NAME

RDKitSearchFunctionalGroups.py - Search for functional groups using SMARTS patterns

SYNOPSIS

RDKitSearchFunctionalGroups.py [--combineMatches <yes or no>] [--combineOperator <and or or>] [--groupNamesFile <FileName or auto>] [--infileParams <Name,Value,...>] [--mode <retrieve or count>] [--mp <yes or no>] [--mpParams <Name,Value,...>] [--negate <yes or no>] [--outfileParams <Name,Value,...>] [--overwrite] [--useChirality <yes or no>] [-w <dir>] [-o <outfile>] -i <infile> -f <Name1,Name2,Name3... or All>

RDKitSearchFunctionalGroups.py [--groupNamesFile <FileName or auto>] -l | --list

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

DESCRIPTION

Perform a substructure search in an input file using SMARTS patterns for functional groups and write out the matched molecules to an output file or simply count the number of matches.

The SMARTS patterns for specified functional group(s) are retrieved from file, Functional_Group_Hierarchy.txt, available in RDKit data directory.

The names of valid functional groups and hierarchies are dynamically retrieved from the functional groups hierarchy file and are shown below:

```
AcidChloride, AcidChloride.Aromatic, AcidChloride.Aliphatic
Alcohol, Alcohol.Aromatic, Alcohol.Aliphatic
Aldehyde, Aldehyde.Aromatic, Aldehyde.Aliphatic
Amine, Amine.Primary, Amine.Primary.Aromatic, Amine.Primary.Aliphatic,
Amine.Secondary, Amine.Secondary.Aromatic, Amine.Secondary.Aliphatic
Amine.Tertiary, Amine.Tertiary.Aromatic, Amine.Tertiary.Aliphatic
Amine.Aromatic, Amine.Aliphatic, Amine.Cyclic
Azide, Azide.Aromatic, Azide.Aliphatic
BoronicAcid, BoronicAcid.Aromatic, BoronicAcid.Aliphatic
CarboxylicAcid, CarboxylicAcid.Aromatic, CarboxylicAcid.Aliphatic,
CarboxylicAcid.AlphaAmino
Halogen, Halogen.Aromatic, Halogen.Aliphatic
Halogen.NotFluorine, Halogen.NotFluorine.Aliphatic,
Halogen.NotFluorine.Aromatic
Halogen.Bromine, Halogen.Bromine.Aliphatic, Halogen.Bromine.Aromatic,
Halogen.Bromine.BromoKetone
Isocyanate, Isocyanate.Aromatic, Isocyanate.Aliphatic
Nitro, Nitro.Aromatic, Nitro.Aliphatic,
SulfonylChloride, SulfonylChloride.Aromatic, SulfonylChloride.Aliphatic
TerminalAlkyne
```

The supported input file formats are: SD (.sdf, .sd), SMILES (.smi, .csv, .tsv, .txt)

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

OPTIONS

-c, --combineMatches <yes or no> [default: yes]

Combine search results for matching SMARTS patterns of specified functional groups against a molecule. Possible values: yes or no.

The matched molecules are written to a single output file for "yes" value. Otherwise, multiple output files are generated, one for each functional group. The names of these files correspond to a combination of the basename of the specified output file and the name of the functional group.

No output files are generated during "count" value of "-m, --mode" option.

--combineOperator <and or or> [default: and]

Logical operator to use for combining match results corresponding to specified functional group names before writing out a single file. This option is ignored during "No" value of "-c, --combineMatches" option.

-e, --examples

Print examples.

-g, --groupNamesFile <FileName or auto> [default: auto]

Specify a file name containing data for functional groups hierarchy or use functional group hierarchy file, Functional_Group_Hierarchy.txt, available in RDKit data directory.

RDKit data format: Name<tab>Smarts<tab>Label<tab>RemovalReaction (optional)

The format of data in local functional group hierarchy must match format of the data in functional group file available in RDKit data directory.

-f, --functionalGroups <Name1,Name2,Name3... or All> [default: none]

Functional group names for performing substructure SMARTS search. Possible values: Comma delimited list of valid functional group names or All. The current set of valid functional group names are listed in the description section.

The match results for multiple functional group names are combined using 'and' operator before writing them out to single file. No merging of match results takes place during generation of individual result files corresponding to fictional group names.

The functional group name may be started with an exclamation mark to negate the match result for that fictional group.

