NAME

RDKitSearchSMARTS.py - Perform a substructure search using SMARTS pattern

SYNOPSIS

RDKitSearchSMARTS.py [--infileParams <Name,Value,...>] [--mode <retrieve or count>] [--mp <yes or no>] [--mpParams <Name,Value,...>] [--negate <yes or no>] [--outfileFiltered <yes or no>] [--outfileParams <Name,Value,...>] [--overwrite] [--useChirality <yes or no>] [-w <dir>] [-o <outfile>] -p <SMARTS> -i <infile>

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

DESCRIPTION

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

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

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

Possible values for smilesDelimiter: space, comma or tab.

-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.

-n, --negate <yes or no> [default: no]

Specify whether to find molecules not matching the specified SMARTS pattern.

-o, --outfile <outfile>

Output file name.

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

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

--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,forceV3000,no
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.

-p, --pattern <SMARTS> [default: none]

SMARTS pattern for performing search.

-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 retrieve molecules containing the substructure corresponding to a specified SMARTS pattern and write out a SMILES file, type:

```
% RDKitSearchSMARTS.py -p 'c1ccccc1' -i Sample.smi -o SampleOut.smi
```

To retrieve molecules containing the substructure corresponding to a specified SMARTS pattern, perform filtering in multiprocessing mode on all available CPUs without loading all data into memory, and write out a SMILES file, type:

```
% RDKitSearchSMARTS.py --mp yes -p 'c1ccccc1' -i Sample.smi -o SampleOut.smi
```

To retrieve molecules containing the substructure corresponding to a specified SMARTS pattern, perform filtering in multiprocessing mode on all available CPUs by loading all data into memory, and write out a SMILES file, type:

```
% RDKitSearchSMARTS.py --mp yes --mpParams "inputDataMode,InMemory"
  -p 'c1ccccc1' -i Sample.smi -o SampleOut.smi
```

To retrieve molecules containing the substructure corresponding to a specified SMARTS pattern, perform filtering in multiprocessing mode on specific number of CPUs and chunk size without loading all data into memory, and write out a SMILES file, type:

```
% RDKitSearchSMARTS.py --mp yes --mpParams "inputDataMode,Lazy,
  numProcesses,4,chunkSize,8" -p 'c1ccccc1' -i Sample.smi -o SampleOut.smi
```

To only count the number of molecules containing the substructure corresponding to a specified SMARTS pattern without writing out any file, type:

```
% RDKitSearchSMARTS.py -m count -p 'c1ccccc1' -i Sample.smi
```

To count the number of molecules in a SD file not containing the substructure corresponding to a specified SMARTS pattern and write out a SD file, type:

```
% RDKitSearchSMARTS.py -n yes -p 'c1ccccc1' -i Sample.sdf -o SampleOut.sdf
```

To retrieve molecules containing the substructure corresponding to a specified SMARTS pattern from a CSV SMILES file, SMILES strings in column 1, name in and write out a SD file, type:

```
% RDKitSearchSMARTS.py -p 'c1ccccc1' --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, RDKitSearchFunctionalGroups.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.