NAME

RDKitPerformMinimization.py - Perform structure minimization

SYNOPSIS

RDKitPerformMinimization.py [--addHydrogens <yes or no>] [--conformerGenerator <SDG, KDG, ETDG, ETKDG, ETKDGv2, None>] [--forceField <UFF, or MMFF>] [--forceFieldMMFFVariant <MMFF94 or MMFF94s>] [--energyOut <yes or no>] [--enforceChirality <yes or no>] [--infileParams <Name,Value,...>] [--maxConfs <number>] [--maxIters <number>] [--mp <yes or no>] [--mpParams <Name,Value,...>] [--outfileParams <Name,Value,...>] [--overwrite] [--quiet <yes or no>] [--removeHydrogens <yes or no>] [--randomSeed <number>] [-w <dir>] -i <infile> -o <outfile>

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

DESCRIPTION

Generate 3D structures for molecules using a combination of distance geometry and forcefield minimization or minimize existing 3D structures using a specified forcefield.

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

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

OPTIONS

-a, --addHydrogens <yes or no> [default: yes]

Add hydrogens before minimization.

-c, --conformerGenerator <text or None> [default: ETKDGv2]

Conformation generation methodology for generating initial 3D coordinates. The possible values along with a brief description are shown below:

```
SDG: Standard Distance Geometry
KDG: basic Knowledge-terms with Distance Geometry
ETDG: Experimental Torsion-angle preference with Distance Geometry
ETKDG: Experimental Torsion-angle preference along with basic
    Knowledge-terms and Distance Geometry [Ref 129]
ETKDGv2: Experimental Torsion-angle preference along with basic
    Knowledge-terms and Distance Geometry [Ref 167]
None: No conformation generation
```

The conformation generation step may be skipped by specifying 'None' value to perform only forcefield minimization of molecules with 3D structures in input file. This doesn't work for molecules in SMILES file or molecules in SD/MOL files containing 2D structures.

-f, --forceField <UFF, MMFF> [default: MMFF]

Forcefield method to use for energy minimization. Possible values: Universal Force Field (UFF) [Ref 81] or Merck Molecular Mechanics Force Field [Ref 83-87] .

--forceFieldMMFFVariant <MMFF94 or MMFF94s> [default: MMFF94]

Variant of MMFF forcefield to use for energy minimization.

--energyOut <yes or no> [default: No]

Write out energy values.

--enforceChirality <yes or no> [default: Yes]

Enforce chirality for defined chiral centers.

-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.

--maxConfs <number> [default: 250]

Maximum number of conformations to generate for each molecule by conformation generation methodology for initial 3D coordinates. The conformations are minimized using the specified forcefield and the lowest energy conformation is written to the output file. This option is ignored during 'None' value of '-c --conformerGenerator' option.

--maxIters <number> [default: 500]

Maximum number of iterations to perform for each molecule during forcefield minimization.

--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: kekulize,yes,forceV3000,no
```

--overwrite

Overwrite existing files.

-q, --quiet <yes or no> [default: no]

Use quiet mode. The warning and information messages will not be printed.

-r, --removeHydrogens <yes or no> [default: Yes]

Remove hydrogens after minimization.

--randomSeed <number> [default: auto]

Seed for the random number generator for reproducing 3D coordinates. Default is to use a random seed.

-w, --workingdir <dir>

Location of working directory which defaults to the current directory.

EXAMPLES

To generate up to 250 conformations using ETKDG methodology followed by MMFF forcefield minimization for a maximum of 500 iterations for molecules in a SMILES file and write out a SD file containing minimum energy structure corresponding to each molecule, type:

```
% RDKitPerformMinimization.py  -i Sample.smi -o SampleOut.sdf
```

To rerun the first example in a quiet mode and write out a SD file, type:

```
% RDKitPerformMinimization.py  -q yes -i Sample.smi -o SampleOut.sdf
```

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

```
% RDKitPerformMinimization.py --mp yes -i Sample.smi -o SampleOut.sdf
```

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

```
% RDKitPerformMinimization.py --mp yes --mpParams "inputDataMode,
  InMemory" -i Sample.smi -o SampleOut.sdf
```

To run the first example in multiprocessing mode on specific number of CPUs and chunk size without loading all data into memory and write out a SD file, type:

```
% RDKitPerformMinimization.py --mp yes --mpParams "inputDataMode,Lazy,
  numProcesses,4,chunkSize,8" -i Sample.smi -o SampleOut.sdf
```

To generate up to 150 conformations using ETKDG methodology followed by MMFF forcefield minimization for a maximum of 250 iterations along with a specified random seed for molecules in a SMILES file and write out a SD file containing minimum energy structures corresponding to each molecule, type

```
% RDKitPerformMinimization.py  --maxConfs 150  --randomSeed 201780117
```

```
    --maxIters 250  -i Sample.smi -o SampleOut.sdf
```

To minimize structures in a 3D SD file using UFF forcefield for a maximum of 150 iterations without generating any conformations and write out a SD file containing minimum energy structures corresponding to each molecule, type

```
% RDKitPerformMinimization.py  -c None -f UFF --maxIters 150
  -i Sample3D.sdf -o SampleOut.sdf
```

To generate up to 50 conformations using SDG methodology followed by UFF forcefield minimization for a maximum of 50 iterations for molecules in a CSV SMILES file, SMILES strings in column 1, name in column 2, and write out a SD file, type:

```
% RDKitPerformMinimization.py  --maxConfs 50  --maxIters 50 -c SDG
  -f UFF --infileParams "smilesDelimiter,comma,smilesTitleLine,yes,
  smilesColumn,1,smilesNameColumn,2"  -i SampleSMILES.csv
  -o SampleOut.sdf
```

AUTHOR

Manish Sud(msud@san.rr.com)

SEE ALSO

RDKitCalculateRMSD.py, RDKitCalculateMolecularDescriptors.py, RDKitCompareMoleculeShapes.py, RDKitConvertFileFormat.py, RDKitGenerateConformers.py, RDKitPerformConstrainedMinimization.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.