-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,yes,sanitize,yes,strictParsing,yes
SMILES: smilesColumn,1,smilesNameColumn,2,smilesDelimiter,space,
    smilesTitleLine,auto,sanitize,yes
```

Possible values for smilesDelimiter: space, comma or tab.

-l, --list

List functional groups information without performing any search.

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

Specify whether to retrieve and write out matched molecules to an output file or simply count the number of matches.

--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.

-o, --outfile <outfile>

Output file name.

--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: compute2DCoords,auto,kekulize,yes
SMILES: smilesKekulize,no,smilesDelimiter,space, smilesIsomeric,yes,
    smilesTitleLine,yes,smilesMolName,yes,smilesMolProps,no
```

Default value for compute2DCoords: yes for SMILES input file; no for all other file types.

--overwrite

Overwrite existing files.

-u, --useChirality <yes or no> [default: no]

Use stereochemistry information for SMARTS search.

-w, --workingdir <dir>

Location of working directory which defaults to the current directory.

EXAMPLES

To list names of all available functional groups along with their SMARTS patterns, type:

```
% RDKitSearchFunctionalGroups.py -l
```

To retrieve molecules containing amine functional group and write out a SMILES file, type:

```
% RDKitSearchFunctionalGroups.py -f Amine -i Sample.smi -o SampleOut.smi
```

To retrieve molecules containing amine functional group, perform search in multiprocessing mode on all available CPUs without loading all data into memory, and write out a SMILES file, type:

```
% RDKitSearchFunctionalGroups.py --mp yes -f Amine -i Sample.smi
  -o SampleOut.smi
```

To retrieve molecules containing amine functional group, perform search in multiprocessing mode on all available CPUs by loading all data into memory, and write out a SMILES file, type:

```
% RDKitSearchFunctionalGroups.py --mp yes --mpParams "inputDataMode,
  InMemory" -f Amine -i Sample.smi -o SampleOut.smi
```

To retrieve molecules containing amine functional group, perform search in multiprocessing mode on specific number of CPUs and chunksize without loading all data into memory, and write out a SMILES file, type:

```
% RDKitSearchFunctionalGroups.py --mp yes --mpParams "inputDataMode,
  lazy,numProcesses,4,chunkSize,8" -f Amine -i Sample.smi -o
  SampleOut.smi
```

To retrieve molecules containing amine functional group but not halogens and carboxylic acid functional groups and write out a SMILES file, type:

```
% RDKitSearchFunctionalGroups.py -f 'Amine,!Halogen,!CarboxylicAcid'
  -i Sample.smi -o SampleOut.smi
```

To retrieve molecules containing amine, halogens or carboxylic acid functional groups and write out a SMILES file, type:

```
% RDKitSearchFunctionalGroups.py -f 'Amine,Halogen,CarboxylicAcid'
  --combineOperator or -i Sample.smi -o SampleOut.smi
```

To retrieve molecules containing amine and carboxylic acid functional groups defined in a local functional groups hierarchy file and write out individual SD files for each funcitonal group, type:

```
% RDKitSearchFunctionalGroups.py -f 'Amine,CarboxylicAcid' -i Sample.sdf
  -g Custom_Functional_Group_Hierarchy.txt --combineMatches No -o SampleOut.sdf
```

To count number of all functional groups in molecules without writing out an output files, type:

```
% RDKitSearchFunctionalGroups.py -m count -f All --combineMatches no -i Sample.smi
```

To retrieve molecule not containing aromatic alcohol and aromatic halogen functional group along with the use of chirality during substructure search and write out individual SMILES files for each functional group, type:

```
% RDKitSearchFunctionalGroups.py --combineMatches no -u yes
   -f '!Alcohol.Aromatic,!Halogen.Aromatic' -i Sample.smi -o SampleOut.smi
```

To retrieve molecule containing amine functional group from a CSV SMILES file, SMILES strings in column 1, name in column 2, and write out a SD file, type:

```
% RDKitSearchFunctionalGroups.py -f Amine --infileParams
  "smilesDelimiter,comma,smilesTitleLine,yes,smilesColumn,1,
  smilesNameColumn,2" --outfileParams "compute2DCoords,yes"
  -i SampleSMILES.csv -o SampleOut.sdf
```

AUTHOR

Manish Sud(msud@san.rr.com)

SEE ALSO

RDKitConvertFileFormat.py, RDKitFilterPAINS.py, RDKitSearchSMARTS.py

COPYRIGHT

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

This file is part of MayaChemTools.

MayaChemTools is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.