3]
([cH0])[cH0:4]
... ... ...
```

--outfileFiltered <yes or no> [default: yes]

Write out a file containing filtered molecules. Its name is automatically generated from the specified output file. Default: <OutfileRoot>_ Filtered.<OutfileExt>.

--outfilesFilteredByRules <yes or no> [default: yes]

Write out SD files containing filtered molecules for individual torsion rules triggering alerts in molecules. The name of SD files are automatically generated from the specified output file. Default file names: <OutfileRoot>_ Filtered_TopRule*.sdf

The following alerts information is added to SD output files:

```
> <RotBondsCount>
Number

>  <TorsionAlertsCount (Green Orange Red)>
Number Number Number

>  <TorsionRule (HierarchyClass HierarchySubClass TorsionPeaks
    Tolerances1 Tolerances2 TorsionRule)>
ClassName SubClassName CommaDelimPeakValues CommaDelimTol1Values
CommDelimTol2Values SMARTS ... ... ...
 ... ... ...

> <TorsionRuleAlertsCount (Green Orange Red)>
Number Number Number

>  <TorsionRuleAlerts (RotBondIndices TorsionAlert TorsionIndices
    TorsionAngle TorsionAngleViolation)>
AtomIndex2,AtomIndex3  AlertType AtomIndex1,AtomIndex2,AtomIndex3,
AtomIndex4 Angle AngleViolation ... ... ...

>  <TorsionRuleMaxAngleViolation>
Number
 ... ... ...
```

For example:

```
... ... ...
>  <RotBondsCount>  (1)
7

>  <TorsionAlertsCount (Green Orange Red)>  (1)
```

```
3 2 2

>  <TorsionRule (HierarchyClass HierarchySubClass TorsionPeaks
    Tolerances1 Tolerances2 TorsionRule)>  (1)
NC Anilines 180.0 10.0 30.0 [cH0:1][c:2]([cH,nX2H0])!@[NX3H1:3][CX4:4]

>  <TorsionRuleAlertsCount (Green Orange Red)>  (1)
0 0 1

>  <TorsionRuleAlerts (RotBondIndices TorsionAlert TorsionIndices
    TorsionAngle TorsionAngleViolation)>  (1)
1,2 Red 32,2,1,0 0.13 149.87

>  <TorsionRuleMaxAngleViolation>  (1)
149.87
... ... ...
```

--outfilesFilteredByRulesMaxCount <All or number> [default: 10]

Write out SD files containing filtered molecules for specified number of top N torsion rules triggering alerts for the largest number of molecules or for all torsion rules triggering alerts across all molecules.

--outfileSummary <yes or no> [default: yes]

Write out a CVS text file containing summary of torsions rules responsible for triggering torsion alerts. Its name is automatically generated from the specified output file. Default: <OutfileRoot>_AlertsSummary.csv.

The following alerts information is written to summary text file:

```
TorsionRule, TorsionPeaks, Tolerances1, Tolerances2,
HierarchyClass, HierarchySubClass, TorsionAlertType,
TorsionAlertCount, TorsionAlertMolCount
```

The double quotes characters are removed from SMART patterns before before writing them to a CSV file. In addition, the torsion rules are sorted by TorsionAlertMolCount. For example:

```
"TorsionRule","TorsionPeaks","Tolerances1","Tolerances2",
    "HierarchyClass","HierarchySubClass","TorsionAlertTypes",
    "TorsionAlertCount","TorsionAlertMolCount"
"[!#1:1][CX4H2:2]!@[CX4H2:3][!#1:4]","-60.0,60.0,180.0",
    "20.0,20.0,20.0","30.0,30.0,30.0","CC","None/[CX4:2][CX4:3]",
    "Red","16","11"
... ... ...
```

--outfileSDFieldLabels <Type,Label,...> [default: auto]

A comma delimited list of SD data field type and label value pairs for writing torsion alerts information along with molecules to SD files.

The supported SD data field label type along with their default values are shown below:

```
For all SD files:

RotBondsCountLabel, RotBondsCount
TorsionAlertsCountLabel, TorsionAlertsCount (Green Orange Red)
TorsionAlertsLabel, TorsionAlerts (RotBondIndices TorsionAlert
    TorsionIndices TorsionAngle TorsionAngleViolation
    HierarchyClass HierarchySubClass TorsionPeaks Tolerances1
    Tolerances2 TorsionRule)

For individual SD files filtered by torsion rules:

TorsionRuleLabel, TorsionRule (HierarchyClass HierarchySubClass
    TorsionPeaks Tolerances1 Tolerances2 TorsionRule)
TorsionRuleAlertsCountLabel, TorsionRuleAlertsCount (Green Orange
    Red)
TorsionRuleAlertsLabel, TorsionRuleAlerts (RotBondIndices
    TorsionAlert TorsionIndices TorsionAngle TorsionAngleViolation)
```

```
TorsionRuleMaxAngleViolationLabel, TorsionRuleMaxAngleViolation
```

--outfileParams <Name,Value,...> [default: auto]

A comma delimited list of parameter name and value pairs for writing molecules to files. The supported parameter names for different file formats, along with their default values, are shown below:

```
SD: kekulize,yes
```

--overwrite

Overwrite existing files.

-r, --rotBondsSMARTSMode <NonStrict, SemiStrict,...> [default: SemiStrict]

SMARTS pattern to use for identifying rotatable bonds in a molecule for matching against torsion rules in the torsion library. Possible values: NonStrict, SemiStrict, Strict or Specify. The rotatable bond SMARTS matches are filtered to ensure that each atom in the rotatable bond is attached to at least two heavy atoms.

The following SMARTS patterns are used to identify rotatable bonds for different modes:

