NAME

RDKitGenerateConformers.py - Generate molecular conformations

SYNOPSIS

RDKitGenerateConformers.py [--alignConformers <yes or no>] [--addHydrogens <yes or no>] [--conformerGenerator <SDG, ETDG, KDG, ETKDG>] [--embedRMSDCutoff <number>] [--energyOut <yes or no>] [--enforceChirality <yes or no>] [--energyRMSDCalcMode <RMSD or BestRMSD>] [--energyRMSDCutoff <number>] [--energyRMSDCutoffMode <All or Lowest>] [--energyWindow <number>] [--forceField <UFF, MMFF, None>] [--forceFieldMMFFVariant <MMFF94 or MMFF94s>] [--infileParams <Name,Value,...>] [--maxConfs <number>] [--mp <yes or no>] [--mpParams <Name,Value,...>] [--maxIters <number>] [--outfileParams <Name,Value,...>] [--overwrite] [--quiet <yes or no>] [--removeHydrogens <yes or no>] [--randomSeed <number>] [-w <dir>] -i <infile> -o <outfile>

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

DESCRIPTION

Generate 3D conformes of molecules using a combination of distance geometry and forcefield minimization. The forcefield minimization may be skipped to only generate conformations by available distance geometry based methodologies.

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

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

OPTIONS

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

Add hydrogens before minimization.

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

Align conformers for each molecule.

-c, --conformerGenerator <SDG, ETDG, KDG, ETKDG> [default: ETKDG]

Conformation generation methodology for generating initial 3D coordinates of a molecule. Possible values: Standard Distance Geometry, (SDG), Experimental Torsion-angle preference with Distance Geometry (ETDG) [Ref 129] , basic Knowledge-terms with Distance Geometry (KDG), and Experimental Torsion-angle preference along with basic Knowledge-terms and Distance Geometry (ETKDG).

--embedRMSDCutoff <number> [default: none]

RMSD cutoff for retaining conformations after embedding and before energy minimization. All embedded conformations are kept by default. Otherwise, only those conformations which are different from each other by the specified RMSD cutoff are kept. The first embedded conformation is always retained.

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

Write out energy values.

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

Enforce chirality for defined chiral centers.

--energyRMSDCalcMode <RMSD or BestRMSD> [default: RMSD]

Methodology for calculating RMSD values during the application of RMSD cutoff for retaining conformations after energy minimization. Possible values: RMSD or BestRMSD. This option is ignore during 'None' value of '--energyRMSDCutoff' option.

During BestRMSMode mode, the RDKit 'function AllChem.GetBestRMS' is used to align and calculate RMSD. This function calculates optimal RMSD for aligning two molecules, taking symmetry into account. Otherwise, the RMSD value is calculated using 'AllChem.GetConformerRMS' without changing the atom order. A word to the wise from RDKit documentation: The AllChem.GetBestRMS function will attempt to align all permutations of matching atom orders in both molecules, for some molecules it will lead to 'combinatorial explosion'.

--energyRMSDCutoff <number> [default: none]

RMSD cutoff for retaining conformations after energy minimization. By default, all minimized conformations with in the specified energy window from the lowest energy conformation are kept. Otherwise, only those conformations which are different from the lowest energy conformation or all selected conformations by the specified RMSD cutoff and are with in the specified energy window are

kept. The lowest energy conformation is always retained.

--energyRMSDCutoffMode <All or Lowest> [default: All]

RMSD cutoff mode for retaining conformations after energy minimization. Possible values: All or Lowest. The RMSD values are compared against all the selected conformations or the lowest energy conformation during 'All' and 'Lowest' value of '--energyRMSDCutoffMode'. This option is ignored during 'None' value of '--energyRMSDCutoff' option.

By default, only those conformations which all different from all selected conformations by the specified RMSD cutoff and are with in the specified energy window are kept.

--energyWindow <number> [default: 20]

Energy window in kcal/mol for selecting conformers. This option is ignored during 'None' value of '-f, --forcefield' option.

-e, --examples

Print examples.

-f, --forceField <UFF, MMFF, None> [default: MMFF]

Forcefield method to use for energy minimization. Possible values: Universal Force Field (UFF) [Ref 81], Merck Molecular Mechanics Force Field (MMFF) [Ref 83-87] or None.

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

Variant of MMFF forcefield to use for energy minimization.

-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: auto]

Maximum number of conformations to generate for each molecule by conformation generation methodology. The conformations are minimized using the specified forcefield as needed and written to the output file. The default value for maximum number of conformations is dependent on the number of rotatable bonds in molecules: RotBonds < = 5, maxConfs = 100; RotBonds >=6 and < = 10, MaxConfs = 200; RotBonds >= 11, maxConfs = 300

--maxIters <number> [default: 250]

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
```

--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 conformers using Experimental Torsion-angle preference along with basic Knowledge-terms and Distance Geometry (ETKDG) followed by MMFF minimization with automatic determination of maximum number of conformers for each molecule and write out a SD file, type:

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

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

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

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

```
% RDKitGenerateConformers.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:

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

To rerun 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:

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

To generate up to 150 conformers for each molecule using ETKDG and UFF forcefield minimization along with conformers within 25 kcal/mol energy window and write out a SD file, type:

```
% RDKitGenerateConformers.py  --energyWindow 25 -f UFF --maxConfs 150
  -i Sample.smi -o SampleOut.sdf
```

To generate up to 50 conformers for each molecule using KDG without any forcefield minimization and alignment of conformers and write out a SD file, type:

```
% RDKitGenerateConformers.py  -f none --maxConfs 50 --alignConformers no
  -i Sample.sdf -o SampleOut.sdf
```

To generate up to 50 conformers using SDG without any forcefield minimization and alignment of conformers for molecules in a CSV SMILES file, SMILES strings in column 1, name in column 2, and write out a SD file, type:

```
% RDKitGenerateConformers.py  --maxConfs 50  --maxIters 50 -c SDG
  --alignConformers no -f none --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, RDKitGenerateConstrainedConformers.py, RDKitPerformMinimization.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.