```
NonStrict: [!$(*#*)&!D1]-&!@[!$(*#*)&!D1]

SemiStrict:
[!$(*#*)&!D1&!$(C(F)(F)F)&!$(C(Cl)(Cl)Cl)&!$(C(Br)(Br)Br)
&!$(C([CH3])([CH3])[CH3])]-!@[!$(*#*)&!D1&!$(C(F)(F)F)
&!$(C(Cl)(Cl)Cl)&!$(C(Br)(Br)Br)&!$(C([CH3])([CH3])[CH3])]

Strict:
[!$(*#*)&!D1&!$(C(F)(F)F)&!$(C(Cl)(Cl)Cl)&!$(C(Br)(Br)Br)
&!$(C([CH3])([CH3])[CH3])&!$([CD3](=[N,O,S])-!@[#7,O,S!D1])
&!$([#7,O,S!D1]-!@[CD3]=[N,O,S])&!$([CD3](=[N+])-!@[#7!D1])
&!$([#7!D1]-!@[CD3]=[N+])]-!@[!$(*#*)&!D1&!$(C(F)(F)F)
&!$(C(Cl)(Cl)Cl)&!$(C(Br)(Br)Br)&!$(C([CH3])([CH3])[CH3])]
```

The 'NonStrict' and 'Strict' SMARTS patterns are available in RDKit. The 'NonStrict' SMARTS pattern corresponds to original Daylight SMARTS specification for rotatable bonds. The 'SemiStrict' SMARTS pattern is derived from 'Strict' SMARTS patterns for its usage in this script.

You may use any arbitrary SMARTS pattern to identify rotatable bonds by choosing 'Specify' value for '-r, --rotBondsSMARTSMode' option and providing its value via '--rotBondsSMARTSPattern' option.

--rotBondsSMARTSPattern <SMARTS>

SMARTS pattern for identifying rotatable bonds. This option is only valid for 'Specify' value of '-r, --rotBondsSMARTSMode' option.

-t, --torsionLibraryFile <FileName or auto> [default: auto]

Specify a XML file name containing data for torsion library hierarchy or use default file, TorsionLibrary.xml, available in MAYACHEMTOOLS/lib/data directory.

The format of data in local XML file must match format of the data in Torsion Library [Ref 146, 152, 159] file available in MAYACHEMTOOLS data directory.

-w, --workingdir <dir>

Location of working directory which defaults to the current directory.

EXAMPLES

To filter molecules containing any rotatable bonds marked with Red alerts based on torsion rules in the torsion library and write out SD files containing remaining and filtered molecules, and individual SD files for torsion rules triggering alerts along with appropriate torsion information for red alerts, type:

```
% RDKitFilterTorsionLibraryAlerts.py -i Sample3D.sdf -o Sample3DOut.sdf
```

To run the first example for only counting number of alerts without writing out any SD files, type:

```
% RDKitFilterTorsionLibraryAlerts.py -m count -i Sample3D.sdf -o
```

```
        Sample3DOut.sdf
```

To run the first example for filtertering molecules marked with Orange or Red alerts and write out SD files, tye:

```
    % RDKitFilterTorsionLibraryAlerts.py -m Filter --alertsMode RedAndOrange
      -i Sample3D.sdf -o Sample3DOut.sdf
```

To run the first example for filtering molecules and writing out torsion information for all alert types to SD files, type:

```
    % RDKitFilterTorsionLibraryAlerts.py --outfileAlertsMode All
      -i Sample3D.sdf -o Sample3DOut.sdf
```

To run the first example for filtering molecules in multiprocessing mode on all available CPUs without loading all data into memory and write out SD files, type:

```
    % RDKitFilterTorsionLibraryAlerts.py --mp yes -i Sample3D.sdf
     -o Sample3DOut.sdf
```

To run the first example for filtering molecules in multiprocessing mode on all available CPUs by loading all data into memory and write out a SD files, type:

```
    % RDKitFilterTorsionLibraryAlerts.py  --mp yes --mpParams
      "inputDataMode, InMemory" -i Sample3D.sdf  -o Sample3DOut.sdf
```

To run the first example for filtering molecules in multiprocessing mode on specific number of CPUs and chunksize without loading all data into memory and write out SD files, type:

```
    % RDKitFilterTorsionLibraryAlerts.py --mp yes --mpParams
      "inputDataMode,lazy,numProcesses,4,chunkSize,8"  -i Sample3D.sdf
      -o Sample3DOut.sdf
```

To list information about default torsion library file without performing any filtering, type:

```
    % RDKitFilterTorsionLibraryAlerts.py -l
```

To list information about a local torsion library XML file without performing any, filtering, type:

```
    % RDKitFilterTorsionLibraryAlerts.py --torsionLibraryFile
      TorsionLibrary.xml -l
```

AUTHOR

Manish Sud (msud@san.rr.com)

COLLABORATOR

Pat Walters

ACKNOWLEDGMENTS

Wolfgang Guba, Patrick Penner, Levi Pierce

SEE ALSO

RDKitFilterChEMBLAlerts.py, RDKitFilterPAINS.py, RDKitFilterTorsionStrainEnergyAlerts.py, RDKitConvertFileFormat.py, RDKitSearchSMARTS.py

COPYRIGHT

This script uses the Torsion Library jointly developed by the University of Hamburg, Center for Bioinformatics, Hamburg, Germany and F. Hoffmann-La-Roche Ltd., Basel, Switzerland.

The functionality available in this script is implemented using RDKit, an open source toolkit for cheminformatics developed by Greg Landrum.